James H Baxter, Yoram Orzach, Charit Mishra

Wireshark Revealed: Essential Skills for IT Professionals

Learning Path

Get up and running with Wireshark to analyze your network effectively

Packt

Wireshark Revealed: Essential Skills for IT Professionals

Table of Contents

Wireshark Revealed: Essential Skills for IT Professionals

Wireshark Revealed: Essential Skills for IT Professionals

Published on: December 2017

Production reference: 1011217

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN - 978-1-78883-322-6

www.packtpub.com

Credits

Authors

James H Baxter

Yoram Orzach

Charit Mishra

Reviewers

Sarath Lakshman

Bruno Vernay

Ms. Samia Yousif

Charles L. Brook

Praveen Darshan

Ritwik Ghoshal

Gilbert Ramirez

Anish Nath

Content Development Editor

Devika Battike

Graphics

Kirk D’penha

Production Coordinator

Aparna Bhagat

Preface

Wireshark is a popular and powerful tool used to analyze the amount of bits and bytes that are flowing through a network. The packet captures displayed in Wireshark give you an insight into the security and flaws of different protocols, which will help you perform the security research and protocol debugging.

What this learning path covers

Module 1, Wireshark Essentials, introduces the Wireshark network analyzer to IT professionals across multiple disciplines.

It starts off with the installation of Wireshark, before gradually taking you through your first packet capture, identifying and filtering out just the packets of interest, and saving them to a new file for later analysis. The subsequent chapters will build on this foundation by covering essential topics on the application of the right Wireshark features for analysis, network protocols essentials, troubleshooting, and analyzing performance issues. Finally, this module focuses on packet analysis for security tasks, command-line utilities, and tools that manage trace files.

Upon finishing this module, you will have successfully added strong Wireshark skills to your technical toolset and significantly increased your value as an IT professional

Module 2, Network Analysis using Wireshark Cookbook, highlights the operations of Wireshark as a network analyzer tool. This book provides you with a set of practical recipes to help you solve any problems in your network using a step-by-step approach.

“Network analysis using Wireshark Cookbook” starts by discussing the capabilities of Wireshark, such as the statistical tools and the expert system, capture and display filters, and how to use them. The book then guides you through the details of the main networking protocols, that is, Ethernet, LAN switching, and TCP/IP, and then discusses the details of application protocols and their behavior over the network. Among the application protocols that are discussed in the book are standard Internet protocols like HTTP, mail protocols, FTP, and DNS, along with the behavior of databases, terminal server clients, Citrix, and other applications that are common in the IT environment.

In a bottom-up troubleshooting approach, the book goes up through the layers of the OSI reference model explaining how to resolve networking problems. The book starts from Ethernet and LAN switching, through IP, and then on to TCP/UDP with a focus on TCP performance problems. It also focuses on WLAN

security. Then, we go through application behavior issues including HTTP, mail, DNS, and other common protocols. The book finishes with a look at network forensics and how to search and find security problems that might harm the network.

Module 3, Mastering Wireshark, will help you raise your knowledge to an expert level. At the start of this module, you will be introduced to its interface so you understand all its functionalities. Moving forward, you will discover different ways to create and use capture and display filters. Halfway through the book, you'll be mastering the features of Wireshark, analyzing different layers of the network protocol, looking for any anomalies. As you reach to the end of the book, you will be taught how to use Wireshark for network security analysis and configure it for troubleshooting purposes.

What you need for this learning path

The primary requirement is as follows:

- You will need to install the Wireshark software that can be downloaded from www.wireshark.org.

Who this learning path is for

This book is aimed at IT professionals who want to develop or enhance their packet analysis skills. A basic familiarity with common network and application services terms and technologies is assumed.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail <feedback@packtpub.com>, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this course from your account at http://www.packtpub.com. If you purchased this course elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at :

https://github.com/PacktPublishing/Wireshark-Revealed-Essential-skills-for-IT-professionals

We also have other code bundles from our rich catalog of books, videos, and courses available at https://github.com/PacktPublishing/. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting http://www.packtpub.com/submit-errata, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to https://www.packtpub.com/books/content/support and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at <questions@packtpub.com>, and we will do our best to address the problem.

Part 1. Module 1

Wireshark Essentials

Get up and running with Wireshark to analyze network packets and protocols effectively

Chapter 1. Getting Acquainted with Wireshark

Since its creation in 1997 by Gerald Combs to troubleshoot network problems at a small ISP, Wireshark (originally called Ethereal) has now become one of the most popular tools available for packet-level analysis of network and application protocols. This is mostly because it is an open source solution, which makes it freely available to any technical professional, as well as its extensive range of features, coverage of over 1000 protocols, and the continued support and improvements made possible by contributions from over 800 developers around the globe.

This introductory chapter will help you to quickly become proficient in Wireshark by installing it on your system and doing something fun and useful with it, before diving into more details and supporting concepts.

In this chapter, we will cover the following topics:

- Installing Wireshark
- Performing a packet capture
- Wireshark user interface essentials
- Using display filters to isolate traffic of interest
- Saving a filtered packet trace file

The chapters that follow will build on and provide the supporting concepts for these basic functions to allow you to develop the Wireshark skills that are most applicable to your technical role and objectives.

Installing Wireshark

Wireshark can be installed on machines running 32- and 64-bit Windows (XP, Win7, Win8.1, and so on), Mac OS X (10.5 and higher), and most flavors of Linux/Unix. Installation on Windows and Mac machines is quick and easy because installers are available from the Wireshark website download page. Wireshark is a standard package available on many Linux distributions, and there is a list of links to third-party installers provided on the Wireshark download page for a variety of popular *nix platforms. Alternatively, you can download the source code and compile Wireshark for your environment if a precompiled installation package isn't available.

Wireshark relies on the WinPcap (Windows) or libpcap (Linux/Unix/Mac) libraries to provide the packet capture and capture filtering functions; the appropriate library is installed during the Wireshark installation.

Note

You might need administrator (Windows) or root (Linux/Unix/Mac) privileges to install Wireshark and the WinPcap/libpcap utilities on your workstation.

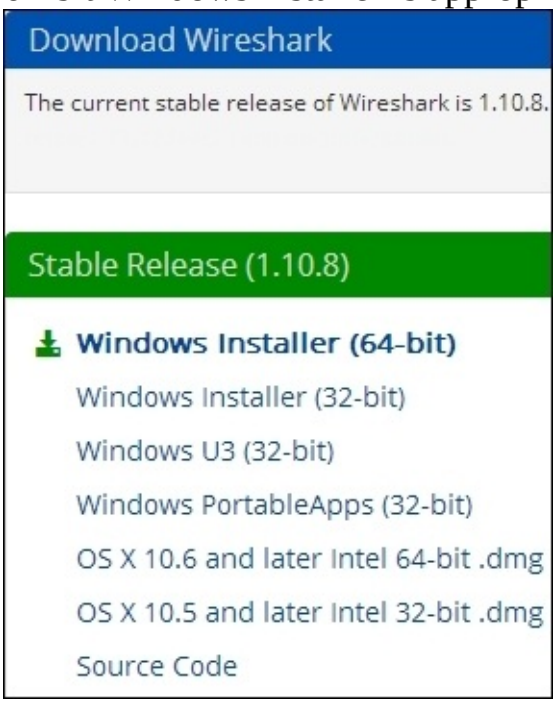
Assuming that you're installing Wireshark on a Windows or Mac machine, you need to go to the Wireshark website (https://www.wireshark.org/) and click on the **Download** button at the top of the page. This will take you to the download page, and at the same time attempt to perform an autodiscovery of your operating system type and version from your browser info. The majority of the time, the correct Wireshark installation package for your machine will be highlighted, and you only have to click on the highlighted link to download the correct installer.

Note

If you already have Wireshark installed, an autoupdate feature will notify you of available version updates when you launch Wireshark.

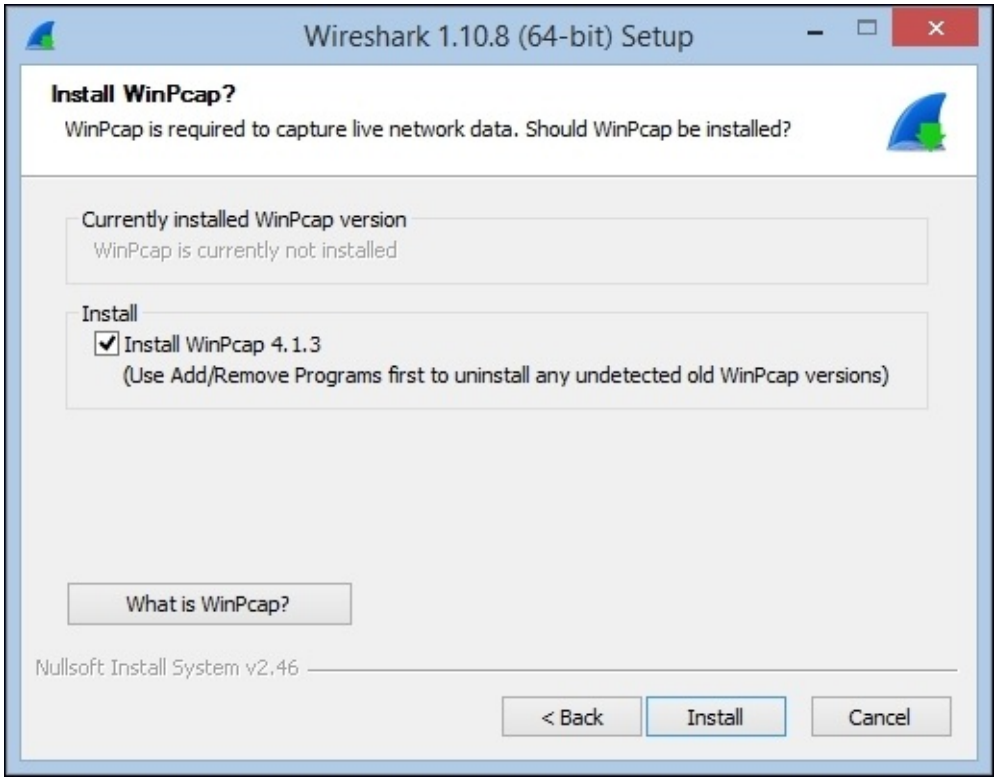
Installing Wireshark on Windows

In the following screenshot, the Wireshark download page has identified that a 64-bit Windows installer is appropriate for this Windows workstation:



Clicking on the highlighted link downloads a `Wireshark-win64-1.10.8.exe` file or similar executable file that you can save on your hard drive. Double-clicking on the executable starts the installation process. You need to follow these steps:

1. Agree to the **License Agreement**.
2. Accept all of the defaults by clicking on **Next** for each prompt, including the prompt to install WinPcap, which is a library needed to capture packets from the **Network Interface Card** (**NIC**) on your workstation.
3. Early in the Wireshark installation, the process will pause and prompt you to click on **Install** and several **Next** buttons in separate windows to install WinPcap.
4. After the WinPcap installation is complete, click through the remaining **Next** prompts to finish the Wireshark installation.


Wireshark 1.10.8 (64-bit) Setup
Install WinPcap?
WinPcap is required to capture live network data. Should WinPcap be installed?
Currently installed WinPcap version
WinPcap is currently not installed
Install
Install WinPcap 4.1.3
(Use Add/Remove Programs first to uninstall any undetected old WinPcap versions)
What is WinPcap?
Nullsoft Install System v2.46
< Back
Install
Cancel

Installing Wireshark on Mac OS X

The process to install Wireshark on Mac is the same as the process for Windows, except that you will not be prompted to install WinPcap; libpcap, the packet capture library for Mac and *nix machines, gets installed instead (without prompting).

There are, however, two additional requirements that may need to be addressed in a Mac installation:

- The first is to install X11, a windowing system library. If this is needed for your system, you will be informed and provided a link that ultimately takes you to the XQuartz project download page so you can install this package.
- The second requirement that might come up is if upon starting Wireshark, you are informed that there are no interfaces on which a capture can be done. This is a permissions issue on the **Berkeley packet filter** (**BPF**) that can be resolved by opening a terminal window and typing the following command:

  ```
  bash-3.2$ sudo chmod 644 /dev/bpf*
  ```

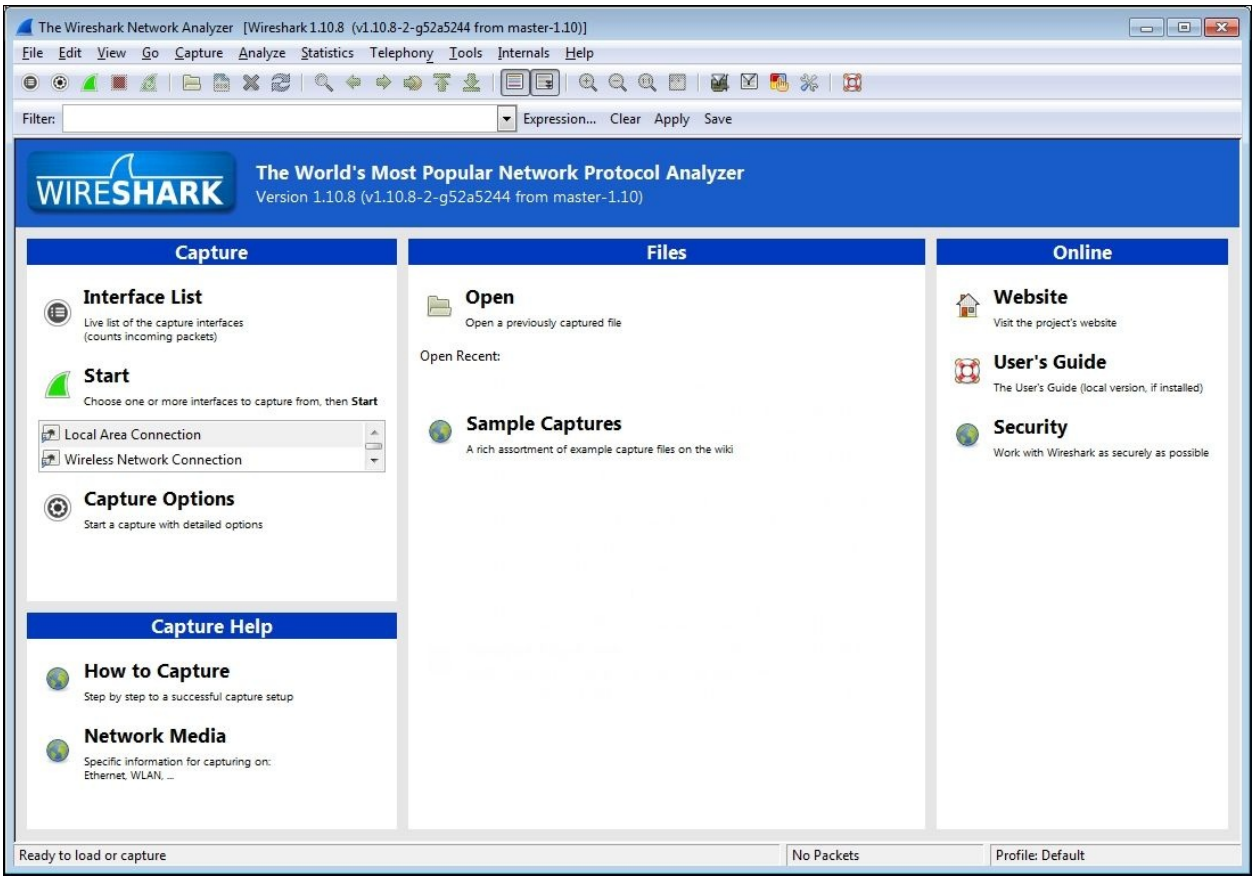
If this process needs to be repeated each time you start Wireshark, you can perform a web search for a more permanent permissions solution for your environment.

Installing Wireshark on Linux/Unix

The requirements and process to install Wireshark on a Linux or Unix platform can vary significantly depending on the particular environment. Wireshark is usually available by default through the package management systems for your specific Linux distribution. Guidance to install Wireshark on Linux can be found in Chapter 2, *Networking for Packet Analysts*, or in the Wireshark user documentation located at www.wireshark.org/docs/wsug_html_chunked/ChapterBuildInstall.html.

Performing your first packet capture

When you first start Wireshark, you are presented with an initial **Start Page** as shown in the following screenshot:

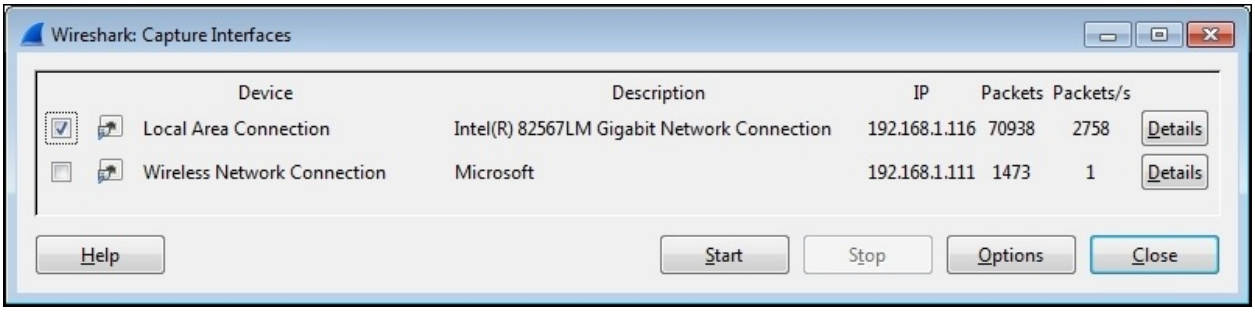


Don't get too fond of this screen. Although you'll see this every time you start Wireshark, once you do a capture, open a trace file, or perform any other function within Wireshark, this screen will be replaced with the standard Wireshark user interface and you won't see it again until the next time you start Wireshark. So, we won't spend much time here.

Selecting a network interface

If you have a number of network interfaces on your machine, you may not be sure which one to select to capture packets, but there's a fairly easy way to figure this out. On the Wireshark start page, click on **Interface List** (alternatively, click on **Interfaces** from the **Capture** menu or click on the first icon on the icon bar).

The **Wireshark Capture Interfaces** window that opens provides a list and description of all the network interfaces on your machine, the IP address assigned to each one (if an address has been assigned), and a couple of counters, such as the total number of packets seen on the interface since this window opened and a packets/s (packets per second) counter. If an interface has an IPv6 address assigned (which may start with `fe80::` and contain a number of colons) and this is being displayed, you can click on the IPv6 address and it will toggle to display the IPv4 address. This is shown in the following screenshot:



Note

On Linux/Unix/Mac platforms, you might also see a loopback interface that can be selected to capture packets being sent between applications on the same machine. However, in most cases, you'll only be interested in capturing packets from a network interface.

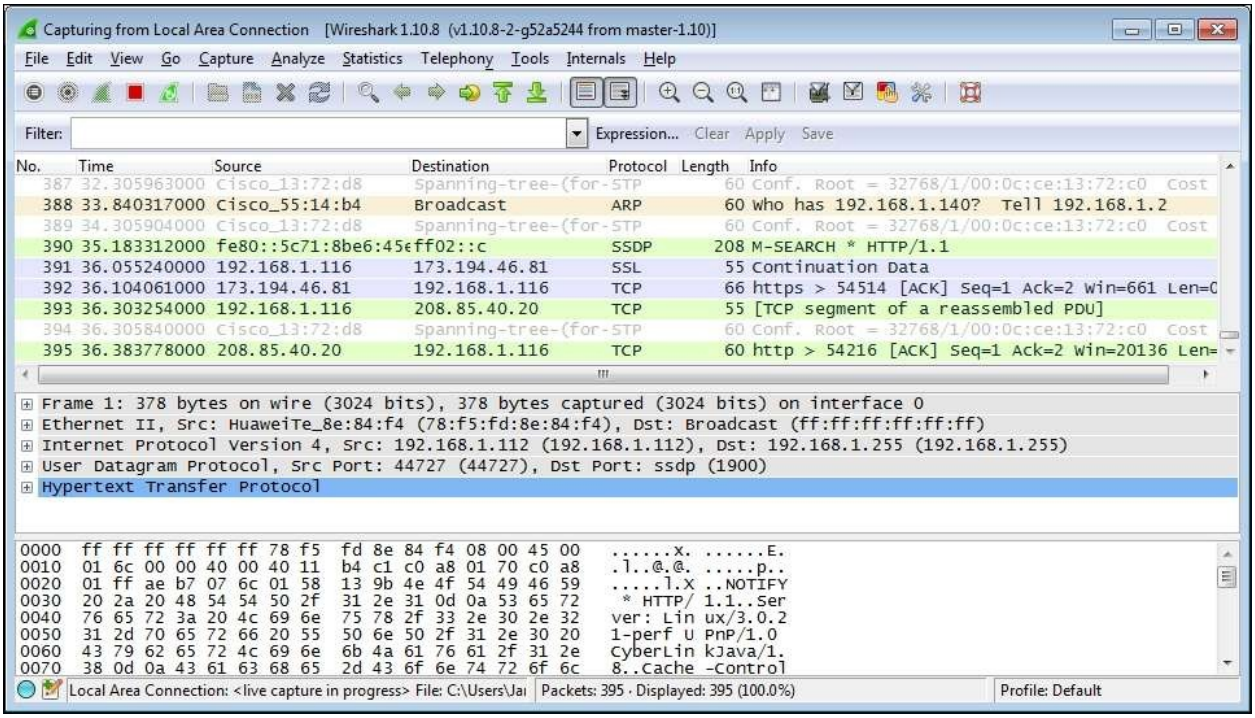
The goal is to identify the active interface that will be used to communicate with the Internet when you open a browser and navigate to a website. If you have a wired local area network connection and the interface is enabled, that's probably the active interface, but you might also have a wireless interface that is enabled and you may or may not be the primary interface. The most reliable indicator of

the active network interface is that it will have greater number of steadily increasing packets with a corresponding active number of packets/s (which will vary over time). Another possible indicator is if an interface has an IP address assigned and others do not. If you're still unsure, open a browser window and navigate to one of your favorite websites and watch the packets and packets/s counters to identify the interface that shows the greatest increase in activity.

Performing a packet capture

Once you've identified the correct interface, select the checkbox on the left-hand side of that interface and click on the **Start** button at the bottom of the **Capture Interfaces** window. Wireshark will start capturing all the packets that can be seen from that interface, including the packets sent to and from your workstation. You'll see a bewildering variety of packets going by in the top section (called the **Packet List** pane) of the screen; this is normal. If you don't see this, try a different interface.

It's a bit amazing just how much background traffic there is on a typical network, such as broadcast packets from devices advertising their names, addresses, and services to and from other devices asking for addresses of stations they want to communicate with. Also, a fair amount of traffic is generated from your own workstation for applications and services that are running in the background, and you had no idea they were creating this much noise. Your Wireshark's **Packet List** pane may look similar to the following screenshot; however, we can ignore all this for now:



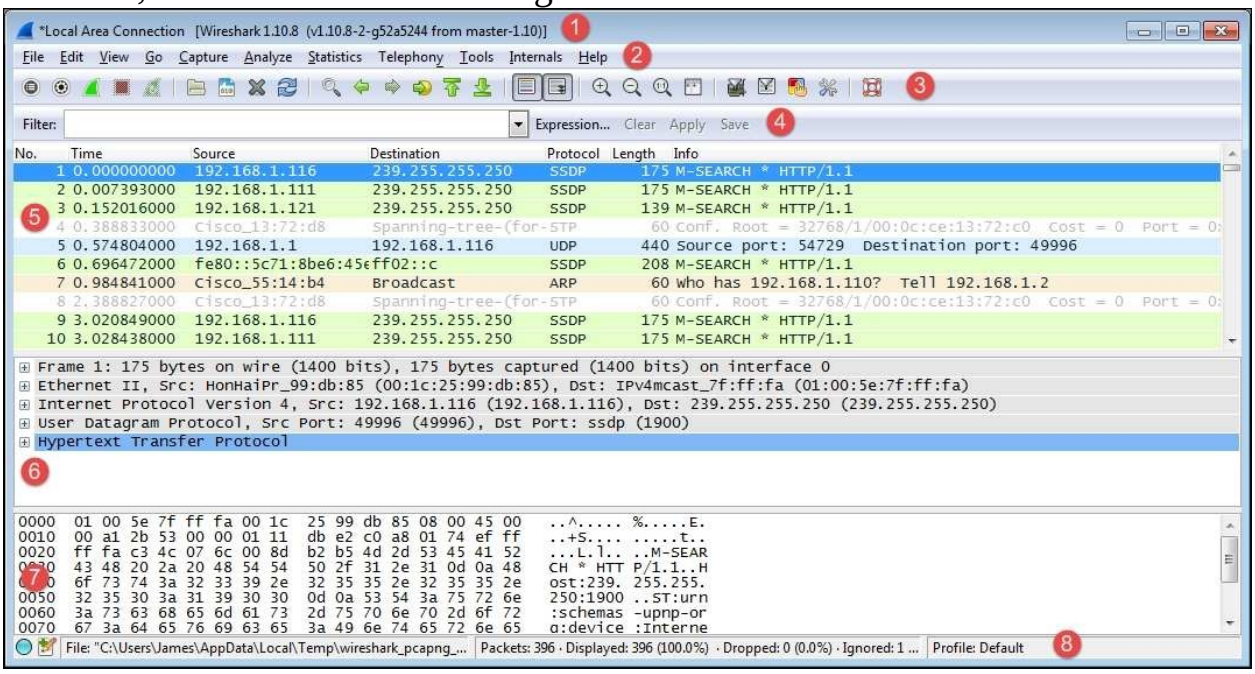
We're ready to generate some traffic that we'll be interested in analyzing. Open a new Internet browser window, enter `www.wireshark.org` in the address box, and press *Enter*.

When the https://www.wireshark.org/ home page finishes loading, stop the Wireshark capture by either selecting **Stop** from the **Capture** menu or by clicking on the red square stop icon that's between the **View** and **Go** menu headers.

Wireshark user interface essentials

Once you have completed your first capture, you will see the normal Wireshark user interface main screen. So before we go much further, a quick introduction to the primary parts of this user interface will be helpful so you'll know what's being referred to as we continue the analysis process.

There are eight significant sections or elements of the default Wireshark user interface, as shown in the following screenshot:



Let's look at the eight significant sections in detail:

- **Title**: This area reflects the interface from where a capture is being taken or the filename of an open packet trace file
- **Menu**: This is the standard row of main functions and subfunctions in Wireshark
- **Main toolbar (icons)**: These provide a quick way to access the most useful Wireshark functions and are well worth getting familiar with and using
- **Display filter toolbar**: This allows you to quickly create, edit, clear, apply, and save filters to isolate packets of interest for analysis
- **Packet list pane**: This section contains a summary info line for each

captured packet, as well as a packet number and relative timestamp

- **Packet details pane**: This section provides a hierarchical display of information about a single packet that has been selected in the packet list pane, which is divided into sections for the various protocols contained in a packet
- **Packet bytes pane**: This section displays the selected packets' contents in hex bytes or bits form, as well as an ASCII display of the data that can be helpful
- **Status bar**: This section provides an expert info indicator, edit capture comments icon, trace file path name and size information, data on the number of packets captured and displayed and other info, and a profile display and selection section

Filtering out the noise

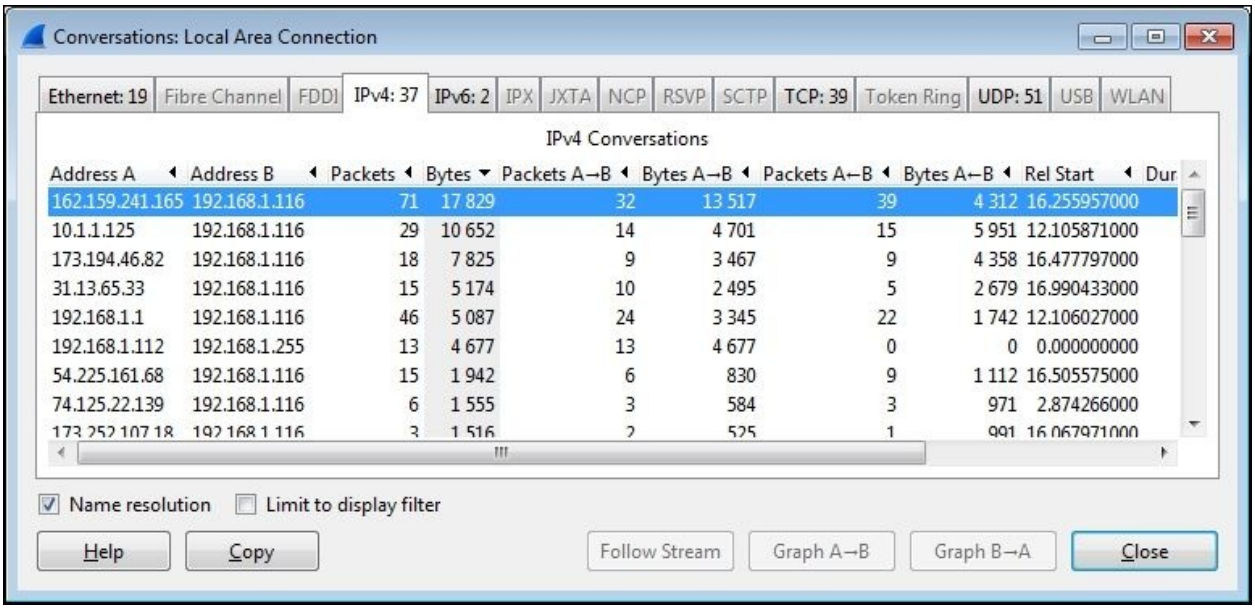
Somewhere in your packet capture, there are packets involved with loading the Wireshark home page—but how do you find and view just those packets out of all the background noise?

The simplest and most reliable method is to determine the IP address of the Wireshark website and filter out all the packets except those flowing between that IP address and the IP address of your workstation by using a display filter. The best approach—and the one that you'll likely use as a first step for most of your post-capture analysis work in future—is to investigate a list of all the conversations by IP address and/or hostnames, sorted by the most active nodes, and identify your target hostname, website name, or IP address from this list.

From the Wireshark menu, select **Conversations** from the **Statistics** menu, and in the **Conversations** window that opens, select the **IPv4** tab at the top. You'll see a list of network conversations identified by **Address A** and **Address B**, with columns for total **Packets**, **Bytes**, **Packets A → B**, **Bytes A → B**, **Packets A ← B**, and **Bytes A ← B**.

Scrolling over to the right-hand side of this window, there are **Relative Start** values. These are the times when each particular conversation was first observed in the capture, relative to the start of the capture in seconds. The next column is **Duration**, which is how long this conversation persisted in the capture (first to last packet seen).

Finally, there are average data rates in **bits per second** (**bps**) in each direction for each conversation, which is the network impact for this conversation. All these are shown in the following screenshot:



We want to sort the list of conversations to get the busiest ones—called the Top Talkers in network jargon—at the top of the list. Click on the **Bytes** column header and then click on it again. Your list should look something like the preceding screenshot, and if you didn't get a great deal of other background traffic flowing to/from your workstation, the traffic from https://www.wireshark.org/ should have the greatest volume and therefore be at the top of the list.

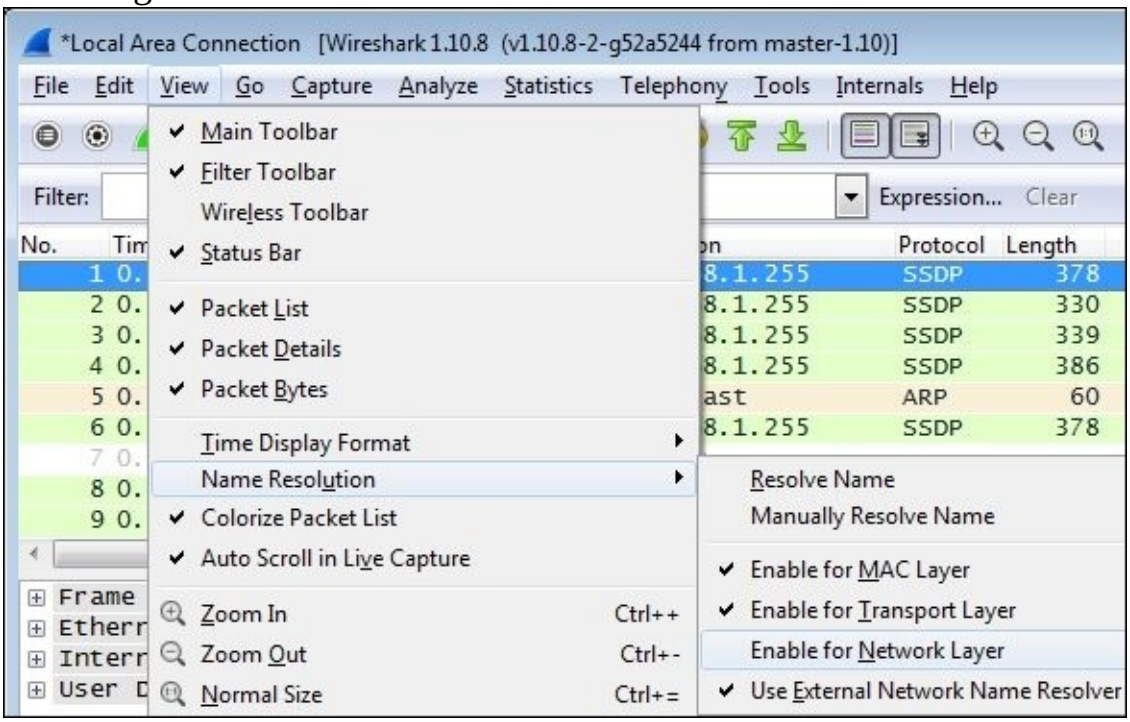
In this example, the conversation between IP addresses **162.159.241.165** and **192.168.1.116** has the greatest overall volume, and looking at the **Bytes A->B** column, it's apparent that the majority of the traffic was from the **162.159.241.165** address to the **192.168.1.116** address. However, at this point, how do we know if this is really the conversation that we're after?

We will need to resolve the IP addresses from our list to hostnames or website addresses, and this can be done from within Wireshark by turning on **Network Name Resolution** and trying to get hostnames and/or website addresses resolved for those IP addresses using reverse DNS queries (using what is known as a pointer (PTR) DNS record type). If you just installed or started Wireshark, the **Name Resolution** option may not be turned on by default.

This is usually a good thing, as Wireshark can create traffic of its own by transmitting the DNS queries trying to resolve all the IP addresses that it comes across during the capture, and you don't really want that going on during a capture. However, the **Name Resolution** option can be very helpful to resolve IP addresses to proper hostnames after a capture is complete.

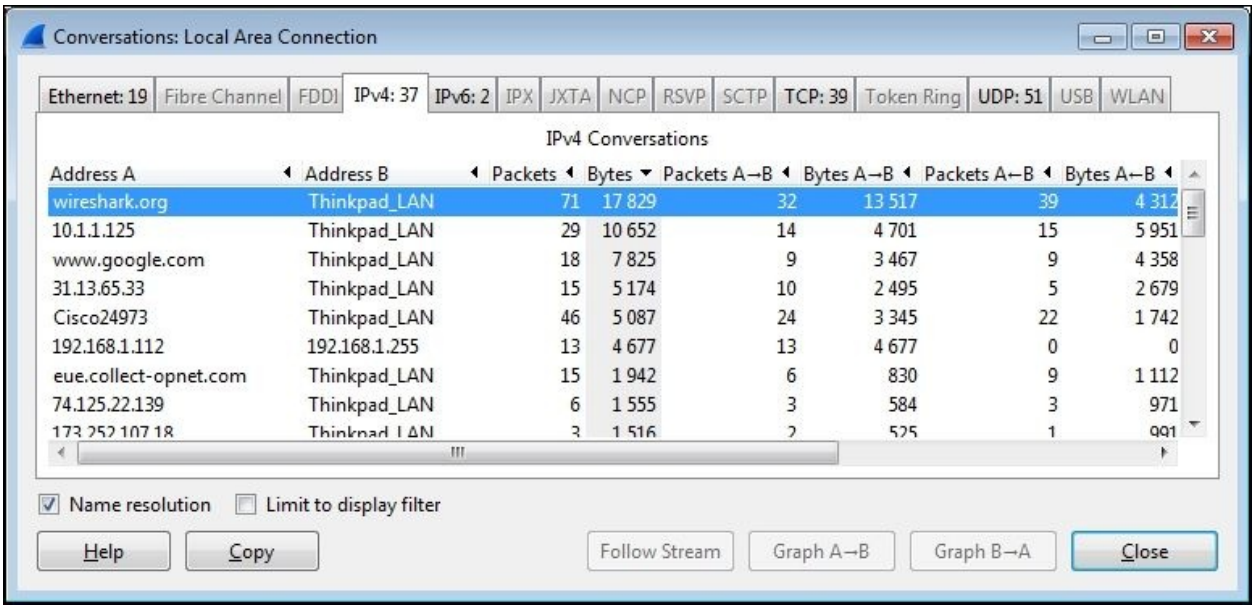
To enable **Name Resolution**, navigate to **View** | **Name Resolution** | **Enable for Network Layer** (click to turn on the checkmark) and make sure **Use External Network Name Resolver** is enabled as well. Wireshark will attempt to resolve all the IP addresses in the capture to their hostname or website address, and the resolved names will then appear (replacing the previous IP addresses) in the packet list as well as the **Conversations** window.

Note that the **Name Resolution** option at the bottom of the **Conversations** window must be enabled as well (it usually is by default), and this setting affects whether resolved names or IP addresses appear in the **Conversations** window (if **Name Resolution** is enabled in the Wireshark main screen), as shown in the following screenshot:



At this point, you should see the conversation pair between **wireshark.org** and

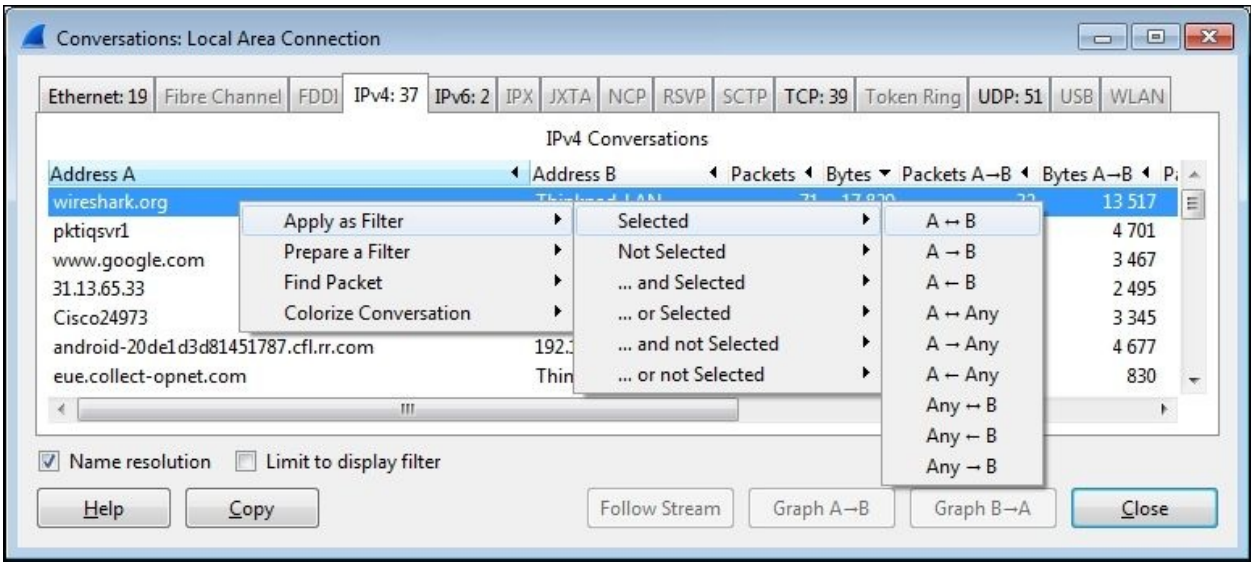
your workstation at or near the top of the list, as shown in the following screenshot. Of course, your workstation will have a different name or may only appear as an IP address, but identifying the conversation to **wireshark.org** has been achieved.



Applying a display filter

You now want to see just the conversation between your workstation and **wireshark.org**, and get rid of all the extraneous conversations so you can focus on the traffic of interest. This is accomplished by creating a filter that only displays the desired traffic.

Right-click on the line containing the **wireshark.org** entry and navigate to **Apply as Filter** | **Selected** | **A<->B**, as shown in the following screenshot:



Wireshark will create and apply a display filter string that isolates the displayed traffic to just the conversation between the IP addresses of **wireshark.org** and your workstation, as shown in the following screenshot. Note that if you create or edit a display filter entry manually, you will need to click on **Apply** to apply the filter to the trace file (or **Clear** to clear it).

This particular display filter syntax works with IP addresses, not with hostnames, and uses an `ip.addr==` (IP address equals) syntax for each node along with the `&&` (and) logic operator to build a string that says `display any packet that contains this IP address *and* that IP address`. This is the type of display filter that you will be using a great deal for packet analysis.

You'll notice as you scroll up and down in the **Packet List** pane that all the other packets, except those between your workstation and **wireshark.org**, are gone.

They're not gone in the strict sense, they're just hidden—as you can observe by inspecting the **Packet No.** column, there are gaps in the numbering sequence; those are for the hidden packets.

Saving the packet trace

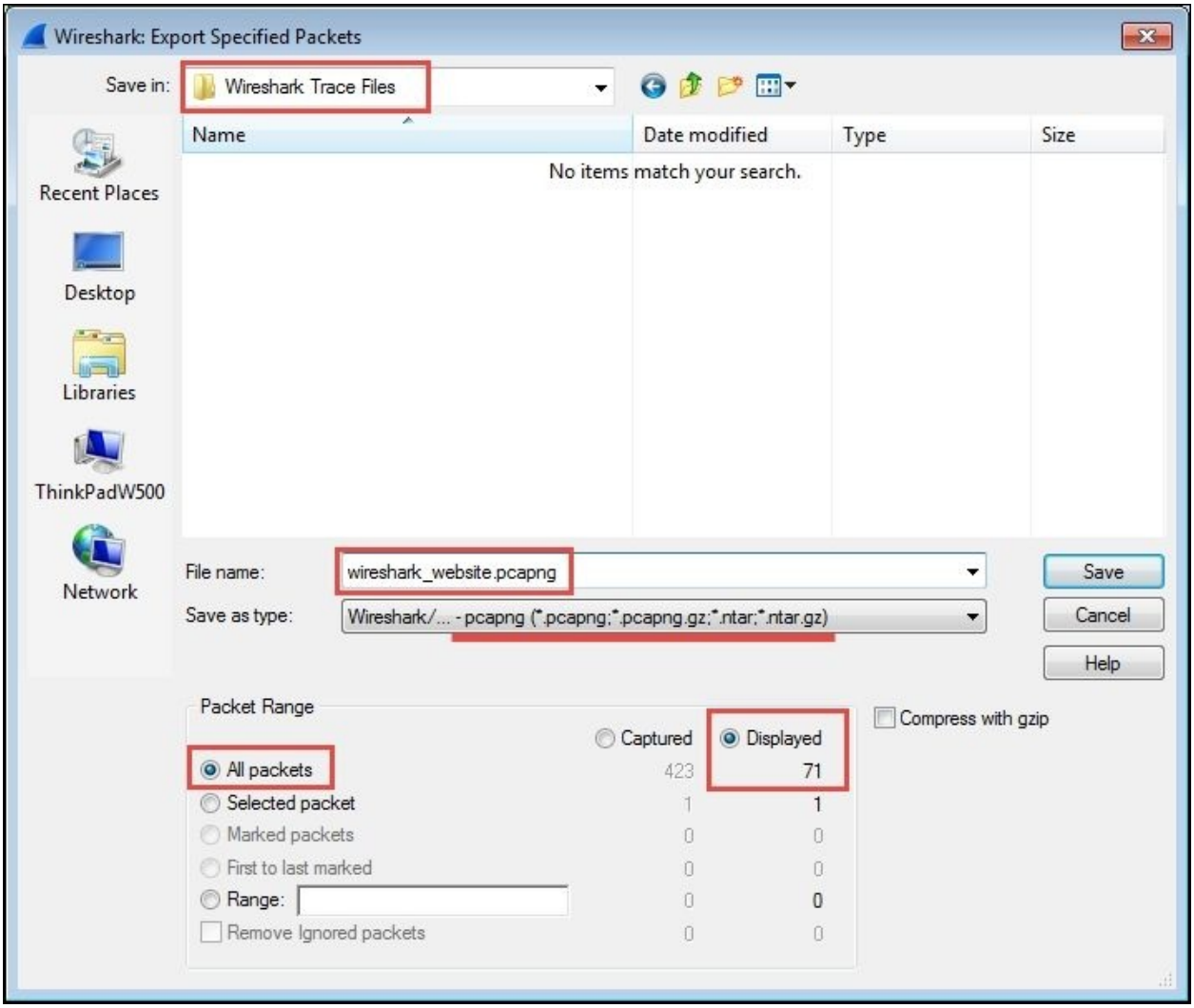
Now that you've isolated the traffic of interest using a display filter, you can save a new packet trace file that contains just the filtered packets.

This serves two purposes. Firstly, you can close Wireshark, come back to it later, open the filtered trace file, and pick up where you left off in your analysis, as well as have a record of the capture in case you need to reference it later such as in a troubleshooting scenario.

Secondly, it's much easier and quicker to work in the various Wireshark screens and functions with a smaller, more focused trace file that contains just the packets that you want to analyze.

To create a new packet trace file containing just the filtered/displayed packets, select **Export Specified Packets** from the Wireshark **File** menu.

You can navigate to and/or create a folder to hold your Wireshark trace files, and then enter a filename for the trace file that you want to save. In this example, the filename is `wireshark_website.pcapng`. By default, Wireshark will save the trace file in the `pcapng` format (which is the preferred format). If you don't specify a file extension with the filename, Wireshark will provide the appropriate extension based on the **Save as type** selection, as shown in the following screenshot:



Also, by default, Wireshark will have the **All packets** option selected, and if a display filter is applied (as it is in this scenario), the **Displayed** option will be selected as opposed to the **Captured** option that saves all the packets regardless of whether a filter was applied. Having entered a filename and confirmed that all the save selections are correct, you can click on **Save** to save the new packet trace file.

Note that when you have finished this trace file save activity, Wireshark still has all the original packets from the capture in memory, and they can still be viewed by clicking on **Clear** in the **Display Filter Toolbar** menu. If you want to work further with the new trace file you just saved, you'll need to open it by clicking

on **Open** in the **File** menu (or **Open Recent** in the **File** menu).

Summary

Congratulations! If you accomplished all the activities covered in this chapter, you have successfully installed Wireshark, performed a packet capture, created a filter to isolate and display just the packets you were interested in from all the extraneous noise, and created a new packet trace file containing just those packets so you can analyze them later. Moreover, in the process, you gained an initial familiarity with the Wireshark user interface and you learned how to use several of its most useful and powerful features. Not bad for a first chapter.

In the next chapter, we'll review some essential network concepts needed to provide a solid foundation to perform packet-level analysis. The main goal of the next chapter is to help you develop a mental model of networking that lends itself well to packet-level analysis without getting too tangled up in unnecessary details.

Chapter 2. Networking for Packet Analysts

Packet analysis is all about analyzing how applications transfer useful data from point A to point B over networks. So, an understanding of how networks function is essential.

In this chapter, we will cover the following topics:

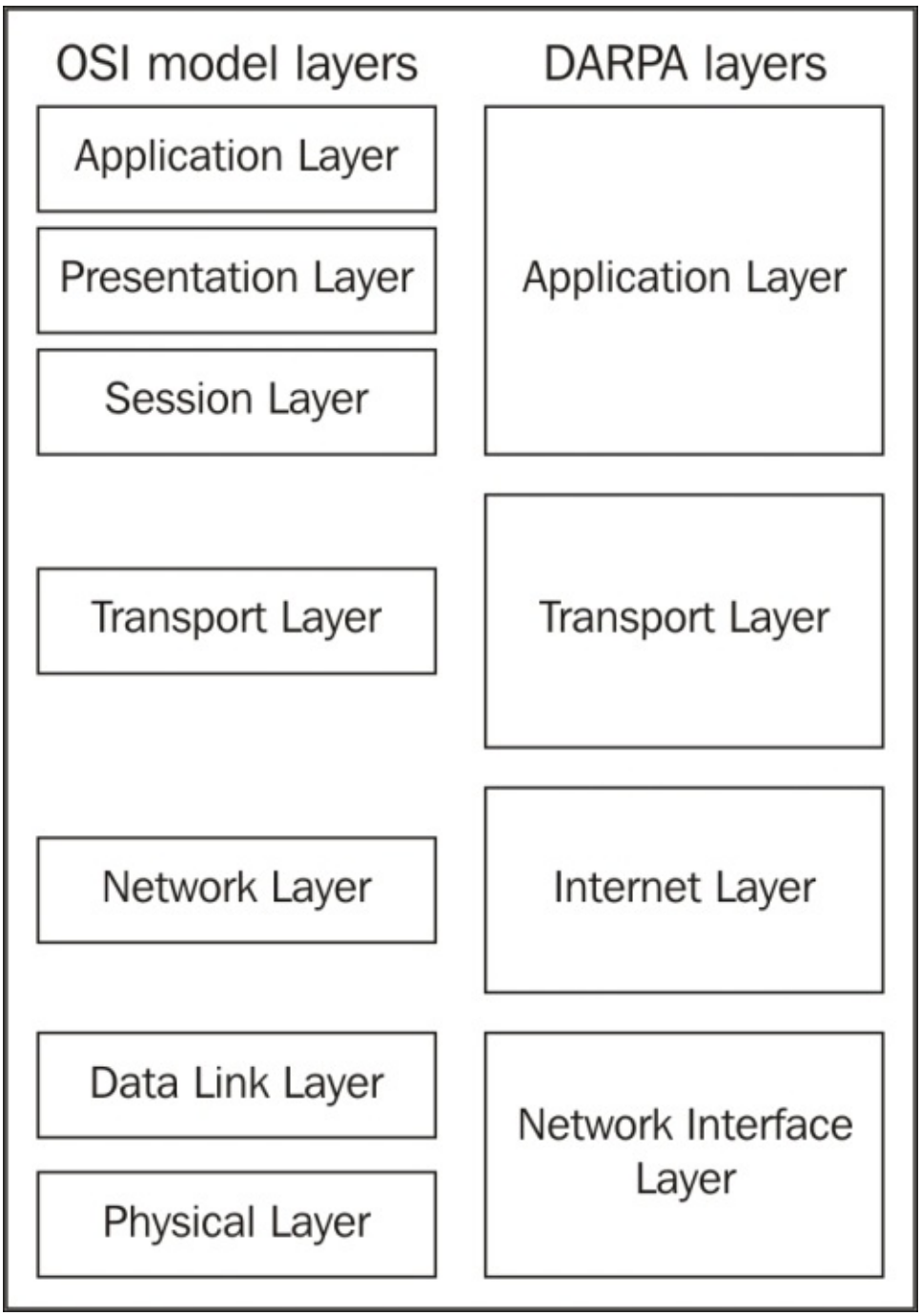
- Why the seven-layer OSI model matters
- IP networks and subnets
- Switching and routing packets
- Ethernet frames and switches
- IP addresses and routers
- WAN links
- Wireless networking

The seven-layer OSI model will be mapped to the most common networking terms, and we'll review frames, switching, IP addressing, routing, and a few other networking topics of interest. The goal is to develop a mental model of networking that lends itself well to packet-level analysis.

The OSI model – why it matters

The **Open Systems Interconnections** (**OSI**) reference model is an industry recognized standard developed by the **International Organization for Standardization** (**ISO**) to divide networking functions into seven logical layers to support and encourage (relatively) independent development while providing (relatively) seamless interconnectivity between each layer from different hardware/software environments, platforms, and vendors. There's also a somewhat simpler four-layer **Defense Advanced Research Projects Agency** (**DARPA**) model that maps to the OSI model, but the OSI version is the most commonly referred to. I'll reference both models when discussing the various layers.

The following diagram compares the OSI and DARPA reference models:



Unless you're in the business of writing protocols, there's no need to study any of the seven layers in great depth, but it is helpful to understand them conceptually because these layers are referred to by the industry and your IT peers.

More importantly, it's essential that you know where and how these layers and their associated protocols are presented in Wireshark's **Packet Details** pane. We'll cover the layers from this aspect to help you remember them and get the most use from the discussion.

Understanding network protocols

Network protocols, like the OSI layers, are a set of industry standard rules and designs used to exchange messages and data between computers and applications. In any discussion about OSI layers, you are directly or indirectly referring to the protocols associated with a given layer—the most commonly known protocols are IP, UDP, TCP, HTTP, and so on—and the significant functions they perform.

For example, you'll often hear the terms network layer and IP layer used interchangeably, and it is assumed and understood that you are talking about the layer and the associated protocol that contains and uses IP addresses to route packets from point A to point B across the network. The discussions that follow will tie the OSI and DARPA layers to their associated protocols.

The seven OSI layers

As we cover the OSI layers starting from layer 1 and working up to layer 7, I'll outline how each layer's associated protocol(s) are displayed in Wireshark and/or used in networking hardware. The mental model you develop from this approach should be the most accurate and useful for packet analysis.

Layer 1 – the physical layer

The physical layer encompasses the electrical characteristics and mechanical standards to get data bits transmitted from a computer's **Network Interface Card** (**NIC**) to a switch port or between switch and router ports. The most common standards, terms, and devices you'll encounter at this layer include the following:

- **Ethernet**: This is a family of networking technologies for **local area networks** (**LANs**).
- **RJ-45**: These are 8-pin modular connectors found on both ends of a copper Ethernet cable that are plugged into the NIC on a computer and a wall jack or switch port
- **Cat 5 (Cat 5e or Cat 6) cables**: These are Ethernet cables that use twisted-pair copper wires. "Cat" stands for the category of cable and reflects its quality and data speed capabilities.
- **100Base-T, 1000Base-T, and 1000Base-LX**: These represent a particular Ethernet standard. 100Base-T is 100 Mbps over twisted-pair cable using RJ-45 connectors, 1000Base-LX is 1000 Mbps over fiber, and so on.
- **Single-mode and multimode fiber optic cables**: These use pulses of light from solid-state LEDs or lasers to transmit data bits.

The Ethernet standards used to connect NICs to switches are also used to connect switches together and to connect switches to routers or other network devices, although the cables and connectors used may vary depending on cable type and speed.

There are other layer 1 standards in common use, including 802.11 Wireless, Frame Relay, and ATM; the last two are used in long distance **wide area networks** (**WANs**).

Layer 2 – the data-link layer

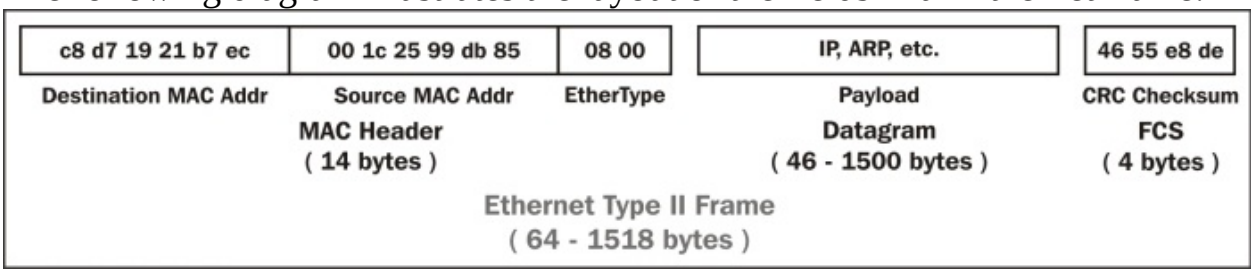
The data-link layer organizes raw bits from the physical layer (typically Ethernet) into frames, which is the first manifestation of what is generally called a packet that you'll see in Wireshark. This layer is a dividing line between physical networking, electrical/mechanical standards, and the logical structures (protocols) used to format and transmit, receive, and decode packets of data in the higher layers.

In the DARPA reference model, the physical and data-link OSI layers are combined and called the network interface layer. The significant features and functions of this layer (for Ethernet II frames) include:

- **Media Access Control (MAC) addresses**: These are the network addresses used in LANs. They are 6-byte network hardware addresses burned into memory chips on NICs, switches, routers, or other network device ports/interfaces:
    - The first three bytes of a MAC address are assigned to and can be associated with a specific manufacturer. Wireshark has a list of these and can display MAC addresses as a combination of the manufacturer code and the last three bytes. The manufacturer creates a unique last-three-bytes address for every interface so that each MAC address is unique across the globe. (Although, an NIC might be programmed to use another arbitrary MAC address, which is done for MAC spoofing for malicious attacks. But this is a very bad idea as another card may be using the same address and can cause a loss of data and some very confusing packet switching problems.)
    - Ethernet frames include a destination and source MAC address. MAC addresses are used to switch (not route—we'll make the distinction shortly) frames between computers on the same LAN or between computers and a router or other device port on a LAN.
- **Type (or EtherType) field**: This indicates the next higher protocol layer (typically IP (0800) or ARP (0806)). Wireshark uses this to determine the next protocol dissector to apply in packet decodes.
- **Payload**: This is the packet or datagram carried by the Ethernet frame.
- **The frame check sequence**: This is a 4-byte **Cyclic Redundancy Check** (**CRC**) error-detection code calculated from all the bits in a frame and added to the end of the frame. This is used to detect frames that have been corrupted usually because of faulty cables, noise induced on the wires in a cable from outside electrical signals, and so on. When a frame is received, this code is

recalculated based on the bits received and compared to the FCS field. The bad frames are then discarded.

The following diagram illustrates the layout of the fields in an Ethernet frame:



A key point here—and this is important to understand—is that Ethernet frames and their MAC addresses are only able to transmit frames between devices on the local area network (LAN and IP subnet) they belong to.

Routers form the boundary between LANs by virtue of their IP subnet (subnetwork) addressing. All the devices belonging to the same IP subnet are part of the same LAN, and getting packets to or from a different subnet requires a router.

Once a frame enters a router port to get routed to a different/distant network, the Ethernet frame with its MAC addresses and FCS is stripped off and discarded. The payload inside the frame is routed to the port and it will leave on its way to the next device, and another frame with a different MAC address and recalculated FCS is created to encase the packet. This frame is then transmitted to the next destination.

The network devices that work at this layer—usually switches—are commonly referred to as layer 2 devices or layer 2 switches.

Finally, you should be aware that layer 2 switches can support several networking enhancements such as **Virtual LAN** (**VLAN**) and **Class of Service** (**CoS**) tagging, which is accomplished by adding a 4-byte 802.1Q field between the MAC addresses and EtherType field. You might see these frames between switches (but not on user ports).

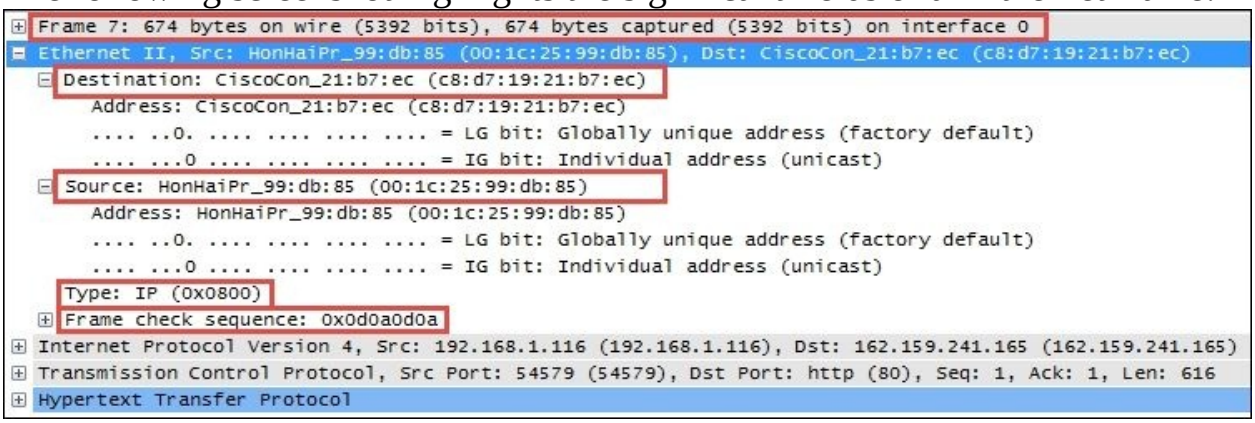
VLAN is a layer 2 solution that allows administrative partitioning of various ports on a switch into separate broadcast domains. Devices located on different VLANs are effectively isolated from each other as if they were on separate physical networks. VLANs can be dispersed across multiple switches without running separate cables for each VLAN if the switches support VLAN tagging. Communication between devices on separate VLANs generally requires using a router.

In the following Wireshark packet details screenshot, the Ethernet II frame **Destination** and **Source** MAC addresses, **Type** (indicating that the next layer protocol is IP), and **Frame check sequence** are circled, as is the **Frame** summary.

Note

Wireshark displays a summary of each frame that includes frame sizes, captured timestamps and interframe times, and other useful information. This is metadata calculated by Wireshark to aid in analysis and not a part of the captured frame.

The following screenshot highlights the significant fields of an Ethernet frame:



Note

Any additional analysis provided by Wireshark in any area of the **Packet Details** pane that is calculated or otherwise not part of actual packet contents will be encased in brackets.

Layer 3 – the network layer

The network layer (called the Internet layer in the DARPA model) primarily handles the routing of packets across and to other networks along the path from source computers to destination hosts based on the destination IP address. The two most common protocols seen at this layer are Internet Protocol and Address Resolution Protocol.

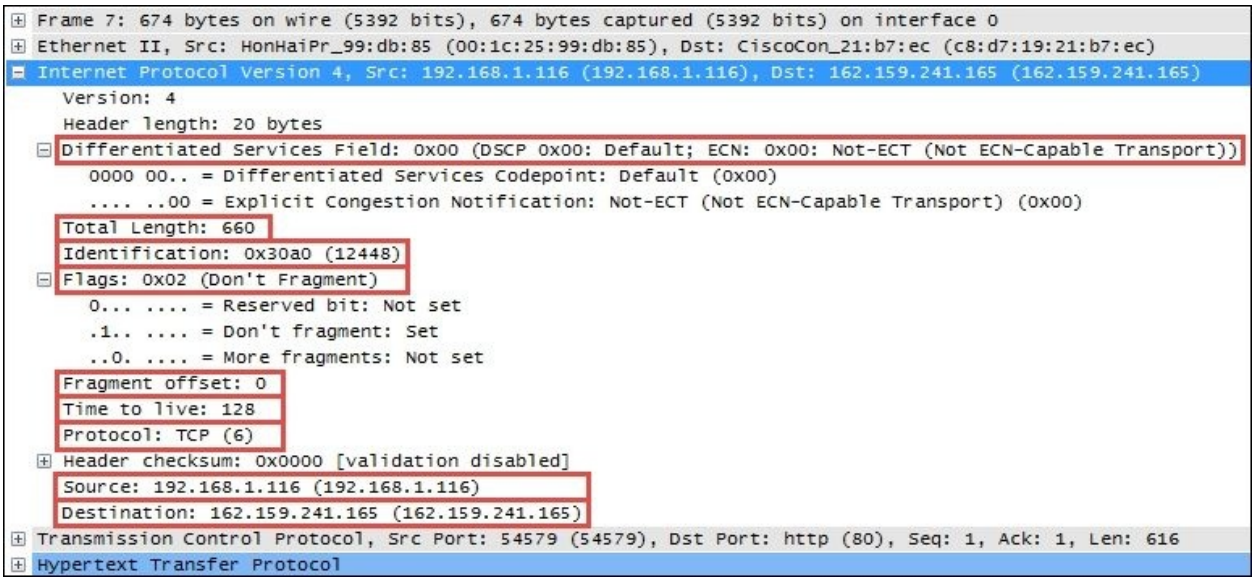
Internet Protocol

The most common protocol in use at this layer is **Internet Protocol Version 4** (**IPv4**), which includes several essential fields to accomplish the task of routing packets across networks:

- **Differentiated Services (DiffServ)**: This field supports an enhancement to the IP that is generally called **Quality of Service** (**QoS**) and allows classification of certain types of traffic (voice, video, and so on) so that these packets can receive priority handling in cases of network congestion.
- **Total length**: This is the total length of the packet (minus the Ethernet MAC header).
- **Identification (IP ID)**: This an incrementing number used to support fragmentation.
- **Flags**: These are used to support fragmenting (dividing a packet into two or more smaller ones) in case the large packets have to be divided into several smaller ones to traverse a packet-size-limited link. These flags along with the IP ID field values allow proper reassembly of the fragmented packets into the original.
- **Fragment offset**: If the **Flag** field is **1** (more fragments), the value in this field indicates the offset from the start of the original payload in bytes that this fragment packet contains.
- **Time to Live (TTL)**: This is a "hop" or time counter that is decremented every time a packet passes through a router. If the TTL reaches zero, the packet is discarded. The primary purpose is to keep packets from living forever and crashing the network in the case of an inadvertent path loop.
- **Protocol**: This identifies the protocol in the IP packet's payload. Wireshark uses this to determine the next protocol dissector to apply to packet decodes.
- **Source and destination IP addresses**: These are the IP addresses of the sending machine and the ultimate destination machine. IP addresses are 4 bytes long and are represented as four octets (numbered 0 through 255

decimal) separated by periods.

In the following screenshot, the significant IPv4 fields are circled. These are the fields you'll want to inspect and be comfortable with when doing packet analysis at this layer.



Address Resolution Protocol

Another protocol you'll see at the network layer is **Address Resolution Protocol** (**ARP**), which is used by a device to obtain the MAC address of another device when it only knows that device's IP address.

In the following Wireshark packet details screenshot, note that the Ethernet frame destination MAC address is **Broadcast (ff:ff:ff:ff:ff:ff)**, **Type** is **ARP (0x0806)**, and the station has provided its own MAC and IP address in the ARP protocol **Sender** fields (which other stations listen to and use to build a table of MAC and IP addresses). It provides the IP address of the target device and puts all zeros in the **Target MAC Address** field. The target device should return a similar ARP packet addressed to the requestor with its MAC address in the **Sender** field.

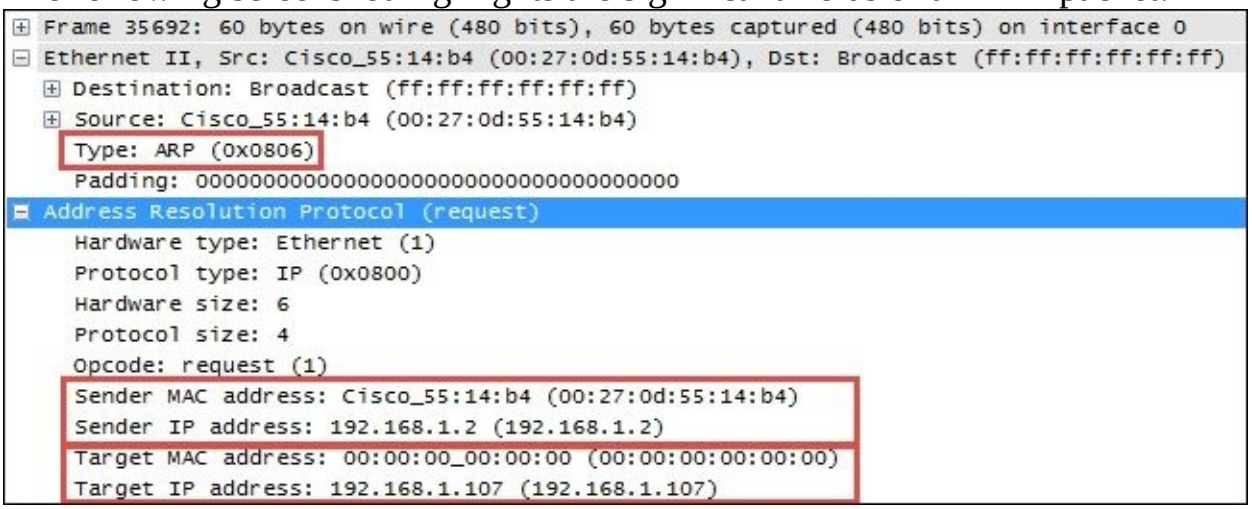
A station will send an ARP request only in the following situations:

- The station that requires a MAC address for a target device hasn't heard a

previous broadcast of that station's MAC address, or its ARP table has timed out (ARP entries are only kept for a period).

- The station that requires a MAC address for a target device has calculated (from the target's IP address and its own subnet mask) that the target device should be on the same LAN. Otherwise, the station assumes the target device is on a different network and sends its first session initiation packet to the default gateway (router) MAC address based on the entry in the sending station's default gateway configuration setting. The default gateway will forward the packet to the appropriate egress port to route it to the destination.
- The station that needs to send a packet to a distant network doesn't know the MAC address of its default gateway (for example, just after a power-up).

The following screenshot highlights the significant fields of an ARP packet:



Other protocols utilized at this layer include **Internet Control Message Protocol** (**ICMP**), which is used to send network error messages between devices, and **Internet Group Management Protocol** (**IGMP**), which is used by hosts and adjacent routers to establish multicast (one-to-many) group memberships for network applications such as streaming video and gaming.

Layer 4 – the transport layer

The transport layer, as it's called in both the OSI and DARPA models, is

responsible for transporting packets of data in unique sessions between applications or a user and an application by means of port numbers. The combination of a device or user's IP address and that device or user's assigned port number (referred to as a socket) will be different from another devices or users' IP address and port numbers (on the client side).

If the source host for a packet is a server, the source port is likely to be a well-known number for standard applications and services, such as port 80 for HTTP.

The transport layer typically uses one of two protocols, User Datagram Protocol or Transmission Control Protocol, with the latter being the more prevalent for most applications.

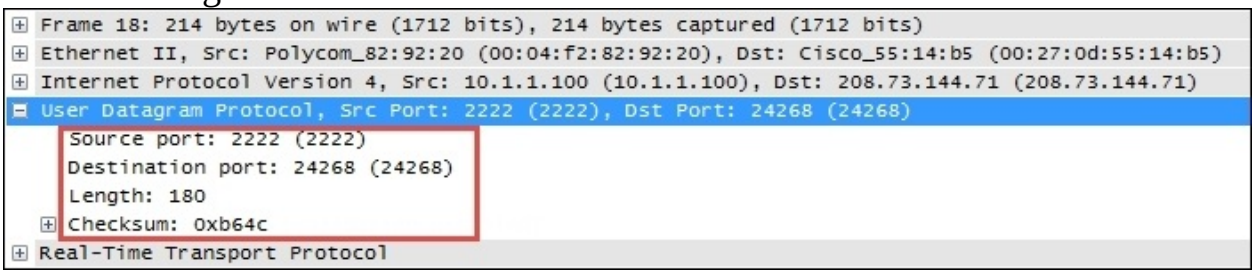
User Datagram Protocol

The **User Datagram Protocol** (**UDP**) is a fairly simple protocol. It is considered an *unreliable* transport as there's no guarantee of packet delivery or ordering, but it has lower overhead and is used by time-sensitive applications such as voice and video traffic, as well as by network services applications such as DNS.

The UDP header is only 8 bytes long and consists of the following:

- **Source and Destination port number**:These are 2 bytes each.
- **Length**: This is the length of the UDP header plus the payload. This is a 2-byte field.
- **Checksum**: This is the 2-byte field used to check errors of the UDP header and data. If no checksum was generated by the transmitter, this will be all zeros.

The following screenshot shows the fields contained in a UDP header:



Transmission Control Protocol

Unlike UDP, the **Transmission Control Protocol** (**TCP**) provides reliable delivery of data by detecting lost, duplicated, or out-of-order packets, requesting retransmission of lost data, or rearranging packets in the right order before delivering them to the application. TCP can also accept a large chunk of data from an application and handle getting the data transported to the other end reliably using multiple packets and reassembling them at the other end (as can UDP, but not reliably; the application has to determine and recover from lost packets).

The TCP header contents and length can vary depending on the options that may be in use, but in its simplest implementation, it consists of:

- **Source and Destination ports (2 bytes each)**: These are well-known registered ports that are used (on servers) to access standard application services such as HTTP, FTP, SMTP, databases, and so on. Port numbers assigned to client/user sessions are usually in a higher number range and assigned sequentially.
- **Sequence number (4 bytes)**: This is a number that represents the first octet in any given segment. Sequence numbers are initialized at the beginning of new sessions as a random number, and then incremented as data bytes and sent.
- **Acknowledgment number (4 bytes)**: When the ACK flag bit is set, this field contains the next sequence number expected from the sender, which in turn acknowledges receipt of all the bytes received up to that point.

  ## Note

  The use of sequence and acknowledgment numbers are how the TCP ensures reliable delivery of data by tracking the number and order of received bytes.

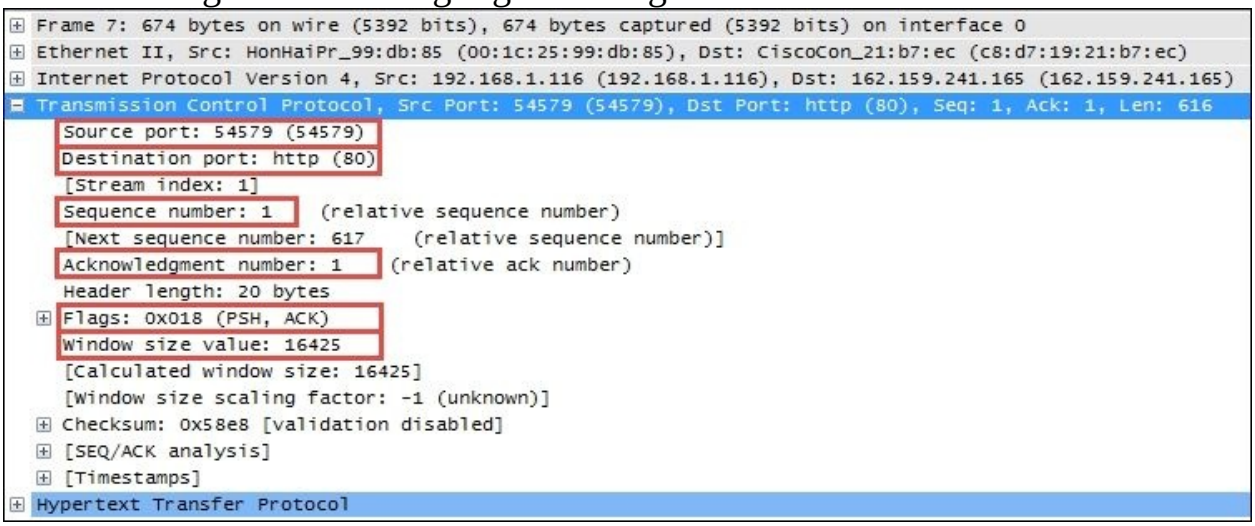
  Sequence and acknowledgment numbers are large and difficult for humans to follow; Wireshark can convert and display these as relative values that start with 0 at the beginning of a session to make it easier to inspect them and relate the values to the number of bytes transmitted and received.

- **Flags (9 bits)**: These bits are used to control connection setups,

terminations, and flow control mechanisms.

- **Window size (2 bytes)**: This indicates the current size of the buffer on this host used to store received data until it can be handed off to the receiving application. This information lets the sending host adjust data flow rates in case of network or host congestion.

The following screenshot highlights the significant fields of a TCP header:



Layer 5 – the session layer

The session layer handles setting up, controlling, and ending sessions within an application between two computers. This is not necessarily the same thing as, for example, a TCP connection, although the two will be related. The application sessions can span and outlive multiple network connections. An example of a networking protocol that operates at this layer is **Network Basic Input/Output System** (**NetBIOS**).

Layer 6 – the presentation layer

The presentation layer converts incoming and outgoing data from one format to another and handles encryption/decryption and/or compression if any of these are required. The presentation layer is also responsible for the delivery and formatting of information to the application layer for further processing or display. An example of a presentation service would be the conversion of an EBCDIC-coded text computer file to an ASCII-coded file.

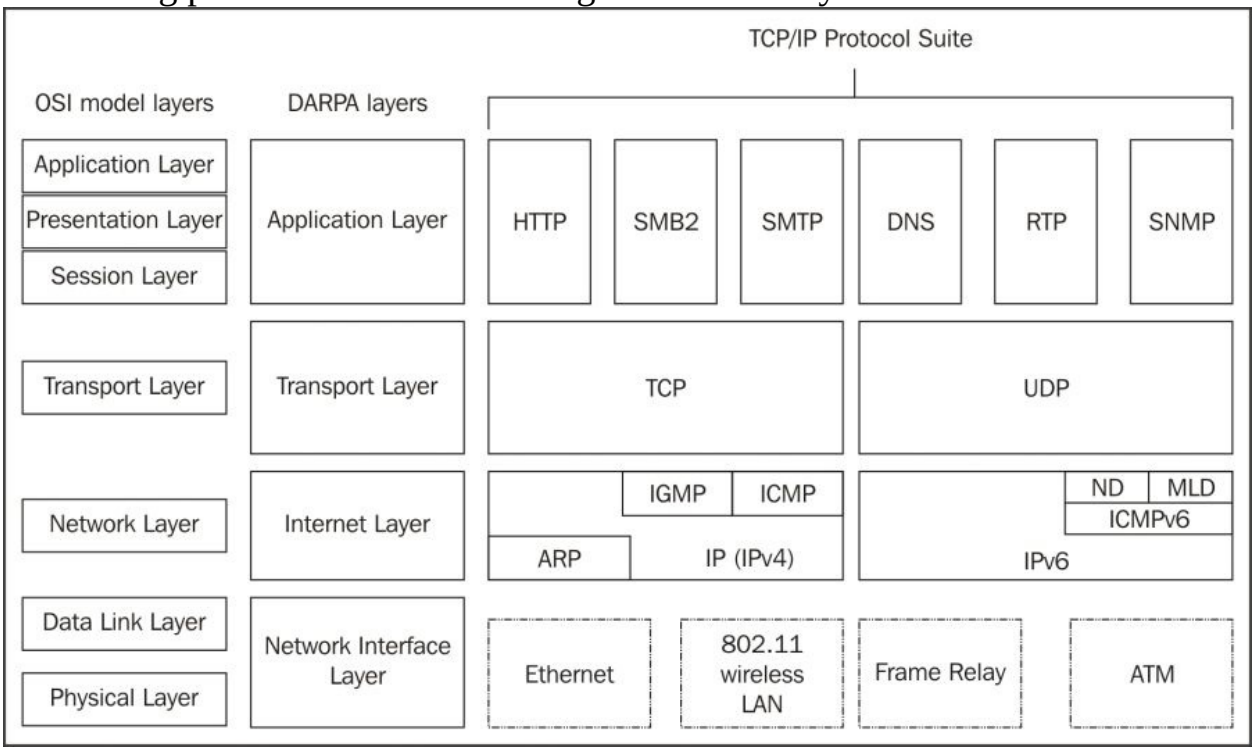
Layer 7 – the application layer

The application layer, which may (or may not) perform separate functions from the application itself, handles message formatting, human to machine interfaces, and so on. This layer represents the services that directly support applications such as software for file transfers, database access, e-mail, and so on.

In many widely used applications, no distinction is made between the presentation and application layers. For example, **HyperText Transfer Protocol** (**HTTP**), which is generally regarded as an application-layer protocol, has presentation-layer aspects such as the ability to identify character encoding for proper conversion, which is then done in the application layer.

In the DARPA model, the OSI layers 5-7 are combined into an application layer. From a packet analysis standpoint, the particular manifestations and visibility (in Wireshark) of the functions in the top layer(s) will vary depending on the applications and specific protocols employed to support them.

The following diagram summarizes the OSI and DARPA layers and how various networking protocols and services align with these layers and each other:



Encapsulation

You may have observed by now that packets encapsulate various protocols into successive layers, just like peeling an onion. An Ethernet frame contains a datagram payload; this datagram is a packet with an IP header and payload. The IP packet payload consists of a TCP header and data segment, which in turn may contain an HTTP header and payload. This encapsulation is easier to visualize when working within Wireshark's **Packet Details** pane.

IP networks and subnets

Before moving on, a short review of typical IP subnetting terms and typical applications should help clarify the terms used in this book and will act as a refresher for those already versed in IP addressing.

A `/24` designator placed after a network IP address in diagrams or device configurations is a **Classless Inter-Domain Routing** (**CIDR**) designator that indicates the following:

- The first 24 out of the 32 bits in the 4-byte IP address represents the network portion of any IP address on this network. This network is designated as `10.1.1.0` (the next `/24` network would be `10.1.2.0`, then `10.1.3.0`, and so on).
- The last 8 bits of the 32-bit address can be used to give workstations, hosts, and other devices an IP address, with the following exceptions:
    - The first host address on this network is reserved as a network designator to build routing tables: `10.1.1.0` (typically called the loopback address)
    - The last *host* address on this network is reserved as an IP broadcast address: `10.1.1.255`

The 8 bits binary is equal to 256 decimal, minus the preceding two exceptions. This leaves 254 usable IP addresses for devices, starting with `10.1.1.1`, `10.1.1.2`, and so on up to `10.1.1.254`.

- Another way of expressing subnet masks is in a dotted decimal format, `255.255.255.0`, which again indicates that the first 24 bits of an IP address is the network and the remaining 8 bits are for hosts.
- There are Class A, Class B, and Class C address ranges, as well as a subset of IP ranges reserved as private addresses to use within organizations.

The following table shows the IP address ranges in the three major classes:

Class of IP address	Starting IP address	Ending IP address
A	`1.0.0.0`	`126.255.255.255`

B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255

The following table shows the private IP address ranges:

Class of private IP addresses	Starting IP address	Ending IP address
A	10.0.0.0	10.255.255.255
B	172.16.0.0	172.32.255.255
C	192.168.0.0	192.168.255.255

- Subnet masks can be configured to allow more or fewer hosts per subnet with a corresponding tradeoff in having fewer or greater network addresses with which to build multiple networks within larger organizations.

A deeper review of IP addressing and subnetting is beyond the scope of this book. If you're not familiar with these concepts, some additional study would be advisable as a solid understanding of IP subnetting is essential for most analysis activities.

Switching and routing packets

So far, we've covered the topics required to discuss how packets of data get routed from computer A to host B across LANs and/or WANs over distances that may range from across a room to across the globe. The important concepts to remember are that Ethernet frames work with switches and IP packets work with routers to accomplish this feat, which we'll cover in the next section.

Ethernet frames and switches

To reiterate what was outlined in the layer 2 (the data-link layer) discussion, Ethernet frames are switched from the entry port to the appropriate destination port based on the destination MAC address. Network switches build tables of which MAC addresses belong to each port, compare a frame's destination MAC address to these tables, and switch the frame to the appropriate egress port if the destination is on the same switch or out a trunk port to another switch or router otherwise.

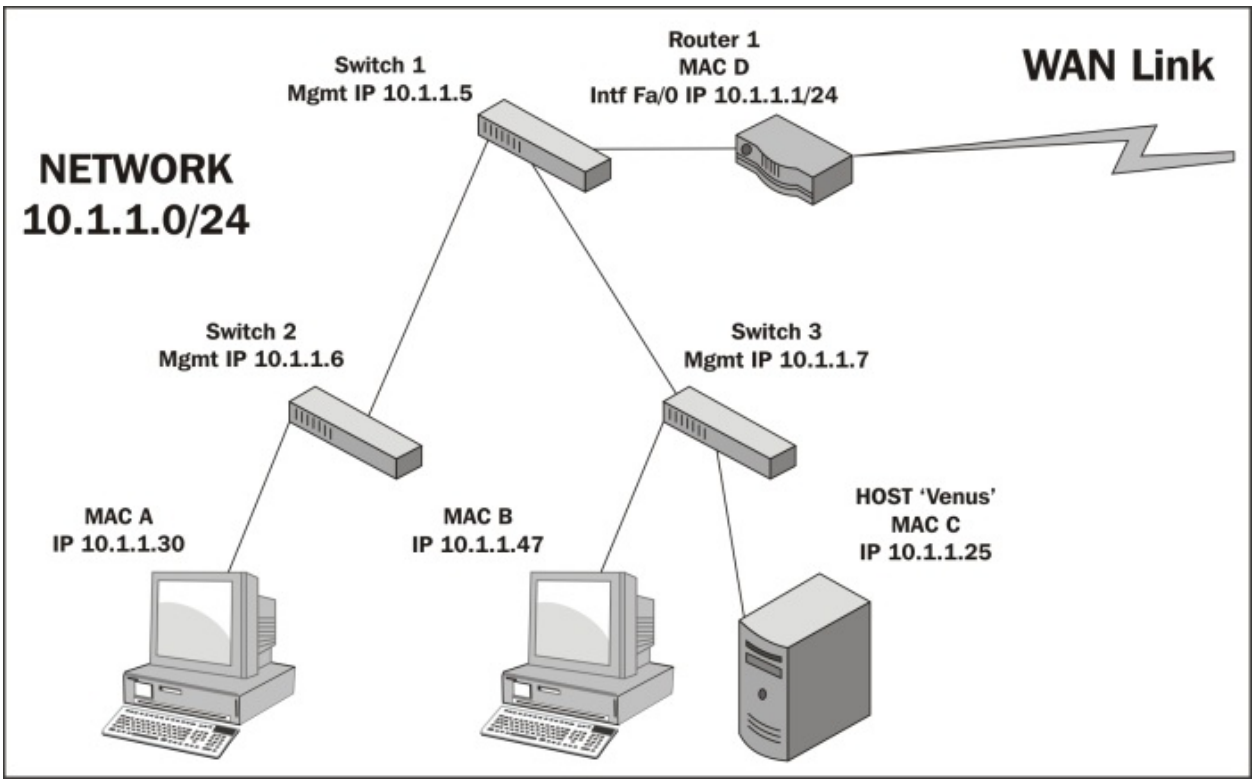
Note that the first time a switch sees a destination MAC address it doesn't recognize, it sends the packet (which will usually be an ARP packet) out all the ports until a device answers and it can add the new MAC address to its **content addressable memory** (**CAM**) table that maps MAC addresses to specific ports.

Frames carrying packets destined for remote networks are sent to the default gateway port MAC address. If you look at a list of MAC addresses in the **Ethernet** tab of a **Conversations** table in Wireshark and see an address with a drastically higher volume of traffic than the other stations, this is likely a default gateway (router) port MAC address. This port is the pathway into/out of this LAN from/to other networks.

On any given LAN, you'll see workstations, servers, and routers generating ARP and **Domain Name Service** (**DNS**) requests:

- **ARP**: This is used to resolve IP addresses to MAC addresses
- **DNS**: This is used to resolve hostnames to IP addresses

In the following diagram, there are two user workstations and a server that are connected together in a LAN residing on the `10.1.1.0/24` IP network. A router is attached to this network, which has a WAN link to another location.



The following two scenarios leverage this drawing to show how MAC addresses are utilized to switch Ethernet frames around local area networks:

- The workstation with MAC address B wants to use an application on the server Venus, which is unknown to all the network devices as it was just powered up. The workstation knows the IP address of Venus as the IP address was preconfigured in the client application, but it doesn't know the server's MAC address.

  The workstation broadcasts an ARP packet with its own MAC and IP address as the sender, the IP address of the Venus server, and all the zeros for the MAC address in the **Target** fields. Venus responds to the workstation with an ARP response that includes its MAC address of C in the sender MAC address.

  The workstation then sends a session initiation packet to the server using the server's MAC address as the destination MAC in the Ethernet frame.

These Ethernet frames traversed switch 3, which learned both devices' MAC addresses from observing the ARP conversations. The rest of the switches in the LAN network learned workstation C's MAC address when it broadcasted its ARP packet (because switch 3 sent this ARP packet out all ports), but not the server's MAC as the server responded directly to C.

- The workstation with MAC address A now wants to use an application on the server Venus. It doesn't know the server's MAC address either, so it sends an ARP request as well, which switch 2 broadcasts out all its ports, as does switch 1 and switch 3 as the switches only look at MAC addresses and the destination MAC address of any ARP request is **ff:ff:ff:ff:ff:ff**, so each switch is obliged to send the broadcast frame out all ports.

  However, when the server Venus responds to A's ARP packet, using A's MAC address, each switch in the path has learned which ports it saw A's MAC address come in on. So, each switch only sends Venus' response out the appropriate port back to workstation A. The same is true for learned non-broadcast frames. If a switch doesn't recognize a destination MAC address of a nonbroadcast frame, these are sent out all ports the first time as well.

  As switch CAM tables get populated with MAC addresses and their associated ports, the number of frames that must be sent to every device in the LAN as well as the workload on all these devices is reduced significantly.

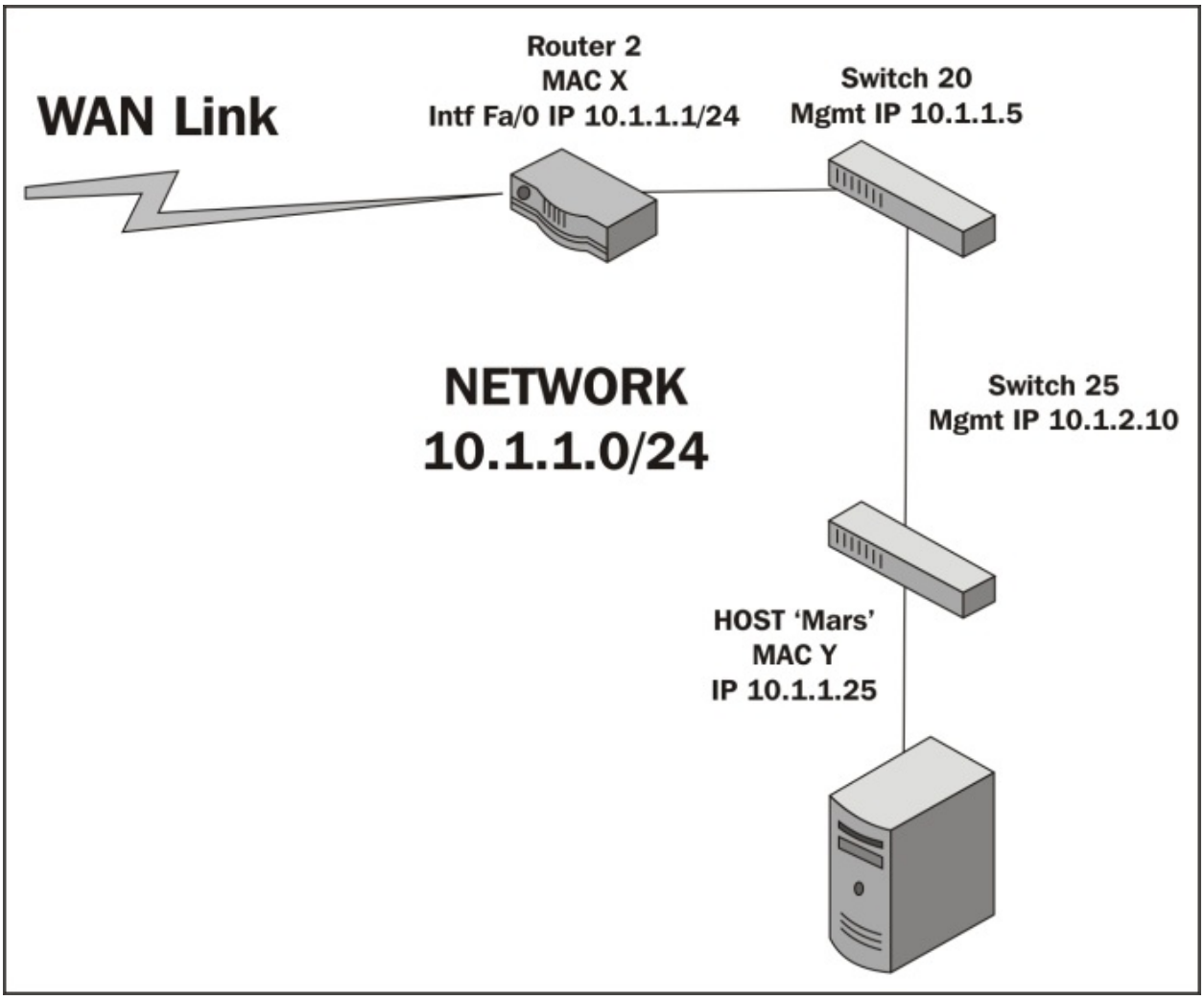
IP addresses and routers

When packets need to leave the LAN to get to a remote IP network, routers are required to route the packets based on their destination IP addresses. The following scenario (still referring to the preceding screenshot) illustrates some of the details involved in one possible situation.

Workstation A now wants to use an application on the server Mars, which resides on a different network than in the previous scenarios. And in this case, workstation A doesn't know the IP address of the server; it only needs its name. Workstation A will send a DNS request packet to the DNS server IP address configured in its network settings (the DNS server isn't shown here) with the hostname Mars; the DNS server will return the IP address of Mars 10.1.2.25. Workstation A calculates that this host isn't on its own network from a comparison of its IP address and subnet mask with Mars' IP address, so it sends the session initiation packet to router 1, which was configured as its default gateway in its network settings. We'll assume that Workstation A already knows the MAC address of router 1's port from a previous ARP exchange to find router 1's MAC address from the given IP address.

When the router receives A's frame, which was sent to the router port's MAC address, it inspects the destination IP address inside the IP header and looks up the appropriate port to forward the packet to. This routing process is supported by routing table entries the router builds from route information broadcasted by other routers; each router tells all the others what networks it knows a route to.

In this case, the Ethernet frame surrounding A's packet is stripped off and the remaining payload (packet) is sent across the WAN link to router 2, which also inspects the IP header destination IP address and looks up the correct port to forward the packet to. Router 2 wraps the packet in a new Ethernet frame with its own MAC address X as the source and the Mars server's Y address as the destination MAC (assuming the router already has the server in its MAC table), and transmits the packet out onto the LAN to get switched to the Mars server, as shown in the following diagram:


WAN Link
Router 2
MAC X
Intf Fa/0 IP 10.1.1.1/24
Switch 20
Mgmt IP 10.1.1.5
NETWORK
10.1.1.0/24
Switch 25
Mgmt IP 10.1.2.10
HOST 'Mars'
MAC Y
IP 10.1.1.25

WAN links

Actually, network packets may traverse several routers and WAN links to reach the destination network, and each router traversed is called a hop. In the context of packet analysis, you should be aware that WAN links can introduce packet delivery delays or latency due to the following four major factors:

- **Physical speed-of-light propagation delay**: This is the amount of time required for electrical or light signals to travel across copper/fiber cables over long distances.
- **Network routing/geographical distance**: The WAN link routes are never in a straight line between points. They have to traverse major telephony switching centers and route along railways, roads, and other opportunistic paths.
- **Serialization delay into and across WAN links**: The WAN links are often slower speed links, and it takes a finite amount of time to send packet data across these links one bit at a time.
- **Queuing delays**: In network device buffers, including additional delays that may be induced by Quality of Service policies, some packets receive priority and others have to wait longer for their turn to be transmitted.

The effects of network delay incurred across LAN and WAN links can be seen and measured in Wireshark packet traces by inspecting the elapsed times between session setup packets.

Wireless networking

Wireless networks utilize a range of 802.11 specifications to provide connectivity over 2.4 or 5 GHz frequency bands at a variety of speeds. The significant differences between wireless frames and those found on wired networks are as follows:

- Wireless networks employ carrier sense (every station is listening), multiple access (shared medium), and collision avoidance (avoiding collisions instead of just recovering from them) techniques, which reduce the throughput
- In addition to data frames, which get forwarded to the wired network, wireless frame types include the following:
    - **Management frames**: This is used for authentication and association tasks
    - **Control frames**: This controls send/receive functions on the shared media to help avoid collisions

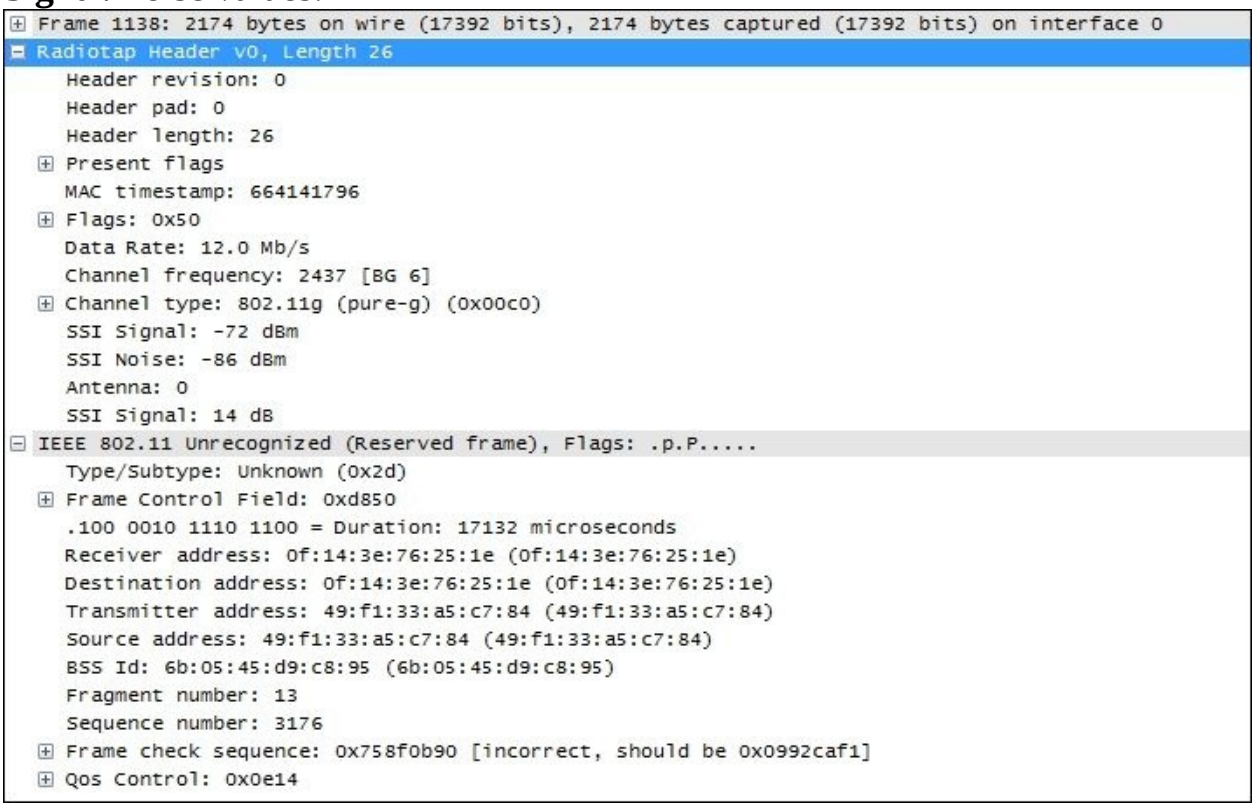
Wireshark can be used to capture and analyze packets on Wireless networks. However, in order to analyze the control and management frames, as well as select the radio channels to capture on without having to associate with a specific channel, specialized adapters are required. These adapters are available from various networking vendors.

These wireless adapters and their drivers enable Wireshark to display a pseudo header just below the frame header in the **Packet Details** pane, which includes information about:

- **Data rate**: This is the maximum data transfer rate possible across the radio channel
- **Channel frequency**: This is the RF channel frequency that the station is using
- **Channel type**: This is the `802.11` protocol used, and the common types are *a*, *b*, *g*, and *n*
- **RF signal and noise levels**: This is the received RF signal strength and background noise levels; the larger the difference between these two the better the signal can be decoded

Remember when analyzing wireless networks, the wireless access points utilize a wired LAN connection to the rest of the network that may warrant a separate analysis. The access point strips off the 802.11 header and encapsulates a packet in an Ethernet frame before sending the packet off on the wired network.

The following screenshot illustrates the contents of a typical **Radiotap Header** and **IEEE 802.11 frame**; note the **Data Rate**, **Channel frequency**, and **Signal/Noise** values:



Note

There are numerous reference materials and books that you can read to learn more about networking and network protocols. One of the classic sources is *TCP/IP Illustrated Volumes I, II, and III, W. Richard Stevens, Addison-Wesley Professional*, available online or in book formats.

Summary

The important points covered in this chapter included how Ethernet frames are switched to the appropriate switch ports on a LAN based on destination MAC addresses that packets are routed across and to remote networks based on destination IP addresses, and how the frames carrying packets destined for remote networks based on the destination IP address are sent to the default gateway's port MAC address.

We also covered how and why slower and/or longer distance WAN links can add significant amounts of delay to packet transmissions, which slows application data exchanges and increases user response times. We finished the chapter by discussing how Wireshark can capture and analyze packets on 802.11 wireless networks using specialized adapters.

In the next chapter, we'll cover in detail how to capture and filter packets using Wireshark.

Chapter 3. Capturing All the Right Packets

In order to analyze packets to troubleshoot connectivity, performance, or security issues, you have to successfully capture all of the right packets and then identify and filter out just the packets that pertain to the goal at hand.

In this chapter, we will cover the following topics:

- Picking the best capture point
- TAPs and switch port mirroring
- Wireshark's capture interfaces, filters, and options
- Verifying a good capture
- Isolating the conversation(s) of interest
- Using the Wireshark Conversations window
- Wireshark's display filters
- Filtering expression buttons
- Following TCP/UDP/SSL streams
- Marking and ignoring packets
- Saving filtered traffic

You'll recognize that many of these activities are the same ones that we accomplished in Chapter 1, *Getting Acquainted with Wireshark*, to perform a capture and filter just the packets involved in loading a web page. In this chapter, we'll expand and finish rounding out your skills in all these topics.

Picking the best capture point

Determining the best location to perform a packet capture depends on several considerations:

- The nature of the issue being investigated
- The relative ability to perform a capture in a location that provides the highest degree of usefulness to the analysis
- The amount of technical difficulty, risk, and time required to perform a capture at a given location

User location

If you're troubleshooting a user complaint, the first capture point should be at the user's workstation to gain a view from the user's perspective and verify/clarify the situation that the user is reporting. From this vantage point, you can:

- Ensure that basic network services such as ARP and DNS are working correctly
- Analyze the initial login process if the user authentication involves a different device than the target application server
- Measure network round trip times from the user to the target host(s)
- Determine whether the TCP session setup handshake is appropriate for the application being accessed
- Measure service response times (such as HTTP or SMB response times)
- Determine whether the user is experiencing packet loss and retransmissions, out-of-order packets, or other network-related anomalies
- Capture any application error messages being sent to the user and the requests that resulted in those errors

Capturing from a user's location is usually much simpler from a practical standpoint and there is a lot less traffic to deal with, which makes capture sizes smaller and filtering the packets of interest simpler. Disconnecting a user's Ethernet cable for a few minutes to insert a TAP (we'll discuss these in the next section) or installing Wireshark on the user's workstation does not typically require special authorization or preparation as the risk to other users is negligible.

Server location

If a capture from a complaining user's workstation isn't possible or practical, a capture from the server end can still be useful, but it might be advantageous to apply a capture filter to gather just the traffic to/from the user's workstation (based on the user's IP address) to limit the capture file size. You can still measure network round trip times, server response times, analyze TCP handshake details, and detect retransmissions caused by packet loss, and perhaps the login/authentication process from this location.

Capturing from a server location is also appropriate when analyzing backend service response times. For example, if users interact with an application server but that app server performs transactions with a backend database in order to fulfill user requests and if there are complaints of slow response times, then an analysis of application server-to-database server interactions can help isolate the true source of the poor performance to one or the other host and the types of requests that result in slow or erroneous responses.

Other capture locations

For the majority of packet captures, you'll likely be at user workstations or server switch ports, but there will also be some cases where captures will need to be performed at other locations.

Mid-network captures

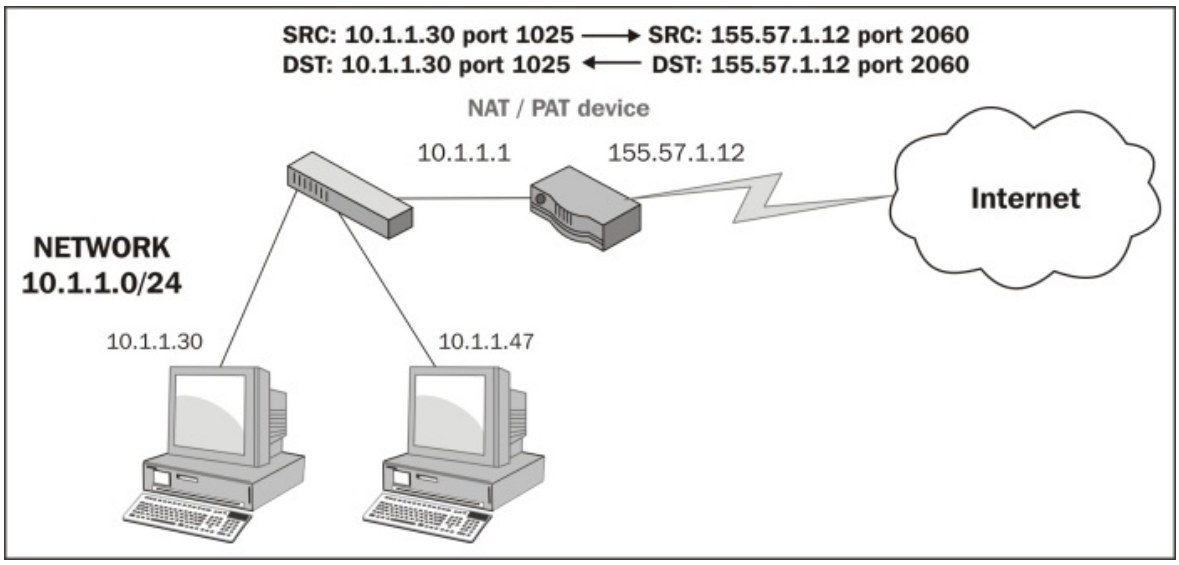
Identifying the source of excessive packet loss or disordering over a network path may require performing packet captures at various points along that path, typically at distribution or core switch trunks, or interfaces to routers, firewalls, and so on, to find the network segment where packet loss becomes apparent.

Both sides of specialized network devices

Today's modern networks often employ a number of network devices that can actually alter the contents of packets flowing between clients and servers; in some (occasional or last resort) cases, it may be necessary to capture on both sides of these devices to isolate or prove a functional or configuration problem:

- **Routers and gateways**: These are also called Internet gateways in some configurations and may be configured to perform a **Network Address Translation** (**NAT**) function that alters and hides the user's actual IP address from an outside network. This is done by substituting a public IP address for the user's real address. This usually involves translating port numbers as well so that a single public IP address can be used to support multiple sessions; in which case, the solution is called **Port Address Translation** (**PAT**). The end result of the PAT functionality is that a capture from the client side and a capture at the server side of the same session conversation will involve different IP addresses and port numbers.

  The following diagram illustrates how a PAT device translates IP addresses and ports from an internal private network to and from an externally visible IP address and has translated the ports used for an individual user session:



- **Proxy servers and firewalls**: Devices such as these can act as an intermediary between clients wanting to use resources from other (usually external) servers. These devices are most typically deployed between users inside a company and outside (web) services accessed via the Internet. These devices are employed for their security capabilities, allowing administrative control over what can be accessed and the type of data content that can be relayed between the two networks, malware scanning, and so on. From a packet analysis standpoint, you should be aware that in addition to performing a NAT/PAT function, some implementations of these devices may actually terminate a user session on one side and initiate a completely different session between the device and the outside host on the other side, on behalf of the user, such that the TCP handshake and session parameters, IP addresses and port numbers, and packet sizes can all differ on either side.
- **IP tunnels using Generic Routing Encapsulation**: These are used to connect two IP networks that don't otherwise have a native routing path to each other. The original packets are encapsulated inside packets with different IP addresses appropriate for the network media that they will traverse. The most common use of IP tunneling is to connect private corporate networks together through public Internet connections or to connect **Internet Protocol Version 6** (**IPv6**) networks together over traditional IPv4 network paths. IP tunnels can be configured between routers and high-end switches.

Although it may be necessary (to validate an issue to other support teams) or

more practical to capture at or near the interfaces to the devices described earlier, it is usually easier and just as effective to perform the captures at user and/or server locations. Unless you're part of a network support team, you won't have to conduct an analysis in such an advanced and complicated environment.

Test Access Ports and switch port mirroring

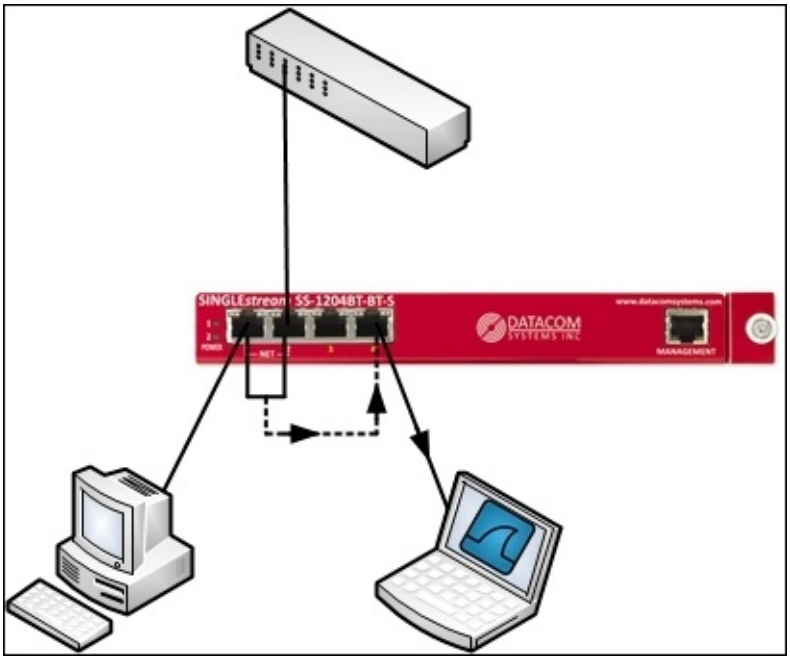
If you're capturing from a user location and cannot or do not wish to install Wireshark on the user's machine or you're capturing at another location in the network, you have two options to obtain a copy of the packets traversing the network: Test Access Ports or switch port mirroring.

Test Access Port

A **Test Access Port** (**TAP**) is a device that copies all the packets flowing through it to one or more monitor ports. A station with Wireshark installed on it can be connected to one of the monitor ports to capture the packets.

You should select an aggregating TAP that supports the link speed of the network ports being analyzed (usually 100 Mbps or 1 Gbps) and that will copy and combine the packets flowing in both directions (transmit data from the user's workstation and receive data from the network); the aggregating TAP funnels the traffic to a single connection (transmit to the Wireshark station) so that you can capture the traffic in both directions with a single network interface on the Wireshark station. Be aware that since you're copying packets from two directions into one pipe to the Wireshark station, it is possible to oversubscribe the monitor port if traffic rates are extremely high. If this happens, the excess packets will be dropped. Oversubscription usually isn't a concern at user workstations, but it could be for switch trunks or other high traffic areas.

The following figure illustrates how a TAP is inserted between a user workstation and that workstation's switch port, and how a Wireshark workstation is attached to capture packets:



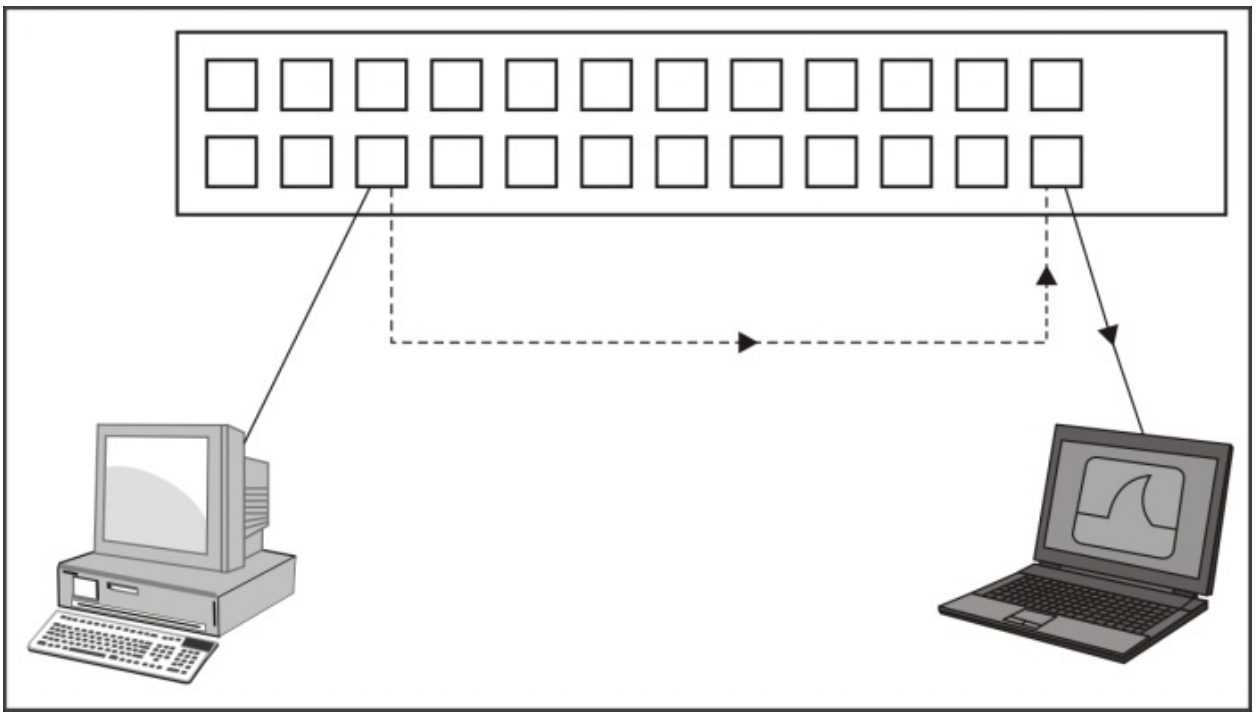
Switch port mirroring

Switch port mirroring, also known as a **Switched Port Analyzer** (**SPAN**) feature or spanning a port, is the practice of configuring a network switch to perform the same function as a TAP: to make a copy of the packets flowing in and out of a specified port and send them to an otherwise unused monitor port where a Wireshark station is attached to capture the packets.

The advantage of using port mirroring is that no connections need to be broken to insert a TAP. The monitor port can be easily configured by a switch administrator and just as easily disabled.

The potential issues with this option include the fact that not all switches support port mirroring, and there is some evidence to suggest that using this feature can affect the performance of the switch, at least for the port being monitored. The possibility of oversubscribing the monitor port from excessive transmit plus receiving traffic levels also exists for port mirroring, as is the case when using a TAP, and this is likely when monitoring switch trunks to other switches, as these will be carrying traffic for multiple users.

The following diagram is a simple illustration of a port mirroring scenario on a switch. The packets to and from the workstation port are copied to the port where the Wireshark station is connected.



Capturing packets on high traffic rate links

If you need to capture packets on a high traffic rate link such as a trunk link between larger switches, Wireshark is probably not the best solution. It may not be able to keep up with a busy link. Wireshark is actually a GUI tool that calls a command-line executable called **dumpcap**, which captures the packets and saves them to a disk file. Wireshark reads this file and presents the processed packets to the user interface. An alternative to Wireshark is to use the dumpcap or **tcpdump** executable directly (these are covered in Chapter 8, *Command-line and Other Utilities*) or a high performance capture appliance offered by numerous vendors.

Capturing interfaces, filters, and options

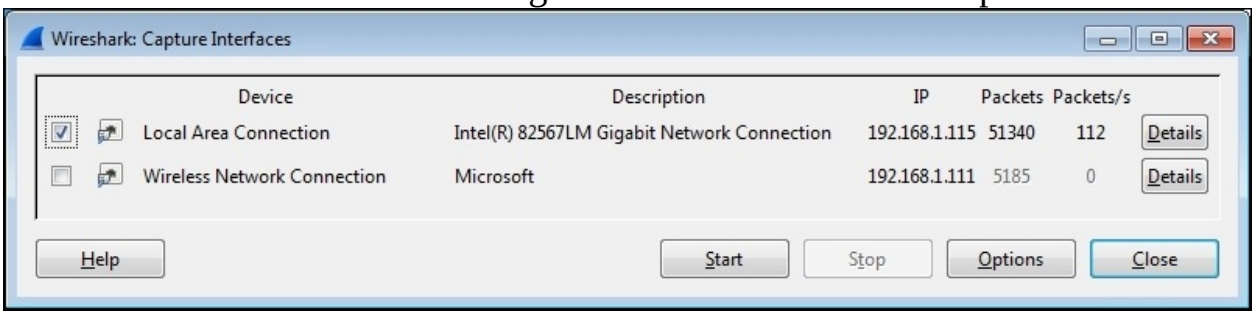
Capturing packets with Wireshark consists of selecting the correct network interface to capture packets from, applying any capture filters that may be appropriate, and applying the correct options to accomplish the capture in the desired manner. We'll cover these three topics in the following sections.

Selecting the correct network interface

As discussed in Chapter 1, *Getting Acquainted with Wireshark*, if you have multiple network interfaces on your machine, you need to determine and select the correct interface to capture packets. In Wireshark's **Capture** menu, click on **Interface** or click on the first icon on the icon bar.

The **Wireshark Capture Interfaces** window provides a list and description of the network interfaces on your machine, the IP addresses assigned, and the total packets and packets per second counters for each interface. If an interface has an IPv6 address assigned and this is being displayed, you can click on the address to toggle and display the IPv4 address.

The following screenshot illustrates a typical **Capture Interfaces** window listing a LAN and wireless interface along with their IP addresses and packet counters:

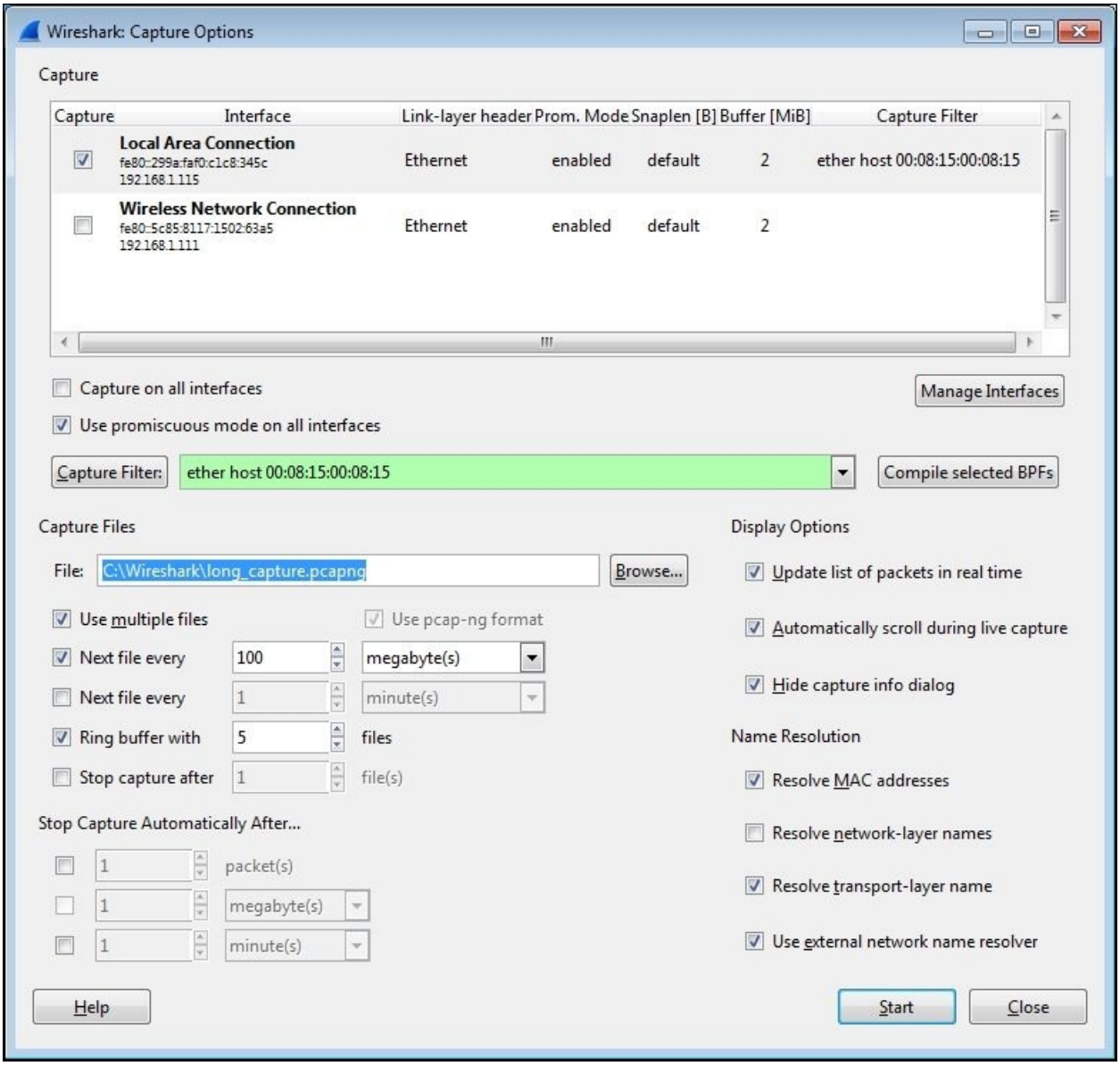


The **Capture Interfaces** window provides the following two options:

- Clicking on the **Details** button for any of the listed interfaces opens an **Interface Details** window that provides a wide range of information that can be useful to verify the interface's operation. The status of the **Link** and **Link Speed** information is displayed in the **Characteristics** tab, and the MAC address of the selected NIC is displayed in the **802.3 (Ethernet)** tab.
- The rest of the capture options are configured in the **Capture Options** window, which is opened by clicking on the **Options** button in the **Capture Interfaces** window, or by selecting **Options** from the **Capture** menu, or by clicking on the second icon in the icon bar.

The following screenshot illustrates a typical **Capture Options** window with a

number of options specified. You can refer to it for examples of the topics on **Capture Options**.



As seen in the preceding screenshot, the **Capture Options** window displays the available interfaces and their IP addresses and allows you to select one or more of these interfaces to perform the capture. Wireshark can capture from multiple interfaces simultaneously, as well as from virtual interfaces. The primary advantage of starting with the **Capture Interfaces** window is the availability of the packet counters to aid in identifying active interfaces, a feature not available

in the **Capture Options** window. Otherwise, if you know which interface you'll want to use, you can skip using the **Capture Interfaces** window and start here.

Clicking on the **Manage Interfaces** button in the **Capture Options** window brings up an **Interface Management** window. From the **Local Interfaces** tab, you can select to hide interfaces that you do not wish to see displayed in the **Capture Interfaces** and **Capture Options** windows.

There is an option to quickly enable **Capture on all interfaces** and a **Use promiscuous mode on all interfaces** option that is enabled by default. In most cases, this option should be left enabled so that the chosen interface(s) can capture and save all the packets seen. Otherwise, only the packets that are being sent to the Wireshark workstation's MAC address, broadcast, and/or multicast packets will be seen and captured, which basically negates its usefulness as a capture device. The **Compile selected BPFs** button provides a machine language display of the compiled capture filter, but has no other functional purpose.

Note

The **Capture Filter** field has a highlighting feature that indicates valid versus invalid filter syntax. A green background indicates a good filter and a red background indicates an invalid or incomplete filter.

Using capture filters

Capture filters are used to reduce the amount of traffic saved during a packet capture. In practice, capture filters should be used sparingly, if used at all, to help make sure that no packets that are important for an analysis are inadvertently missed because they fall outside the capture filter parameters. Remember that you can always filter out unwanted traffic from a capture, but you can't do anything about missed packets once the capture is finished. If you're unsure about a capture, perform the capture again with a more generous capture filter or none at all.

One scenario where a capture filter is appropriate is when you want to let a capture run for a long period of time. Then, you should filter out as much extraneous traffic as possible to keep capture file sizes under control. However, take care to make sure the capture filter you apply doesn't exclude any traffic that may be useful for the analysis.

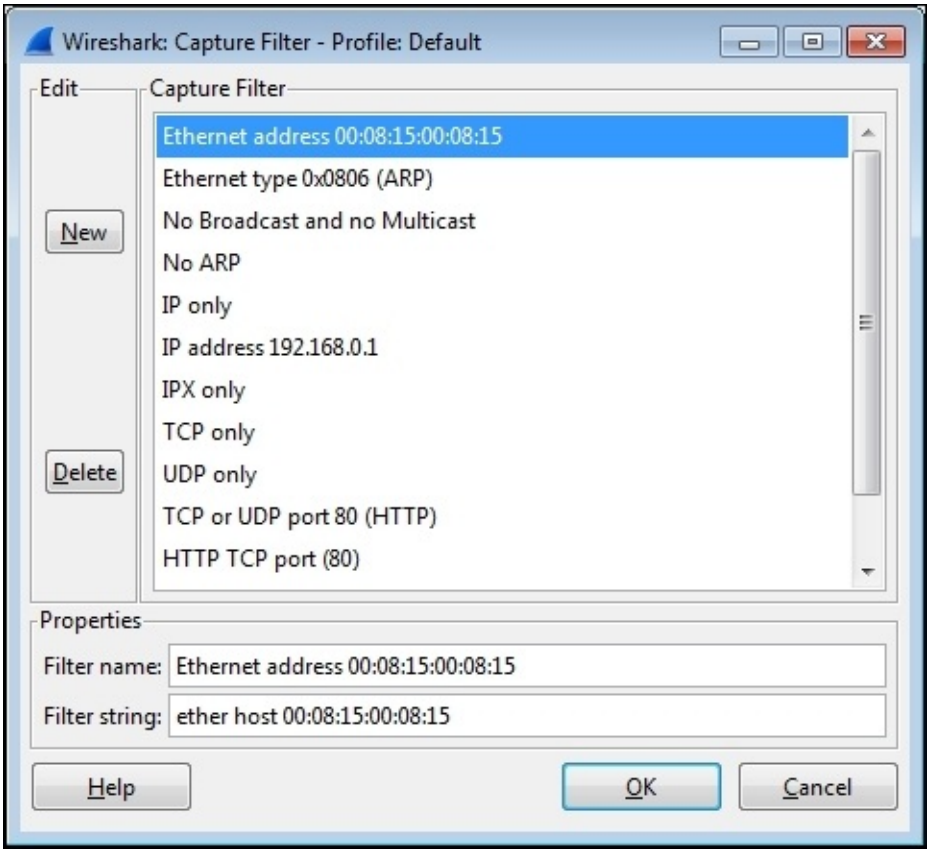
It's usually a good idea to do some trial captures when using capture filters to verify that the filter is working as desired before doing the official capture that you want to keep.

Configuring capture filters

Wireshark provides a **Capture Filter** window that makes it easy to select a preconfigured capture filter, or you can configure your own based on your needs.

Click on the **Capture Filter** button in the **Capture Options** window to open the **Capture Filters** window. From this window, you can select from a number of useful preconfigured capture filters, create a new and unique capture filter for your purposes, or delete unwanted or erroneous filters. Creating a new filter only involves giving the filter a name, entering the capture filter syntax, clicking on **New** to save the filter, and then finally clicking on **OK**. Alternatively, you can click on an existing filter and then click on **New**, which will create a copy of that filter at the bottom of the list that can then be modified for your purposes.

The following screenshot illustrates a typical **Capture Filter** window. In this case, a capture filter that will only allow traffic to and from a specific Ethernet MAC address has been selected:



Wireshark's capture filters use a syntax that is known as the **Berkley Packet Filter** (**BPF**) format, which has legacy roots in the Unix world and is still in use today with packet-level drivers. Note that the syntax used to capture filters in Wireshark differs significantly from the syntax used for display filters.

The default selection of capture filters from the **Capture Filter** window is helpful in providing examples of capture filter syntax. Some additional examples of capture filter syntax and examples of that syntax are outlined in the following table:

Description	**Syntax**	**Examples**
Filter on an Ethernet MAC address Filter to capture just the traffic from or to a MAC address	ether host `xx:xx:xx:xx:xx:xx` ether `src` or ether `dst`	ether host `00:1c:25:99:db:85` ether `src` `00:1c:25:99:db:85`

Filter on an IP address or hostname Filter to capture just the traffic between two IP addresses Filter traffic in one direction only between two hosts	host `xxx.xxx.xxx.xxx`
	
	`src` host and `dst` host
		`host` `192.168.1.115`
		
		`host` `www.wireshark.org`
		
		`host` `192.168.1.115` and `host 10.1.1.125`
		
		`src host` `192.168.1.115` and `dst host` `10.1.1.125`

Filter based on a port number
 	
Filter for DNS packets
 	
Filter for DHCP packets
	`port`, `dst port`, and `src port`	`port 53`
		
		`port 67`

Filter based on a protocol Filter for HTTP traffic only	`arp`, `icmp`, `ip`, `upd`, `tcp`, `http`, `ip6`, and `icmp6`	`http`
Capture filter logical operators	`=`, `!=`, `>`, `<`, `>=`, `<=`, `!`,	`not arp` and `port not 53`

Filter to exclude ARP and DNS packets	`not`, `&&`, `and`, `||`, `or`	`! arp && port ! 53`

More information and examples of capture filters can be found on the Wireshark wiki at http://wiki.wireshark.org/CaptureFilters and the protocol-specific capture filter syntax is included in the reference information found at http://wiki.wireshark.org/ProtocolReference.

Capture options

The Wireshark **Capture Options** window offers a variety of controls to configure captures to suit a particular need.

Capturing filenames and locations

Clicking on the **Browse** button on the **File** option allows you to navigate to a chosen directory in which you can store the capture files and enter a filename for the capture files.

When the **File** option is used, Wireshark will append a file number and date-time stamp to the filename you specify and will not provide a file extension. You should specify a `.pcapng` extension in the filename. This is better illustrated with an example.

The user provided directory and filename is `C:\Wireshark\long_capture.pcapng`, and Wireshark will create and save packets to files in the format `C:\Wireshark\long_capture_00001_20140724132952.pcapng`.

If Wireshark is configured to capture to more than one file (this will be discussed later), the file numbers and date-time stamps will be incremented accordingly as the capture progresses, for example, `long_capture_00002_20140724133343.pcapng` and `long_capture_00003_20140724133612.pcapng`.

Multiple file options

Wireshark can be configured to save packets to multiple files to allow long-term captures, and offers a number of options to control how this is accomplished.

Selecting the **Use multiple files** option causes the appropriate underlying controls to become active as specific options are enabled. You can choose to start a new (next) file when each file reaches a given size and/or after a configurable period.

Note

Wireshark can become very sluggish or might even crash when working with capture file sizes of much greater than 100 MB, so you should consider this as a good maximum file size.

Ring buffer

The **Ring buffer** option works in conjunction with the **Next File every** option to cause Wireshark to fill the specified number of files, and as the capture continues to progress, it deletes the oldest files.

This feature is useful to keep a capture running while waiting for some intermittent problem or an event to occur, after which the capture is manually stopped. The ring buffer files provide historical capture data for a period just prior to stopping the capture, without filling a hard drive with an excessive number of large capture files.

Stop capture options

Wireshark has options to automatically stop a capture after a specified number of packets, file size, or time period. If the **Use multiple files** option is enabled, an option to stop the capture after a specified number of files can be employed. Otherwise, the capture can be stopped after a specified number of packets, file size, or time period has elapsed.

Display options

The **Update list of packets in real time** option specifies that Wireshark is to periodically read the capture file as it is being written during the capture and update the **Packet List** contents accordingly. Otherwise, the Wireshark user interface will be grayed out during the capture.

The **Automatically scroll during live capture** option specifies that Wireshark updates and displays the latest captured packets in the **Packet List** pane such that the packets seem to scroll up as the list is updated. The **Update list of packets in real time** option must be enabled for this option to function.

Both of these options have a processing time cost that could result in lost packets and/or a sluggish display and should be disabled if capturing on a very busy link. However, the ability to view and confirm that the expected packet flows are

occurring during the capture will be lost.

The **Hide capture info dialog** option (which is enabled by default) controls whether a simple window is displayed during the capture that displays the packet counts and percentages by protocol. Unless specifically needed, it is best to leave this window hidden.

Name resolution options

If the **Resolve MAC addresses** option is enabled, it causes Wireshark to display MAC addresses with an assigned manufacturer code in place of the first three octets. For example, Wireshark will display **CiscoCon_21:b7:ec** instead of **c8:d7:19:21:b7:ec**. The list of manufacturer's codes is kept in the `manuf` file of the Wireshark installation directory.

The **Resolve network-layer names** option works in conjunction with **Use external network name resolver** to determine if or how captured IP addresses are resolved into their hostnames, as follows:

- The **Resolve network-layer names** option specifies that Wireshark should attempt to resolve IP addresses into hostnames. If the **Use external network name resolver** option is enabled, Wireshark will perform reverse DNS lookups for each unique IP address. This causes Wireshark to generate traffic of its own.
- If the **Use external network name resolver** option is disabled, Wireshark will attempt to resolve the IP addresses using a `hosts` text file provided by a user (which uses typical IP address `<tab>` hostname syntax) located in the Wireshark installation directory when using a default profile or in the appropriate profile directory when using a custom profile.

During a capture, it is better to leave the **Resolve network-layer names** option disabled so that Wireshark isn't creating additional network traffic while trying to resolve IP addresses during a capture. This feature can always be temporarily enabled (by navigating to **View** | **Name Resolution** | **Enable for network layer** from the menu) after the capture is finished.

If the **Resolve transport-layer name** option is enabled, it causes Wireshark to display the human-readable, port- and protocol-specific services' names instead of the port numbers in the Info display field in the **Packet List** pane. For

example, TCP port 80 will be displayed as HTTP. The list of port number services is kept in the `services` file in the Wireshark installation directory.

The screenshot at the beginning of this section illustrates a **Capture Option** window set to use the LAN interface, a filter to capture traffic only to and from a specific Ethernet MAC address, to save up to five files of 100 MB each in a ring buffer scenario, and to save those files in a provided directory with a specific leading filename and extension. The **Display Options** and **Name Resolution** options have been left in their default settings.

Once all the desired **Capture Options** have been selected, clicking on the **Start** button will start the capture.

Having covered all the most useful **Capture Options** features, now is probably the right time to tell you that for many of your captures, especially from a relatively low traffic volume location such as from a user workstation, you don't want or need to set any capture options (except the appropriate interface to capture from) and can simply jump into starting a capture using all the defaults by clicking on the third (green shark-fin shaped) icon on the icon bar or selecting **Start** from the **Capture** menu. Not using a capture filter allows you to capture all the relevant packets—without missing anything—and filter any unwanted traffic out using display filters after the capture is done.

Verifying a good capture

After a capture is complete, you should scroll through and inspect the packets in the **Packet List** pane to ensure that you're seeing the traffic you were expecting —usually traffic to and from a specific host.

You should also ensure there were no dropped packets, which would be displayed in the **Packet Information** section of the **Status Bar** at the bottom center of the Wireshark user interface. Dropped packets indicate that Wireshark or the selected NIC could not keep up with the traffic volume and had to discard packets, which could of course affect the quality of your analysis. If dropped packets occur, you may need to use a higher performance workstation to perform the captures or select a lower traffic volume capture location.

Saving the bulk capture file

After completing and verifying a good capture, you should save the bulk (all captured packets) capture file (assuming a single file capture) to your directory of choice. You will later be filtering and saving a subset of packets to a smaller file, but it is advantageous to be able to load the original capture file again at a later time if during the analysis you discover that you might have inadvertently filtered out more packets than you wanted.

Using the **Save As** option in the **File** menu, navigate to the directory of your choice and give the file a name. If no file extension is specified, Wireshark will append a file extension based on the **Save as type** option selected; the default is the `.pcapng` format. However, you can save the file in several other popular vendor-specific formats if you intend to share the capture file with someone who is using a different protocol analysis tool.

If multiple files were saved using one of the multiple file and/or ring buffer capture options, navigate to the **File** | **File Set** | **List Files** to select and open one of the files.

Isolating conversations of interest

After you have completed a packet capture and saved a bulk capture file, you'll be with an almost overwhelming number of packets of various types and addresses in the **Packet List** pane. It's now time to par this down to just the packets that pertain to the analysis task at hand.

The idea is to progressively eliminate unrelated packets; analyze the pertinent conversations looking for anomalies; and again progressively filter, measure, and analyze packet flow and application behavior until you have discovered and can document the root cause of the issue.

There are two basic ways to isolate and inspect packets and conversations of interest, and you'll likely use both of the following methods in most of your analysis activities:

- **Conversations**: This window creates a list of conversation pairs by MAC or IP address and/or TCP/UDP ports that can be sorted. It displays filters that will isolate and display only the selected conversation packets can be quickly applied from this window.
- **Display Filters**: These filters are based on MAC or IP addresses and/or protocol-specific fields that limit the packets displayed in the **Packet List** pane.

We'll discuss each of these methods in the following sections.

Using the Conversations window

The basics of using the **Conversations** window were covered during the first capture in Chapter 1, *Getting Acquainted with Wireshark*. In this section, we'll cover a few other handy features of the **Conversations** window.

The Ethernet tab

The **Conversations** window exhibits specific behaviors in the **Ethernet** tab, depending on the available **Name Resolution** settings. If **Enable for Network Layer** in the **Name Resolution** menu, which can be found in the **View** menu, is enabled and **Name Resolution** is also enabled in the **Conversations** window, then the IP address that is associated with a given device's MAC address is displayed as an IP address instead of a MAC address. Toggling the **Name Resolution** option in this scenario is useful for easily associating a devices' IP address with its MAC address.

If the **Enable for Network Layer** option is not enabled, then the **Name Resolution** option in the **Conversations** window controls whether the MAC addresses are displayed with manufacturer prefixes or as the basic 6-octet MAC address.

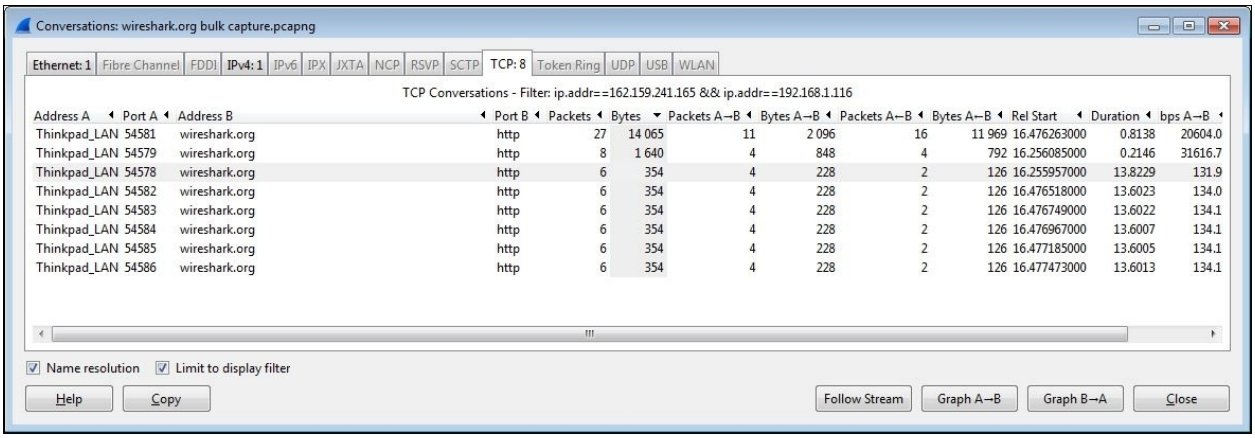
The TCP and UDP tabs

The **TCP** and **UDP** tabs of the **Conversations** window list all of the conversations between devices based on IP addresses and ports. Considering that network communications between a pair of devices, each with their associated IP addresses, could include multiple sequential or simultaneous sessions with differing port numbers, the **TCP** and **UDP** tabs (depending on the protocol in use) make it much easier to inspect the number and relative size and start/duration of these individual sessions.

As can be done in any of the other tabs in the **Conversations** window, a display filter can be quickly prepared or applied using the right-click functionality.

A helpful practice when investigating TCP or UDP sessions is to apply a display filter on just the IP addresses initially and then enabling the **Limit to display filter** option at the bottom of the **Conversations** window. Upon returning to the **TCP** or **UDP** tab, only the port-level sessions between the filtered host pair are displayed, which makes investigating these sessions much easier than picking them out from the entire list.

The following screenshot shows the multiple TCP sessions that were involved in loading the https://www.wireshark.org/ home page after applying a display filter (from the bulk capture file) and enabling the **Limit to display filter** option in the **Conversations** window. It can be seen that the (top) conversation between port **54581** on the user workstation and port 80 (HTTP) carried the vast majority of the traffic; the remaining ports carried much smaller amounts of traffic.



The WLAN tab

Since the **Conversations** window tabs are ordered alphabetically, the **WLAN** tab comes at the end. This tab displays the wireless station MAC addresses, as well as the **Bytes**, **Packets**, and other columns offered in the other tabs.

Wireshark display filters

Wireshark provides a very wide range of protocol-specific display filters that can be extremely useful for analysis activities by allowing you to focus on specific packets, based on criteria that you define. You can filter on just the traffic that you want to see or filter undesired traffic out of view. Display filters are one of the most helpful features of Wireshark, so they warrant becoming very familiar with.

Display filters can be created in several ways:

- By applying display filters from the **Display Filter** window
- By typing in the display filter syntax (using autocomplete)
- By applying display filters from the **Conversations** (or **Endpoints**) window
- By applying saved display filters from Filter Expression Buttons
- Using the **Expressions** button for assistance creating filters
- Using right-click menus on specific packet fields

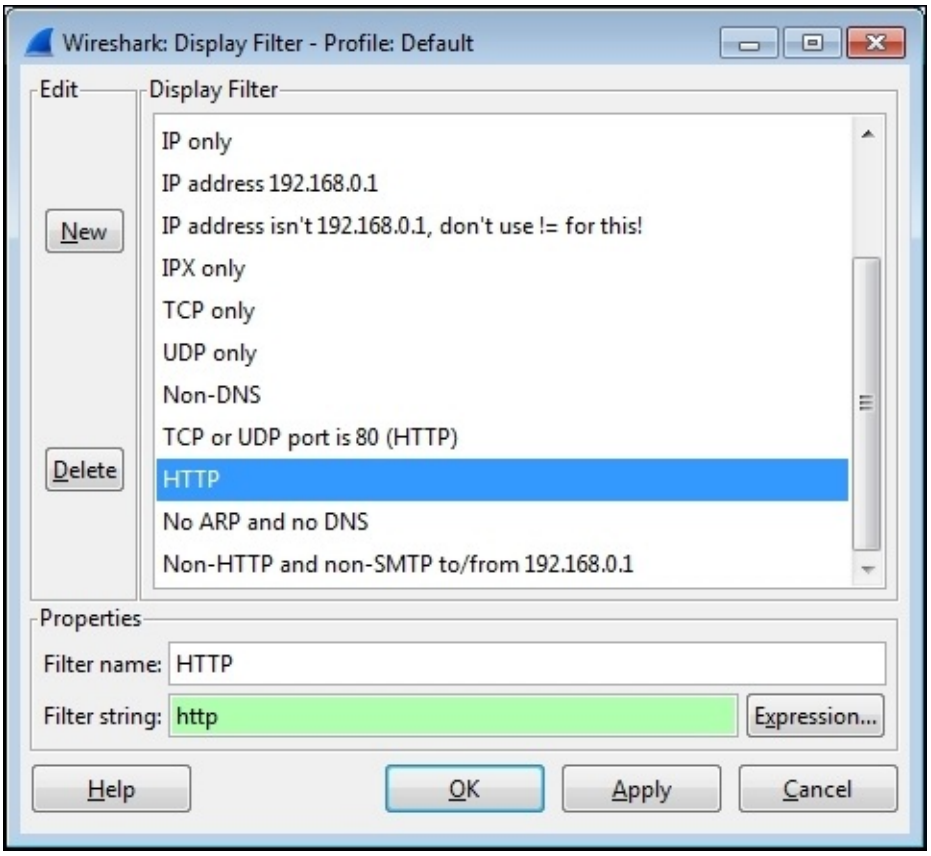
  Note

  Remember that display filters use a proprietary Wireshark filter format, which is protocol-dependent and significantly different from capture filter syntax.

The Display Filter window

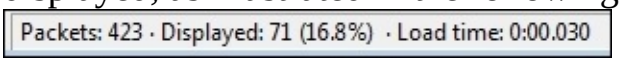
You can open the **Display Filter** window by selecting **Display Filters** from the **Analyze** menu, by clicking on the **Edit/apply display filter** icon on the icon bar, or by just clicking the **Filters** button next to the display filter textbox on the display filter bar.

The **Display Filter** window looks and functions in a similar fashion to the capture filters window, as shown in the following screenshot. You can create a new custom display filter to be added to this window by entering a filter name and the appropriate syntax and clicking on **New** or clicking an existing filter. Click on **New** and modify/rename as per your requirements.



Display filters listed in this window were saved in a `dfilters` file in the Wireshark installation directory for the default profile and in the appropriate **Personal configuration** directory when custom profiles are in use.

When you apply a display filter, the **Status Bar** at the bottom of the Wireshark user interface screen reflects the total number of packets and the packets displayed, as illustrated in the following screenshot:



The display filter syntax

The default selection of capture filters from the **Display Filter** window shown previously provides examples of basic capture filter syntax. Additional examples of display filter syntax are outlined in the following table:

Description	**Syntax**	**Examples**
Basic protocols	`arp`, `bootp`, `dns`, `dhcp6`, `eth`, `snmp`, `smb`, `smb2`, `icmp`, `rtp`, `ip`, `ipv6`, `udp`, `tcp`, `http`, and `sip`	Same as syntax examples
Display filter comparison operators	`eq`, `==`, `ne`, `!=`, `gt`, `>`, `lt`, `<`, `ge`, `>=`, `le`, `<=`, `!`, `not`, `and`, `&&`, `or`, `||`, `XOR`, and `^^`	`ip.addr == 192.168.1.115` and `!(ip.addr == 192.168.1.125)`
Protocol-specific extensions	protocol-specific	`ip.addr`, `tcp.port`, `tcp.dstport`, `tcp.analysis`, `udp.port`, and `udp.srcport`
Classless InterDomain Routing (**CIDR**) notation on IPv4 addresses	`A.B.C.D/CIDR` notation	`ip.addr == 192.168.1.0/24` that matches any IP address in the `192.168.1.0` subnet

Note

Using the `!=` operator on expressions such as `eth.addr`, `ip.addr`, `tcp.port`, and `udp.port` and alike may not work as expected as there are usually two addresses and ports in a packet, and the `!` operator will not match both instances.

Use `!(ip.addr == x.x.x.x)` or a similar syntax for these types of filters.

More information and examples of display filters can be found on the Wireshark wiki at http://wiki.wireshark.org/DisplayFilters and protocol-specific display filter syntax is included in the reference information found at http://wiki.wireshark.org/ProtocolReference.

Typing in a display filter

You can type a display filter syntax directly into the **Filter** textbox in the display filter bar, and then click on **Apply** to apply the filter or **Clear** to clear a filter and start over.

A helpful feature of typing the display filter syntax into the textbox is the autocomplete function. Upon typing a protocol and then a period (.), the textbox will display a list of available protocol-related extensions that can be selected and then the appropriate comparison operator and value added before clicking on **Apply**.

The textbox also has a color-coded background indicating the display filter syntax status. If the syntax is incorrect or incomplete, the background is red and a correct filter results in a green background. A yellow background is a warning that the entered syntax may not work as expected.

Display filters from a Conversations or Endpoints window

Creating a display filter to be applied from a **Conversations** window has already been covered. The same functionality is available from an **Endpoints** window, which can be opened by selecting **Endpoint List** from the **Statistics** menu and one of the listed protocols.

Filter Expression Buttons

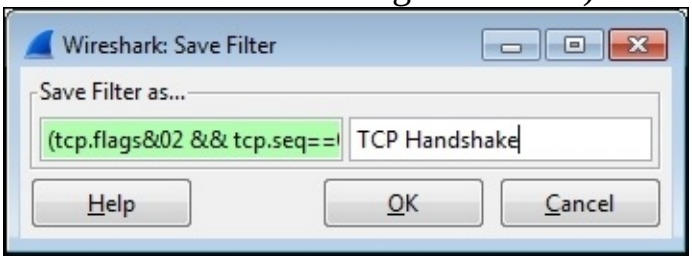
Filter Expression Buttons are buttons you can create that are based on display filters; these can be used to quickly apply previously-saved display filters to your capture data to identify network and application problems.

For example, to create a **Filter Expression Button** option that displays just **TCP SYN**, **SYN/ACK**, **FIN**, or **RST** packets to analyze the TCP session setup parameters, network round-trip delay times, and session terminations:

1. Type the following display filter string into the **Filter** textbox on the **Display Filter Bar**:

```
(tcp.flags&02 && tcp.seq==0) ||  (tcp.flags&12 && tcp.seq==0)
|| (tcp.flags.ack && tcp.seq==1 && !tcp.nxtseq > 0 && !tcp.ack
>1)  || tcp.flags.fin == 1 || tcp.flags.reset ==1
```

- Clicking on **Apply** will apply this filter to a capture that you have loaded so that you can confirm that it is working properly.
- Then, click on **Save** and give the button a name, such as `TCP Handshake` (as illustrated in the following screenshot). Then, click on **OK**:



The filter expression buttons you create will appear on the right-hand side of the initial controls in the display filter bar, as illustrated in the following screenshot:

The filter expression button definitions are stored in the preferences file for the profile you are using. You can edit the button display order, edit the name or

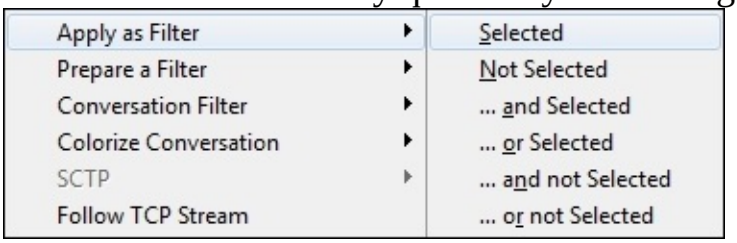
filter syntax, or delete the buttons in Wireshark's Preferences window.

Using the Expressions window button

To the right-hand side of the textbox on the display filter toolbar is the **Expression** button. Clicking on this button opens a **Filter Expression** window that allows you to select a protocol and the extension to that protocol, one of the appropriate relation (comparison) operators, and assign a comparison value. Click on **OK** to populate the display filter textbox with the resultant display filter syntax and then click on **Apply** to apply the filter.

Right-click menus on specific packet fields

If you right-click on a specific field in the **Packet List** or **Packet Details** panes, you can select the **Apply as Filter** or **Prepare a Filter** option and the required submenu option to create display filter syntax, as illustrated in the following screenshot. This is a very quick way of creating display filter syntax:



If you are selecting a field and using the right-click functionality to create display filter syntax, it is usually better to use the **Prepare a Filter** option, which will allow you to edit the syntax before clicking on **Apply** to apply the filter.

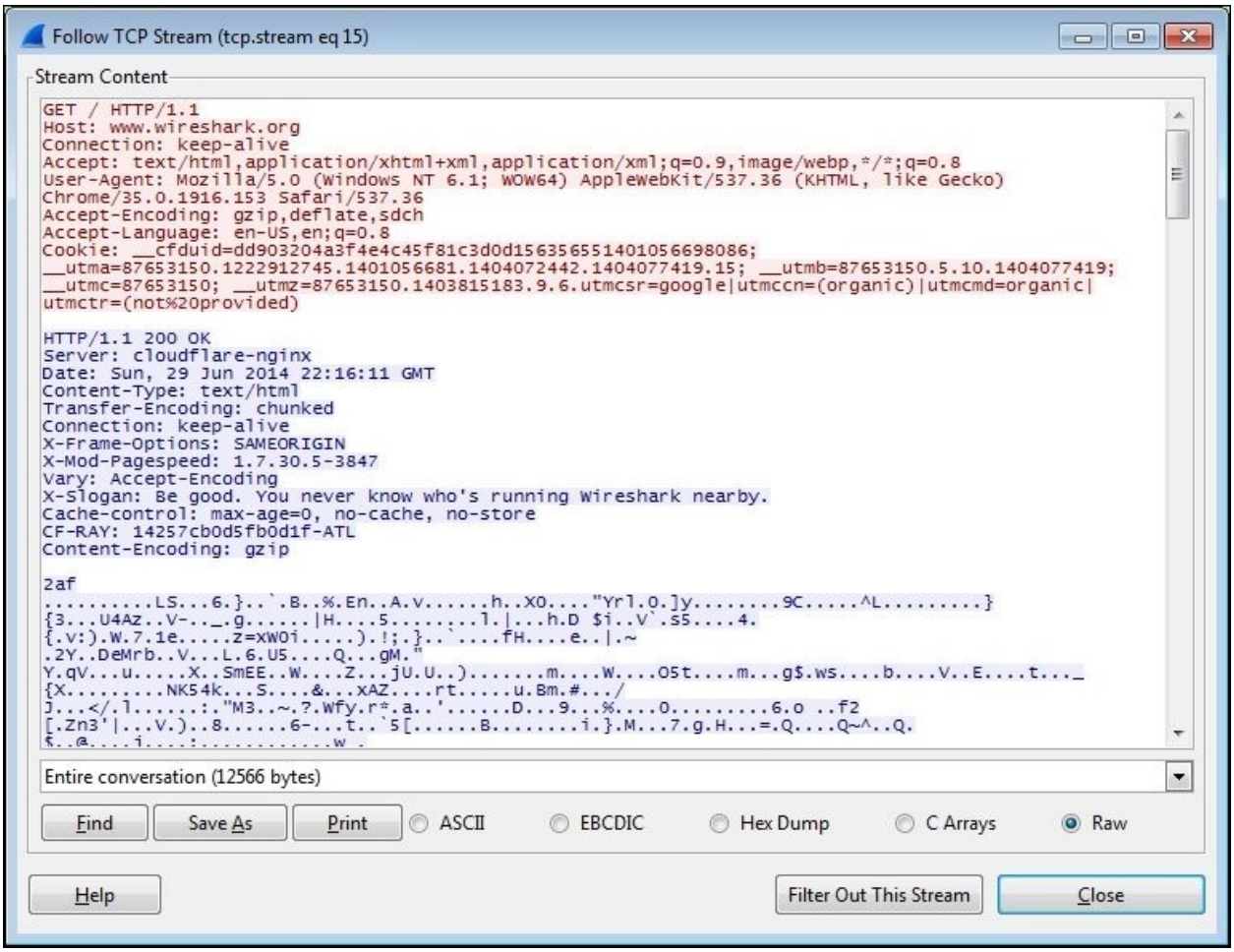
Note

Clicking on a protocol field in the **Packet Details** pane results in that field and the display filter syntax that reflects that field to be displayed in the bottom-left **Status** bar field. This is very helpful for starting a display filter string that will use a particular field.

Following TCP/UDP/SSL streams

Selecting a packet in a conversation, right-clicking, and selecting a **Follow TCP Stream**, **Follow UDP Stream**, or **Follow SSL Stream** option (as appropriate) from the menu provides a display window that contains a textual depiction of the payload data from all of the packets in a conversation. This is an excellent way to inspect the contents of a stream without having to select and inspect multiple packets. Viewing the exchanges between the client and server can be very helpful for troubleshooting purposes.

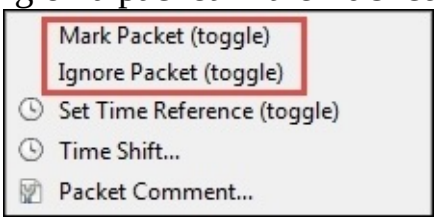
When a **Follow** Stream option is selected for a given packet, a display filter is automatically created and applied to support creation of this window. The following screenshot illustrates a **Follow TCP Stream** window. Also, note the display filter syntax (**tcp.stream eq 15**) that was created and applied when this stream was selected:



Marking and ignoring packets

You can toggle **Mark/Unmark Packet** or **Ignore/Unignore Packet** from the Wireshark **Edit** menu, or by right-clicking on a packet in the **Packet List** pane and selecting **Mark Packet (toggle)** or **Ignore Packet (toggle)**.

The menu displayed by right-clicking on a packet in the **Packet List** pane is



shown in the following screenshot:

Wireshark allows you to mark one or more packets in the **Packet List** pane to make it easier to find those packets later by giving the packet entry a black background with white font. This marking can be toggled on and off on a per-packet basis. Marking a packet has no other effect on the display or packet context.

You can also ignore one or more packets. However, when you invoke the ignore function on a packet that packet entry disappears from the **Packet List**, **Packet Details**, and **Packet Bytes** panes and it effectively ceases (temporarily) to be part of the capture file. Note that ignoring packets can result in Wireshark reporting re-transmissions or other error conditions caused by the missing packet.

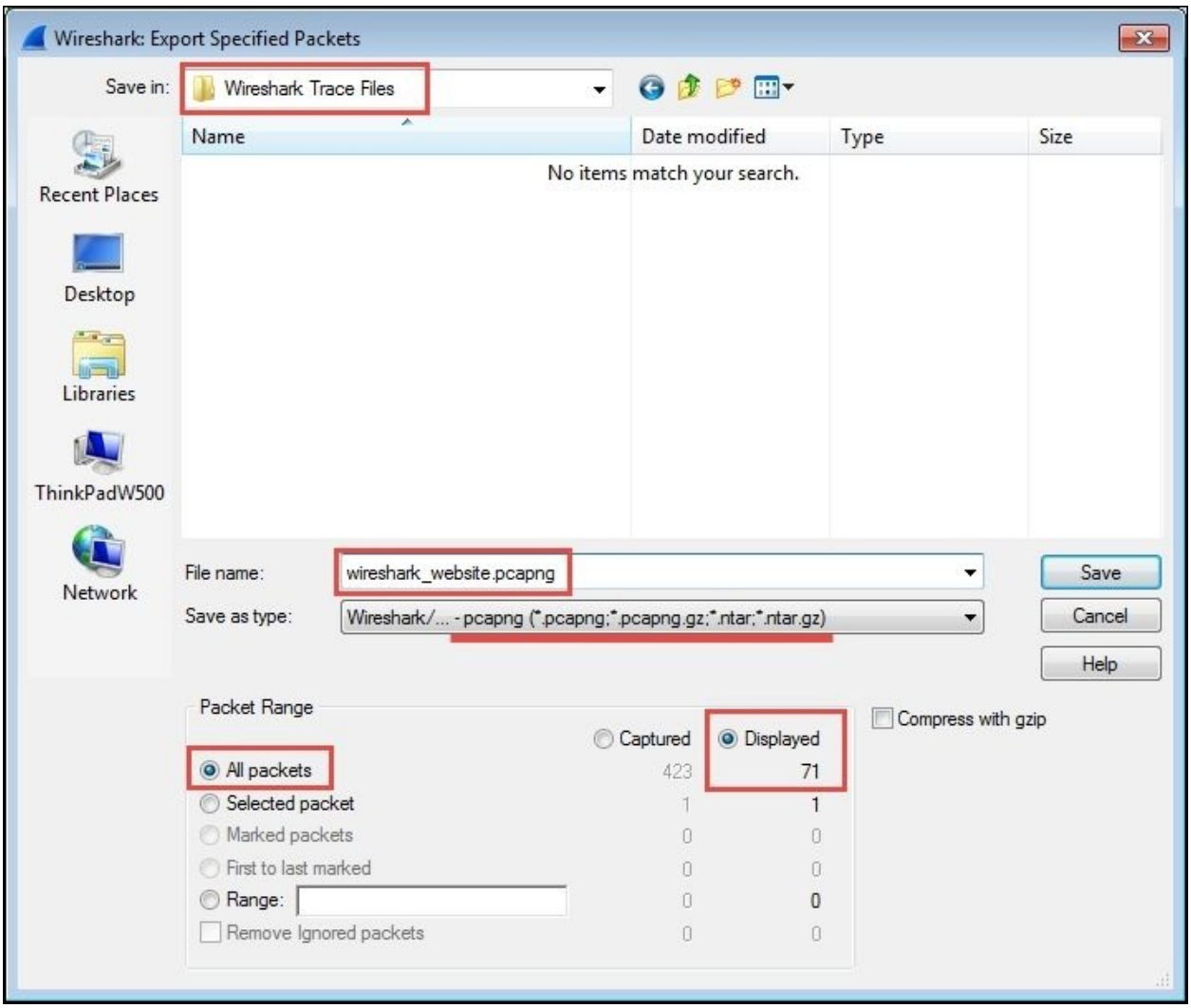
The ignored packets aren't actually deleted from the capture file as you can use the **Reload** option in the **View** menu or click the **Reload** icon on the icon bar to recover the ignored packets.

Saving the filtered traffic

During or after completing an analysis, you will want to save a set of filtered packets into a new capture file. Saving a filtered subset of the bulk capture data and opening the new, smaller file in Wireshark is helpful to reduce the distracting background noise packets displayed when clearing display filters, working with **Conversations** windows, and so on during your analysis. Finally, upon completing your analysis, you will want a filtered capture file that represents the analysis evidence and conclusion and can be quickly loaded for review at a later time.

Use the **Export Specified Packets** option in the **File** menu to save a new capture file consisting of just your filtered packets. Navigate to the desired directory; enter a filename (Wireshark will provide the appropriate filename extension); make the appropriate selections to save all the **Displayed** packets, **Marked packets**, and/or to **Remove Ignored packets**; and then click on **Save**. Remember to save the complete capture using the **Save As** option in the **File** menu as well, because you may need this file again.

The following screenshot illustrates a typical **Export Specified Packets** window and its selections:


Wireshark: Export Specified Packets
Save in:
Wireshark Trace Files
Name
Date modified
Type
Size
No items match your search.
Recent Places
Desktop
Libraries
ThinkPadW500
Network
File name:
wireshark_website.pcapng
Save as type:
Wireshark/... - pcapng (*.pcapng;*.pcapng.gz;*.ntar;*.ntar.gz)
Save
Cancel
Help
Packet Range
Captured
Displayed
Compress with gzip
All packets
423
71
Selected packet
1
1
Marked packets
0
0
First to last marked
0
0
Range:
0
0
Remove Ignored packets
0
0

Summary

The important points covered in this chapter included picking an optimal capture point, selecting between TAPs and mirrored/SPAN ports, Wireshark's capture filters and options, verifying a good capture, using Wireshark's **Conversation** windows and display filters to isolate packets of interest, creating Filter Expression Buttons, marking and ignoring packets, and saving the filtered traffic for later or more detailed analysis.

In the next chapter, we'll cover the rest of Wireshark's basic packet analysis features.

Chapter 4. Configuring Wireshark

Wireshark offers a number of features that can be configured to enhance the accuracy and ease of performing packet analysis activities such as troubleshooting a functional or performance problem. Selecting the best format to measure the elapsed time between packets is an important factor. There are a number of protocol-specific options that affect how Wireshark displays time-related information that are useful as well. Coloring rules, preferences settings, and profiles let you customize Wireshark for your particular style of analysis, as well as the different environments that you might work in.

In this chapter, we will cover the following topics:

- Working with packet timestamps
- Colorization and coloring rules
- Wireshark preferences
- Wireshark profiles

These topics will wrap up our introduction to the most essential and useful features and options of Wireshark.

Working with packet timestamps

Understanding how Wireshark handles time and using the right incarnation of packet timestamp displays is crucial to properly analyze packet flows and identify time-related anomalies.

How Wireshark saves timestamps

When packets are captured, Wireshark gives each packet a timestamp derived from the system clock of the machine from where the capture takes place. This timestamp is converted to **Universal Coordinated Time** (**UTC**) based on an offset calculated from the time zone setting and any **Daylight Savings Time** (**DST**) rules that apply for the capture machine, and then converted again to an epoch number (the UTC-based number of seconds since January 1, 1970). This is the time value that gets saved in the capture file for each packet. When Wireshark reads the capture file, it turns the epoch number back to the familiar date and time display, adjusted for the time zone and DST offsets for your machine.

This means that if a packet capture is conducted on a machine in Los Angeles, which has an offset from UTC of -8 hours, and you look at the same capture file on a machine in Berlin, which is UTC +1 hour (an overall difference of 9 hours, plus any DST differences), a packet that was captured at 10 a.m. local time in Los Angeles will display a timestamp of 7 p.m. in Berlin.

Examples of the timestamp displays in different time zones are shown in the following table:

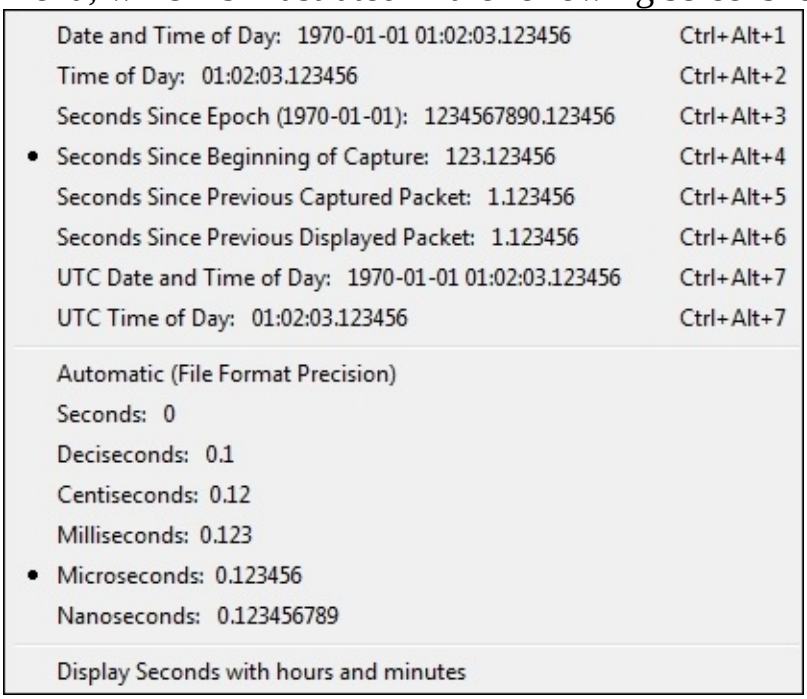
	Los Angeles	**London**	**Berlin**	**Bangalore**
Capture file time (UTC)	10:00	10:00	10:00	10:00
Local offset to UTC	-8	0	+1	+5:30
Displayed time (local time)	02:00	10:00	11:00	17:30

If you're going to look at a packet capture someone has sent you and the absolute time when an event occurred is important to the analysis, you'll need to know or ask what time zone the capture was taken in, determine the offset between your time zone and the capture location time zone, and mentally make the time difference adjustment for the timestamps that Wireshark will display. Otherwise, this difference won't matter as you're usually more interested in the elapsed time or the time between specific events in the capture.

Wireshark time display options

There are a wide variety of packet time displays available for use in Wireshark. By default, Wireshark provides a **Time** column in the **Packet List** pane configured to display **Seconds Since Beginning of Capture** with microsecond resolution (`123.123456`) for each packet.

However, the way in which time is displayed in this column can be changed by selecting the desired format from the **Time Display Format** option in the **View** menu, which is illustrated in the following screenshot:

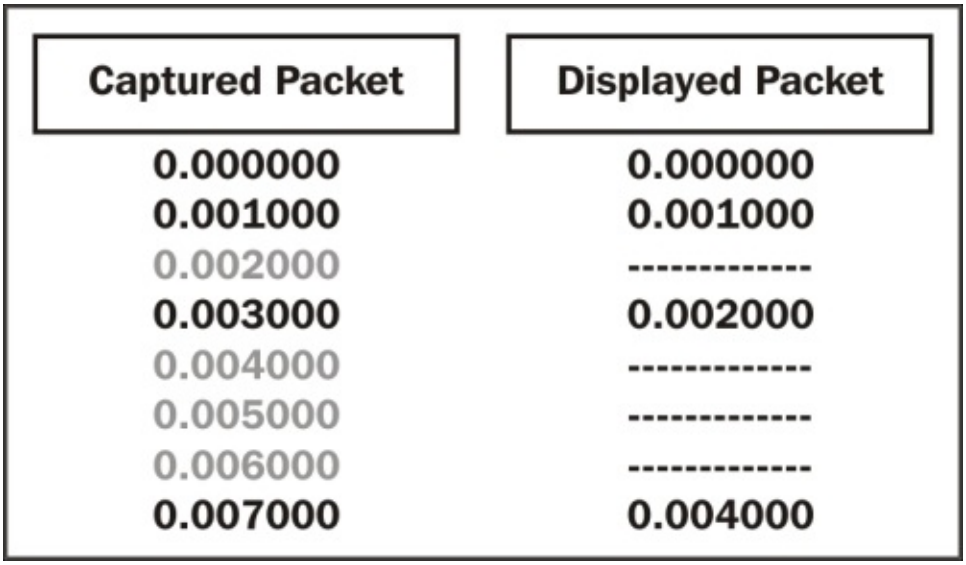


If the **Seconds Since Beginning of Capture** option is in use, the first packet in a capture displays a time value of **0.000000**; all other packets are timed in reference to this first packet such that the elapsed time from the beginning of the capture is displayed.

The time display menu options provide examples of their formats and are fairly self-explanatory, except perhaps **Seconds Since Previous Captured Packet** and **Seconds Since Previous Displayed Packet**. The **Seconds Since Previous Captured Packet** option provides the elapsed time between each captured

packet, while the **Seconds Since Previous Displayed Packet** option displays the elapsed time from the previous packet that is visible when a display filter is applied.

The way the **Displayed Packet** option works is illustrated in the following screenshot. You can see how the **Captured Packet** timestamps continue to increment, while the **Displayed Packet** timestamps show the time since the last displayed packet.



Captured Packet	Displayed Packet
0.000000	0.000000
0.001000	0.001000
0.002000	----------
0.003000	0.002000
0.004000	----------
0.005000	----------
0.006000	----------
0.007000	0.004000

The time display precision options in the **Time Display Format** menu are also shown with examples of the display format and are self-explanatory, except for the **Automatic (File Format Precision)** setting, which requires a description.

Wireshark relies on the NIC driver and the capture devices' system clock for packet timestamps. The accuracy of these timestamps in terms of the precision and number of subsecond digits (milliseconds, microseconds, and nanoseconds) will vary, but usually a millisecond resolution is available. This precision value is saved in the capture file. The **Automatic (File Format Precision)** setting tells Wireshark to display timestamps using this precision value.

The ability to use the **Nanoseconds precision** setting depends on having an NIC driver that supports this level of precision. If you select this option and the capture file doesn't contain the higher resolution, the last three digits of each

timestamp will be all zeroes.

Adding a time column

It is often helpful to have two (or more) time columns in the **Packet List** pane to provide a variety of time display types without having to change the format of a single time column back and forth. You can add a new time column using one of two methods.

The following is the first method, the preferences settings method:

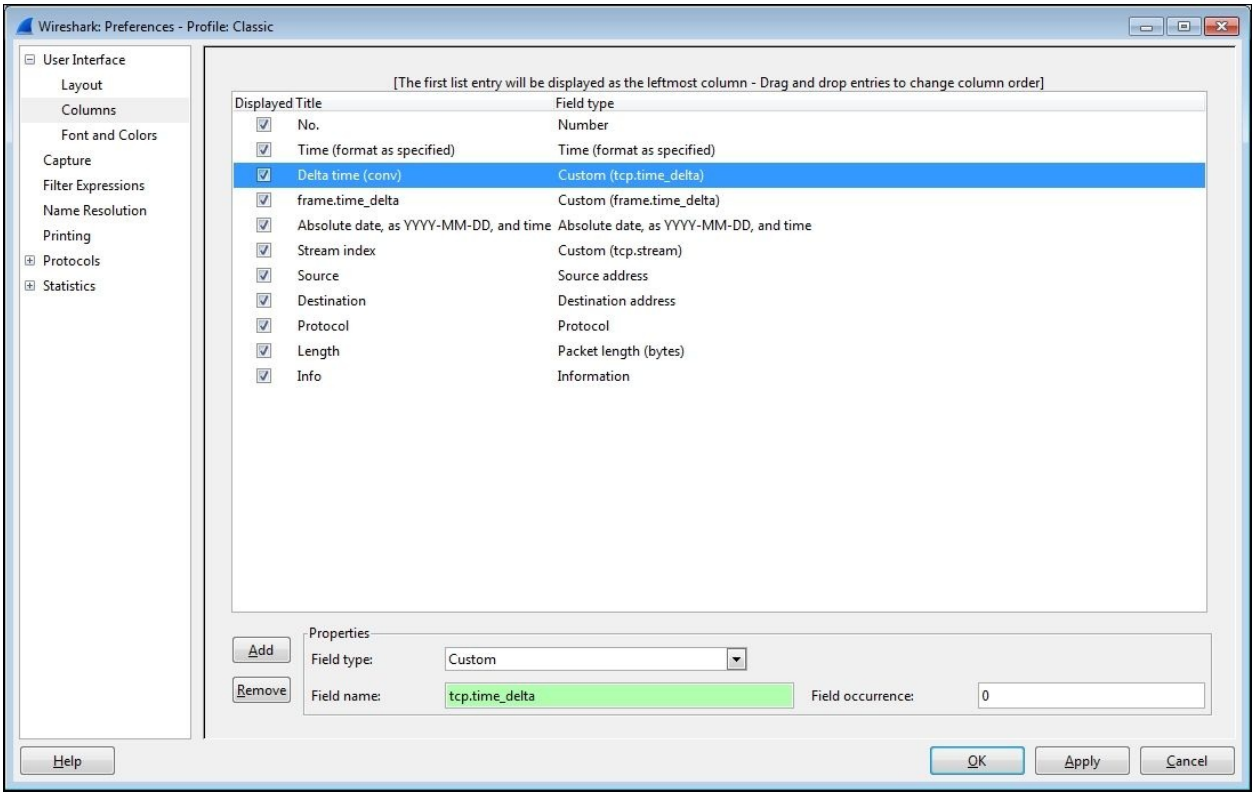
1. Go to **Preferences** from the **Edit** menu, or click on the **Preferences** icon to open the **Preferences** window.
2. Select **Columns**.
3. Click on **Add** to add a new entry at the bottom of the list.
4. Click on the **Title** area of the new entry and give the column a name.
5. Ensure that the new entry is highlighted, and select the desired time display format from the drop-down **Field type** box.
6. Click and drag the new entry up the list to select its relative position in the **Packet List** pane.
7. Finally, click on **OK**.

The selectable options in the **Field type** box for time display columns include the following:

- **Absolute date, as YYYY-MM-DD, and time**: This is the actual capture date and time based on the time zone of the capture device.
- **Absolute date, as YYYY/DOY, and time**: This is another format to display the date and time based on the time zone of the capture device.
- **Relative time**: This is the time from the first packet in a capture file. This is similar to the **Seconds Since Beginning of Capture** option.
- **Relative time (conversation)**: This is the time from the first packet in the trace file for a conversation (this doesn't work).
- **Delta time**: This is the elapsed time from the previous packet to the current packet.
- **Delta time (conversation)**: This is the time from the previous packet to the current packet in a conversation (this doesn't work).
- **Delta time displayed**: This is the time from the end of one packet to the end of the next displayed packet only.

- **Custom**: The **Relative time (conversation)** and **Delta time (conversation)** options, which are also listed in the preferences settings, no longer work in the version of Wireshark currently available (v1.12) as of this writing. You can accomplish the previously offered functionality with these options by using the **Custom** option with display filter-style **Field** types instead. Select the **Custom Field** type and enter either `tcp.time_relative` or `tcp.time_delta` in the **Field name** field, leaving the **Field occurrence** field with the default entry of **0**.

An example of creating a functional **Delta time (conv)** time column using the **Custom** option and the **tcp.time_delta** display filter is shown in the following screenshot:

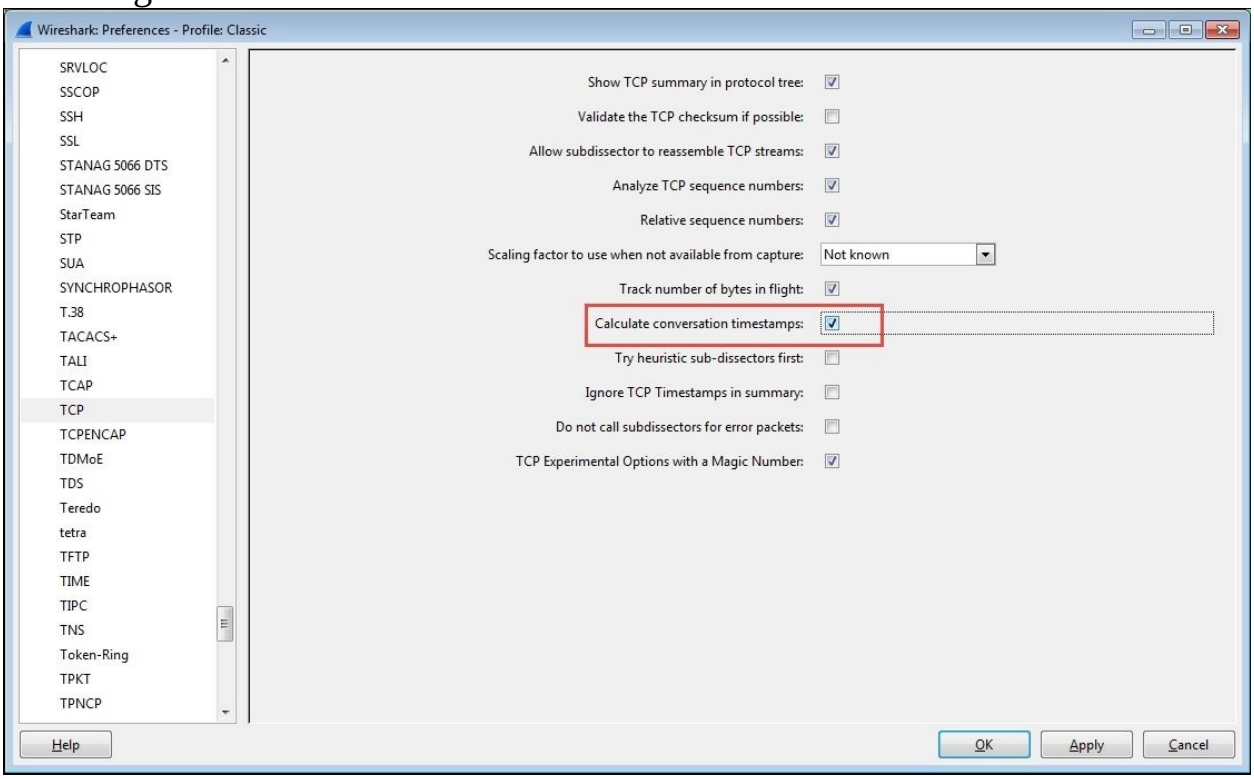


For the **tcp.time_relative** and **tcp.time_delta** fields to work properly, you must also enable **Calculate conversation timestamps** in the preferences settings using the following steps:

1. In the **Preferences** window, select **TCP** from the **Protocols** menu.

2. Enable the **Calculate conversation timestamps** option.
3. Finally, click on **OK**.

An example of enabling **Calculate conversation timestamps** is depicted in the following screenshot:



The following steps show you the second method, the right-click method of adding a column:

1. Select an appropriate packet in the **Packet List** pane.
2. In the **Packet Details** pane, expand the **Frame** header, or if applicable, expand the **Transmission Control Protocol** header.
3. Locate the desired time value field in the **Frame** or **TCP** sections (these are surrounded by brackets). If you are selecting a time value in the TCP section, you will need to expand the **[Timestamps]** section to see the values.
4. Right-click on the desired time field and select **Apply as Column** from the menu.

5. The new column will appear beside the **Info** column in the **Packet List** pane. Click and drag the new column to the desired location.
6. You can right-click on the new column header, select **Edit Column Details**, and give the column a shorter name if desired.

As previously discussed in the preferences settings method, you must enable **Calculate conversation timestamps** in the **TCP** protocol option of the preferences settings to view and use the time values in the **TCP** section.

Conversation versus displayed packet time options

The difference between time displays for a conversation versus a displayed packet time option is perhaps subtle but important.

As illustrated previously, if you are using one of the displayed packet time options, the time value shown for a given packet will be the elapsed time since the previous packet was displayed in the **Packet List** pane. This time value option has no useful value until you apply a display filter, after which you can easily see the elapsed time between each packet being displayed with no other mental math or adjustments necessary. This is very useful if you're sequentially filtering, clearing, and viewing more than one conversation using, for example, a **tcp.stream==xx** display filter setting.

If you are not using a display filter, however, there may be packets from multiple conversations displayed in the **Packet List** pane. If you are using one of the conversations time displays, the time value shown for a given packet will be the elapsed time since the previous packet for that conversation, regardless of other packets that may be interspersed and visible between the packet you're looking at and the previous packet in that conversation. This allows a quick perusal of conversation packet times without having to apply a display filter.

Choosing the best Wireshark time display option

With so many time display options available, it may be difficult to know when and where to use a given option. Choosing the optimal time display in a Wireshark time column depends greatly upon the objectives of the analysis:

- If you need to know the specific date and time of day when an event occurred in a capture, as might be the case if you're trying to find and correlate packets with user-reported events or log entries, you should use one of the **Absolute time** formats.
- If you're looking for an event that occurred some known period of time after a capture started, use one of the **Relative time** formats.
- On the other hand, if you just need to measure the time between certain packets, such as when measuring the time between a request and a response, one of the **Delta time** formats will be the most helpful.

Using the Time Reference option

Another useful Wireshark feature is the **Time Reference** menu option, which can be used to measure time from one packet to another in the midst of a capture file. You can click on a specific packet and toggle this option on and off for that packet using either the **Set/Unset Time Reference** option from the **Edit** menu, or by right-clicking and selecting the **Set Time Reference (toggle)** option from the pop-up menu. The packet will be marked with a ***REF*** designator in the first time column, and any relative timestamps following the **Time Reference** packet will be displayed relative to that packet.

The **Time Reference** setting is temporary; it isn't saved to a capture file and will disappear if you reload the file.

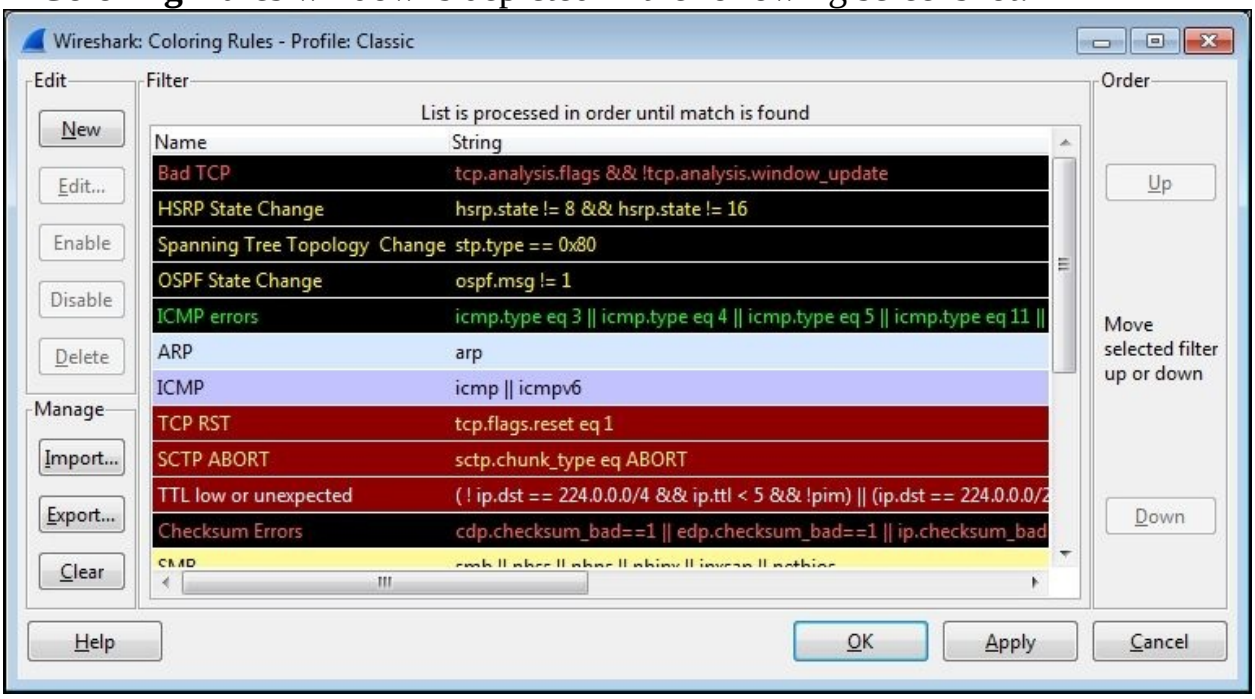
Colorization and coloring rules

Colorization of packets displayed in the **Packet List** pane can be an effective tool to identify and highlight packets of interest, especially the packets that contain or indicate some kind of error condition.

Wireshark has predefined coloring rules that are enabled by default and which can result in a kaleidoscope of colored packets in the **Packet List** pane. You can enable or disable the coloring rules by selecting **Colorize Packet List** from the **View** menu or by clicking on the **Colorize Packet List** icon in the icon bar if this becomes overwhelming.

You can also view, enable/disable, add, delete, reorder, and edit the coloring rules by selecting **Coloring Rules** from the **View** menu or by clicking on the **Edit Coloring Rules** icon in the icon bar. There is a **Clear** button that removes all the changes you may have made to the rules and restores them to default settings if needed.

A **Coloring Rules** window is depicted in the following screenshot:



Coloring rules employ display filter formats with specific values to identify packets that should be colored. The rules are compared to packets starting with the top rule and working down through the list. Only the first rule that matches a packet's condition is applied, so the ordering of the rules dictates which rule gets applied if more than one rule matches a packet. If you create or modify a rule, you have to check the ordering to make sure you get the desired behavior.

Clicking on a rule and then clicking on **Edit** allows you to modify the foreground and background colors for that rule, as well as change the filter string if desired.

You can also export/import coloring rules if you want to share them with others. Coloring rules are stored in a file called `colorfilters` in one of your personal configuration directories depending on the profile in use.

Packet colorization

You can also temporarily color a series of packets in a conversation by selecting one of the conversation packets, selecting **Colorize Conversation** from the **View** menu, and selecting a color from the adjoining menu, or by right-clicking on a packet, selecting **Colorize Conversation** from the menu, selecting one of the protocol-specific options, and then selecting the desired color. This colorization will disappear when the capture file is reloaded, or you can select **Reset Coloring 1-10** from the **View** menu.

Wireshark preferences

In the *Adding a time column* section, we opened the **Preferences** window using **Preferences** in the **Edit** menu or by clicking on the **Preferences** icon in the icon bar to configure the time display column options. There are quite a number of **Preferences** options that you should be aware of and may want to adjust to customize your Wireshark environment:

- **Layout**: This is used to select the ordering of the **Packet List**, **Packet Details**, and **Packet Bytes** panes.
- **Columns**: This is used to add, remove, and move columns in the **Packet List** pane.
- **Capture**: This is used to set the default capture options.
- **Filter Expressions**: This is used to add, remove, or move the **Filter Expression** buttons.
- **Name Resolution**: This is used to set the MAC, transport, and network (IP) resolution options.
- **Protocols**: There are options that can be set for all of the protocols that Wireshark supports; some of the most important and useful of these options include:
    - **HTTP**: This is used to add any additional TCP ports that should be recognized as HTTP traffic in your environment.
    - **IEEE 802.11**: This is used to add/edit the **Wireless Decryption** keys if needed to decode an encrypted wireless session.
    - **IPv4**: You may want to disable **Validate IPv4 checksum if possible** to avoid inadvertent error messages caused by an **NIC** option called checksum offloading, wherein checksums are checked after the packet is sent to Wireshark.
    - **RTP**: Enable **Allow subdissector to reassemble RTP streams** to support decoding audio from VoIP captures.
    - **SMB**: Enable **Reassemble SMB Transaction payload** to support exporting file objects from an SMB stream in a packet capture.
    - **SSL**: Wireshark can decrypt the SSL/TLS traffic if you have the private key file. To add a key to Wireshark, go to the **Preferences** window and click on the **RSA keys list Edit** button. Then, in the **SSL Decrypt** window, click on **New** and complete the **SSL Decrypt: New** fields (**IP address** of the SSL server; **Port**, which is usually 443 for

HTTP; **Protocol**, such as HTTP; and **Key File**, which is used to select the path to an RSA private key (if the key file is a PKCS#12 keystore (usually has a `.pfx` or `.p12` extension), the **Password** field must be completed)), and finally, click on **OK** to close each subsequent window.

- **TCP**: This provides you with multiple options, as follows:
  - **Validate TCP checksum if possible**: Disable this to avoid inadvertent error messages caused by checksum offloading.
  - **Allow subdissector to reassemble TCP streams**: Enable this to support exporting file objects from a TCP stream.
  - **Relative sequence numbers**: Enable this to make it easier to read and track TCP sequence numbers in a capture file.
  - **Track number of bytes in flight**: This is a value calculated and displayed in the TCP protocol header in the **Packet Details** pane, which is useful for performance analysis.
  - **Calculate conversation timestamps**: This is the setting discussed earlier that is needed to support the **tcp.time_relative** and **tcp.time_delta** time displays.

There are numerous other preferences settings that may be pertinent to your personal preference or analysis environment; you will have to investigate most or all of these options. If you are unsure of a particular setting, you can get more information by clicking on the **Help** button at the bottom of the **Preferences** window.

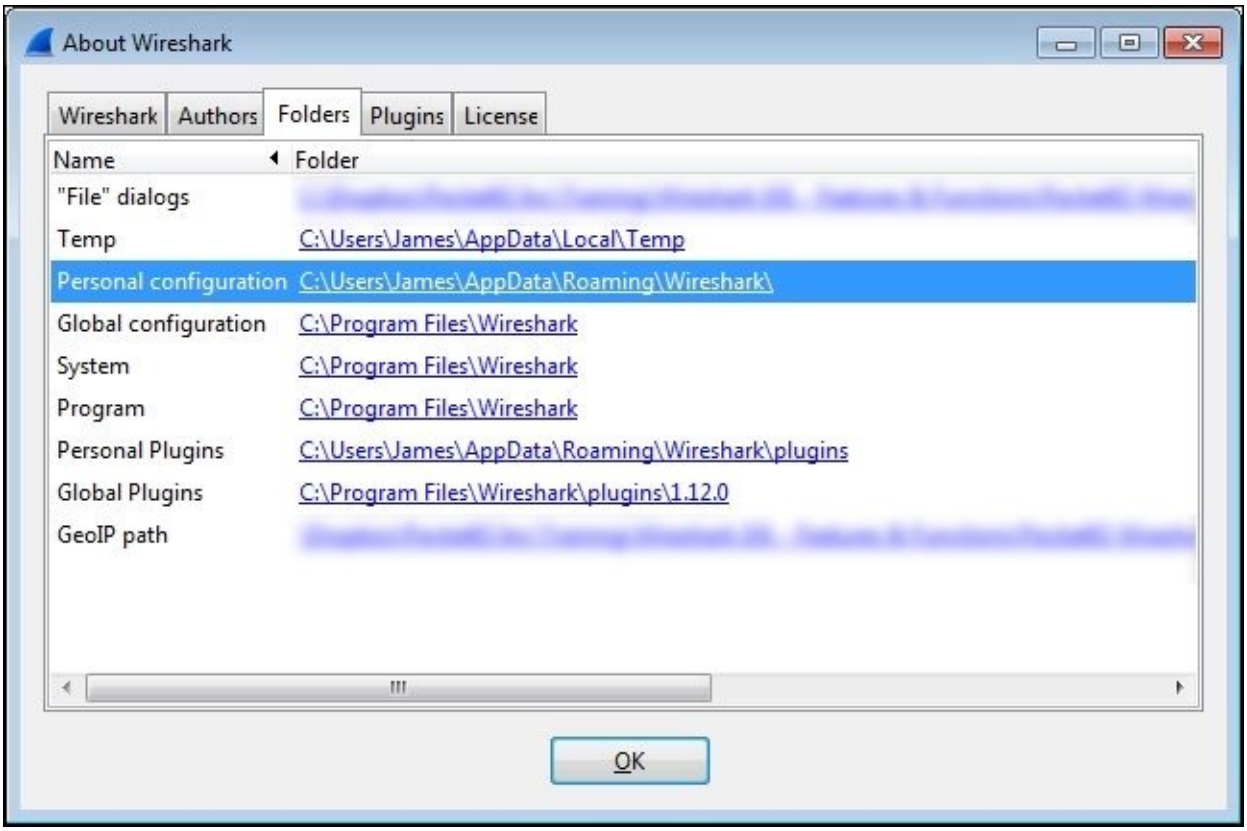
The preferences settings are stored in a file called `preferences` in one of your **Personal configuration** directories, depending on the profile in use.

Wireshark profiles

As we have covered the numerous Wireshark configuration options that are saved in specific files, such as `cfilters` for **Capture Filters**, `dfilters` for **Display Filters**, `colorfilters` for **Coloring Rules**, and `preferences` for preferences settings, it was mentioned that these files were saved in one of your **Personal configuration** directories, but I have left a full explanation of profiles and these configuration directories until now so that you would better understand what makes up a profile and why they are useful.

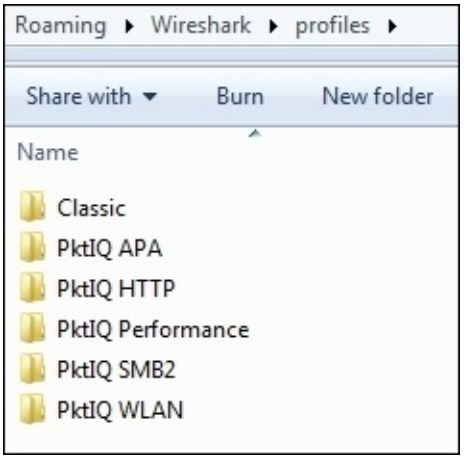
A profile is a collection of Wireshark configuration files customized for your specific needs and tastes in capture and display filters, coloring rules, columns and layouts, and so on for the particular environment you are working in. You can create one or more profiles and quickly reconfigure Wireshark to work best in differing environments by selecting the appropriate profile.

When you first install Wireshark, it operates with a default set of configuration files that are located in the **Global configuration** directory, which is usually the same as the **System** directory where the Wireshark program files reside. When you change any of the default settings, the changes are saved in new configuration files that are stored in a **Personal configuration** directory, the location of which varies depending upon your installation. You can determine and quickly open the **Personal configuration** directory for your installation from Wireshark by clicking on the **About Wireshark** option in the **Help** menu and clicking on the **Folders** tab. Within this tab is a list of all the directories that Wireshark uses, as shown in the following screenshot:



You can double-click on a Wireshark directory link to open a window to that directory.

Double-clicking on the **Personal configuration** link in the **Folders** tab opens the directory where (under a `profiles` subdirectory) your custom profile files are stored. Each profile is stored in a separate subdirectory that reflects the name you give a profile, as shown in the following screenshot:



Each custom `profile` directory contains all the Wireshark configuration files that determine how that profile controls Wireshark's features. You can copy and share these custom profile directories with other Wireshark users; copying the `profile` directory into their **Personal configuration** directory makes that profile available for selection.

Creating a Wireshark profile

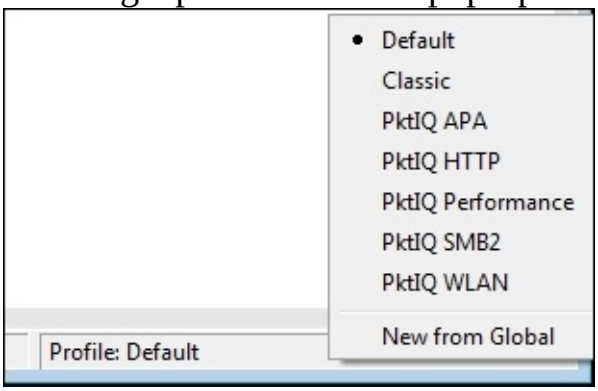
To create a new Wireshark profile, follow these steps:

1. Right-click on the **Profile** section (on the right-hand side pane) of **Status Bar** at the bottom of the Wireshark user interface and click on **New**, or navigate to **Edit** | **Configuration Profiles** | **New** in the menu bar.
2. In the **Create New Profile** window that appears, you can give the profile a name. You can also choose to create the profile starting with the settings from an existing profile by making a selection from the **Create from** drop-down list or start from scratch. The **Create New Profile** window is shown in the following screenshot:

- Clicking on **OK** will save the new profile in its own directory by the same name in your `Profiles` directory in the **Personal configuration** menu.

Selecting a Wireshark profile

You can select one of your custom profiles by selecting **Configuration Profiles** from the **Edit** menu, clicking on one of the listed profiles, and clicking on **OK**. A quicker method is just clicking on the **Profile** section of **Status Bar** and selecting a profile from the pop-up menu, as shown in the following screenshot:



Summary

The topics covered in this chapter included working with Wireshark's time displays, colorization and coloring rules, selecting the appropriate Wireshark preferences for a given analysis environment, and saving all of these settings in profiles that can be selected as required.

In the next chapter, we'll cover a selection of network layer, transport layer, and application layer protocols in common use in modern networks, which will help you to prepare for more advanced packet analysis activities in the later chapters.

Chapter 5. Network Protocols

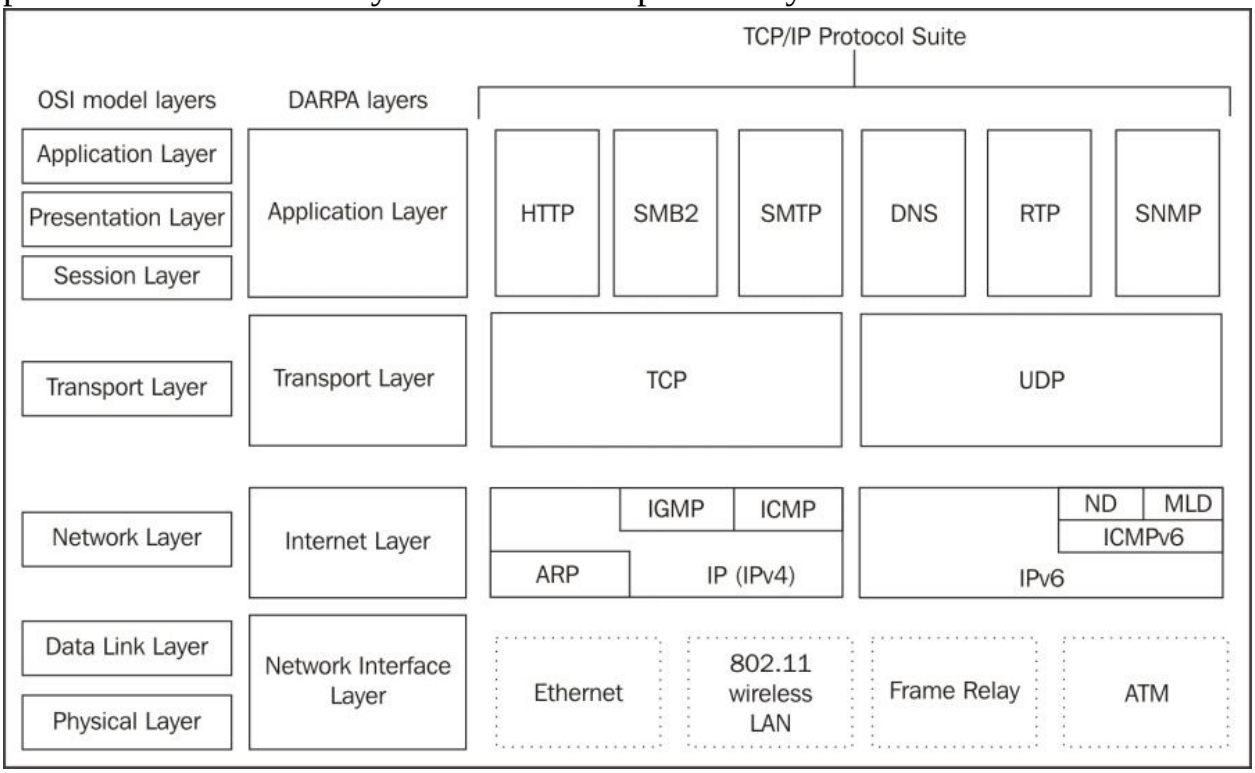
Effective packet analysis requires familiarity with the primary protocols in use in modern networks. In this chapter, we will review the most common protocols in their respective layers:

- Network layer protocols
- Transport layer protocols
- Application layer protocols

We'll cover the significant purpose and relevant fields to support network connectivity and/or application functionality in each protocol, as well a sampling of Wireshark capture and display filters for each protocol.

The OSI and DARPA reference models

We reviewed the purpose of the OSI and DARPA reference models in Chapter 2, *Networking for Packet Analysts*. The visual depiction of their layers is repeated in the following diagram as a reference and summary of some of the primary protocols and where they fit into their respective layers:



Network layer protocols

Network layer protocols, also known as Internet layer protocols in the DARPA reference model, provide basic network connectivity and internetwork communications services. In this layer, you will predominantly find the IP protocol being used to get packets transported across the network, along with ARP, IGMP, and ICMP.

We covered the IP and ARP protocol packet header structures and fields in Chapter 2, *Networking for Packet Analysts*, so this information won't be repeated. However, basic Wireshark capture and display filters are provided here and also for the remaining protocols in the following sections:

Wireshark IPv4 filters

Capture filter(s): `ip`

Display filter(s): `ip ip.addr==192.168.1.1 ip.src== ip.dst== ip.id > 2000`

Wireshark ARP filters

Capture filter(s): `arp`

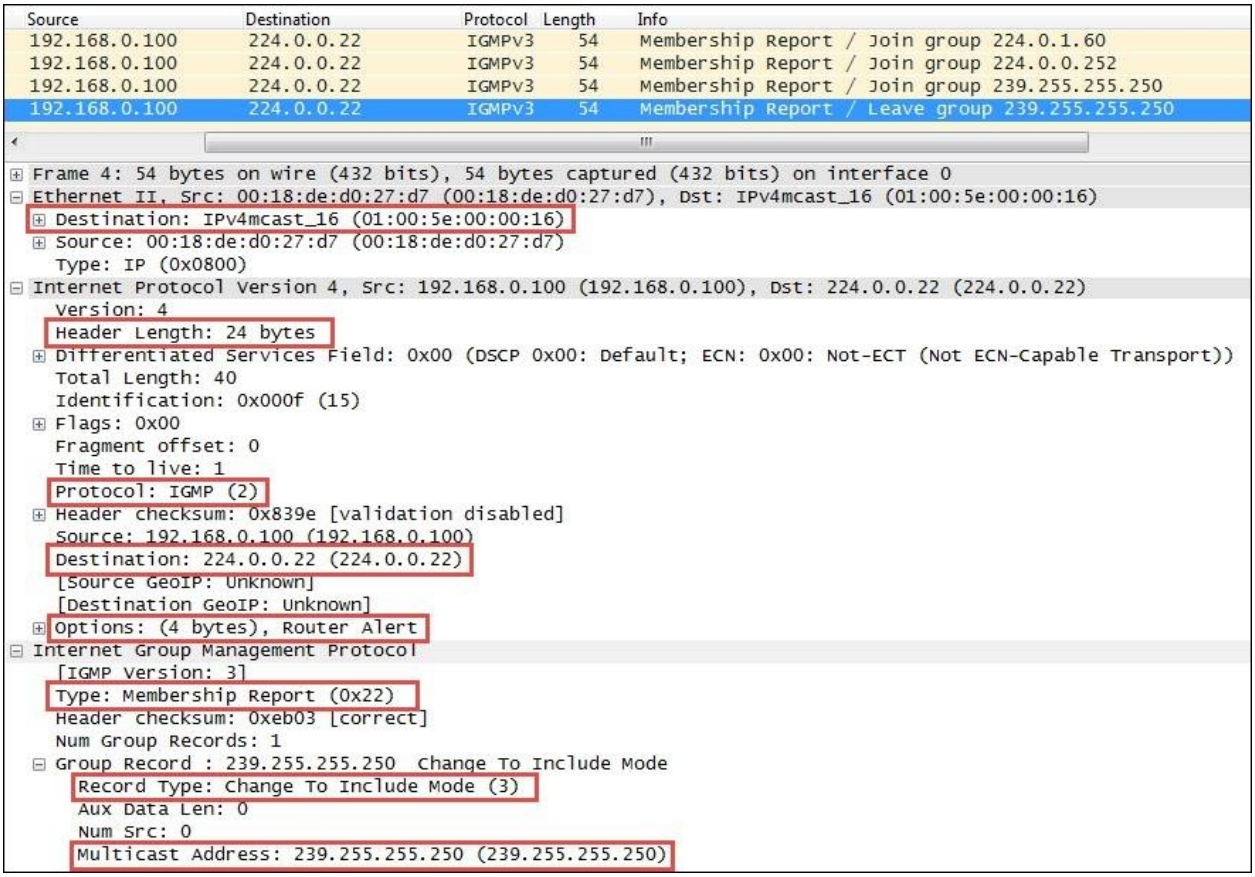
Display filter(s): `arp arp.opcode==1 arp.src.hw_mac==00:1c:25:99:db:85`

Internet Group Management Protocol

The **Internet Group Management Protocol** (**IGMP**) is used by hosts to notify adjacent routers of established multicast (one-to-any) group memberships. In other words, IGMP enables a computer that provides content (video feeds), for example, to provide such content to a distributed group of users using one set of the multicast address ranges (in the `224.0.0.0` to `239.255.255.255` class D multicast range). This multicast capability depends on routers that are capable and configured to support this service; clients must join the multicast group. When a host wants to start a multicast, it sends an **IGMP Membership Report** message to the `224.0.0.2` (all multicast routers) address that specifies the multicast IP address for this particular group. Clients who wish to join or leave this group (so they can receive the multicast content) send an IGMP join or leave message to the router. The following table shows the various ranges for addresses:

Starting address range	**Ending address range**	**Description**
`224.0.0.0`	`224.0.0.255`	These are reserved for special well-known multicast addresses
`224.0.1.0`	`238.255.255.255`	These are globally-scoped (Internet-wide) multicast addresses
`239.0.0.0`	`239.255.255.255`	These are locally-scoped and administered multicast addresses

The following screenshot shows the significant fields in the IGMP protocol header:



The preceding significant fields in the IGMP protocol header include:

- **Type**: This is a type of IGMP message. Type 22 is IGMPv3 **Membership Report**.
- **Record Type**: There are different types of **Group Records**. The value of **Record Type 3** is **Change To Include Mode**, which indicates that content from the source device is to be forwarded to the in-group hosts by the multicast router.
- **Multicast Address**: This is the multicast IP address for a specific group.

You should also note the following interesting fields in the previous protocol layers:

- The Ethernet frame destination MAC address is one of a range of multicast MAC addresses (`01:00:5e:00:00:00 – 01:00:5e:7f:ff:ff`)
- The **Protocol** field in the IP header specifies IGMP **2**

- The IP layer destination IP Address is `224.0.0.22`, which is a reserved IGMPv3 multicast IP address

The IGMP protocol has multiple versions and is rather complex. Refer to the protocol references provided at the beginning of this chapter for more information.

Wireshark IGMP filters

Capture filter(s): `igmp`

Display filter(s): `igmp igmp.type==0x22 igmp.record_type==4 igmp.maddr==244.0.1.60`

Internet Control Message Protocol

The **Internet Control Message Protocol** (**ICMP**) is used by network devices such as routers to send error messages indicating that a requested service is not available, or a host or network router could not be reached. ICMP is a control protocol. This means that although it is transported as IP datagrams, it does not carry the application data—instead, it carries the information about the status of the network itself.

ICMP pings

One of the most well-known uses of ICMP is to ping, wherein a device sends an ICMP echo request (**Type 8**, **Code 0**) packet to a distant host (via that host's IP address), which will (if the ICMP service isn't disabled or blocked by an intermediate firewall) respond with an ICMP echo reply (**Type 0**, **Code 0**) packet. Pings are used to determine whether the target host is available and can be reached over the network. By measuring the time that expires between ping requests and replies, we know the **round trip time** (**RTT**) delay time over the network path.

ICMP traceroutes

A variation of ping functionality is used to perform a traceroute (also known as traceroute), which is a list of the IP addresses of the router interfaces that packets traverse to get from a sending device to a target host or device. The traceroutes are used to determine or confirm the network path taken from a sending device to a target host or device.

A traceroute is accomplished by sending the ICMP echo request packets to a distant host just as in a normal ping, but with modifications to the **Time-to-Live** (**TTL**) field in the IP header of each packet. The traceroute function takes advantage of the fact that each router in a network path decrements the TTL value in a packet by 1, so as the packet traverses, the routers in a path and the TTL value will decrease accordingly along the way. If a router receives a packet with a TTL value of 1, it will send an ICMP TTL exceeded in transit (**Type 11**, **Code 0**) error message back to the sender (along with a copy of the request packet it received) and otherwise discard (not forward) the packet.

The traceroute works by sequentially setting the TTL in multiple ICMP request packets to 1, then to 2, then 3, and so on, which results in each router in the network path sending TTL exceeded error messages back to the sender. Since these returned messages are sent by the in-path router using the IP address of the interface where the ICMP packet was received, the traceroute utility can build and display a progressive list of router interface IP addresses in the path and the RTT delay to each router.

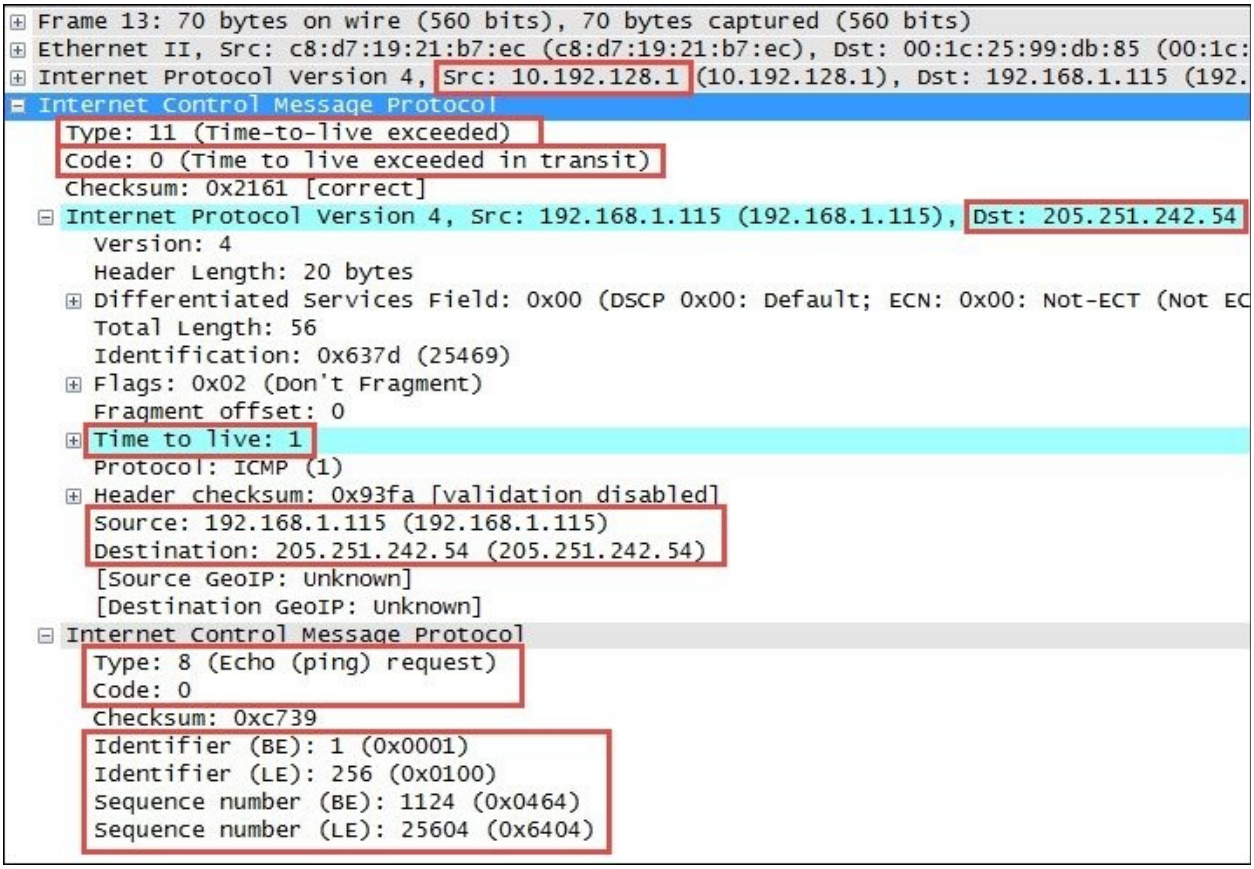
ICMP control message types

A sampling of the most commonly seen types of ICMP control messages, including their type and code (subtype) numbers, are provided in the following table:

Type	Code	Description
0	0	This indicates echo reply (ping)
3	0	This indicates destination network unreachable
3	1	This indicates destination host unreachable
3	4	This indicates fragmentation required and do not fragment bit set
3	6	This indicates destination network unknown
3	7	This indicates destination host unknown
5	0	This indicates redirect datagram for the network
5	1	This indicates redirect datagram for the host
8	0	This indicates echo request (ping)
11	0	This indicates TTL expired in transit (seen in traceroutes)

The Wireshark packet details fields for the ICMP packet illustrated in the

following screenshot depict a **Time-to-live exceeded** message as seen in a typical traceroute capture:



The following points are significant to analyze this packet:

- The source IP address seen in the IPv4 header summary is **10.192.128.1**, which is the IP address of the router interface sending the ICMP message to the originator, **192.168.1.115**
- The ICMP packet is **Type 11**, **Code 0** (TTL exceeded in transit)

The second set of IPv4 and ICMP headers that follow the first IPv4 and ICMP headers are copies of the original packet transmitted by the sender. This copy is returned to allow determination of the packet that caused the ICMP message. The significant points in the packet details of this ICMP message copy include:

- The target destination IP address, where the echo request packet was intended to be sent (and would have been if the TTL value hadn't been

altered) is `205.251.242.51`.

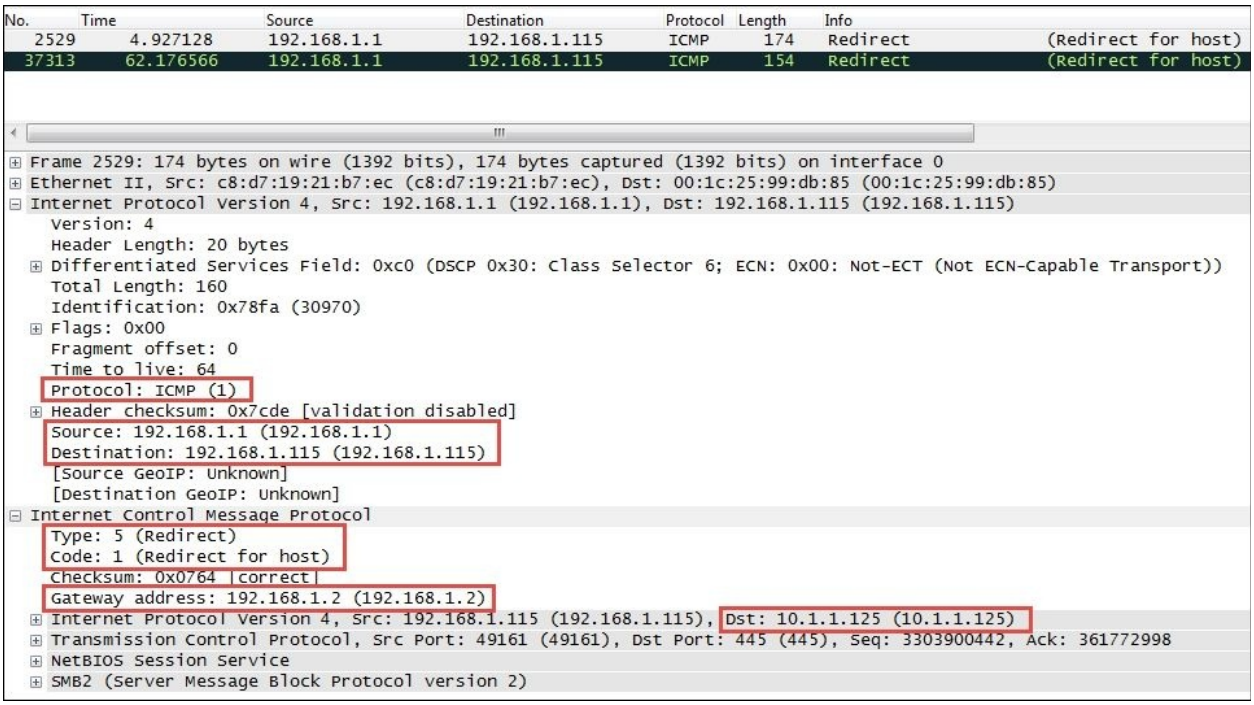
- The TTL value was **1** when this packet reached the `10.192.128.1` router interface. This packet cannot be forwarded, resulting in the TTL exceeded message being sent back to the sender.
- The original ICMP packet was a **Type 8**, **Code 0** echo request message.
- The **Header Data** section of the ICMP packet for the echo requests and replies will include a 16-bit identifier and 16-bit sequence number, which are used to match echo replies to their requests.

ICMP redirects

Another common use of ICMP is to redirect a client to use a different default gateway (router) to reach a host or network than the gateway it originally tried to use. In the ICMP **Redirect** packet depicted in the following screenshot, a number of packet fields should be noted:

- The source IP address of the ICMP redirect packet is `192.168.1.1`, which was the client's default gateway; this is the router sending the redirect packet back to the client
- The ICMP **Type** is **5** (**Redirect**) and **Code** is **1** (**Redirect for host**)
- The gateway IP address that the router `192.168.1.1` is telling the client to use to reach the desired target host is `192.168.1.2`
- The IP address of the target host was `10.1.1.125`

The following screenshot shows the ICMP **Redirect** packets:



Wireshark ICMP filters

Capture filters(s): `icmp`

Display filter(s): `icmp icmp.type==8 || icmp.type==0 (pings) icmp.type==5`

`&& icmp.code==1 (host redirects)`

Internet Protocol Version 6

The **Internet Protocol Version 6** (**IPv6**) is the latest version of Internet protocol, and although it is in its earliest stages of adoption, it is intended to eventually replace IPv4—mostly to alleviate the shortage of IP addresses that can be assigned to network devices. IPv4, with its 32-bit address space, provides approximately 4.3 billion addresses, nearly all of which have been assigned to companies and private interests worldwide.

IPv6 utilizes a 128-bit address space, which allows 2^{128} or approximately 3.4 x 10^{38} addresses; that number is 340,282,366,920,463,463,374,607,431,768,211,456 unique addresses.

IPv6 addressing

The 128 bits of an IPv6 address are represented in eight groups of 16 bits each, written as four hexadecimal digits separated by colons (:). An example of an IPv6 address is `2001:0db8:0000:0000:0000:ff00:0042:8329`.

For convenience, an IPv6 address may be abbreviated to shorter notations by application of the following rules, wherever possible:

- One or more leading zeroes from any groups of hexadecimal digits are removed; this is usually done to either all or none of the leading zeroes. For example, the hexadecimal group 0042 can be converted to just 42.
- Consecutive sections of zeroes are replaced with a double colon (::). The double colon may only be used once in an address, as multiple use would render the address indeterminate. A double colon must not be used to denote a single section of omitted zeroes.

An example of applying these rules to IPv6 addresses is as follows:

- **Initial address**: `2001:0db8:0000:0000:0000:ff00:0042:8329`
- **After removing all leading zeroes**: `2001:db8:0:0:0:ff00:42:8329`
- **After omitting consecutive sections of zeroes**: `2001:db8::ff00:42:8329`

The 128 bits of an IPv6 address are logically divided into a network prefix and a host identifier. The **Class Inter-Domain Routing** (**CIDR**) notation is used to

represent IPv6 network prefixes, for example, `2001:DB8:0:CD30::/64` represents network `2001:DB8:0000:CD30::`.

IPv6 address types

There are three basic types of IPv6 addresses:

- **Unicast**: These packets from one-to-one device use a single interface address. Unicast addresses can be of one of the following three types:
  - **Global Unicast**: This is routable to and over the Internet. Global Unicast addresses generally start with `2xxx` (such as `2000::/3`).
  - **Link-local**: This is automatically assigned to an interface and used on the local network link; this is not routable to the Internet, much like a MAC address. Link-local Unicast addresses start with `FE80` (`FE80::/10`). They are automatically assigned to an interface when it is initialized using an algorithm that uses a rearranged version of the NIC's 48-bit MAC address in the IPv6 address and are used to communicate on the local link. These addresses are not routable. IPv6 uses link-local addresses for neighbor discovery functions.
  - **Unique local**: This is not routable to the Internet, but it is routable within an enterprise (similar to IPv4 private addresses). Unique local Unicast addresses start with `FC00` (`FC00::/7`). This block of addresses is reserved for use in private IPv6 networks.
- **Multicast**: These are packets from one-to-many devices. Multicast addresses start with `FFxx`. An example of a multicast address is `FF01:0:0:0:0:0:0:101`, which can be shortened to `FF01::101`. There is no broadcast address in IPv6; multicasts are used as a replacement. Some well-known multicast addresses are shown in the following table:

Address	Description	Scope
`ff01:0:0:0:0:0:0:1`	All nodes address	Interface-local (spans only a single interface on a node useful only for loopback transmission of multicast packets)
`ff02:0:0:0:0:0:0:1`	All nodes address	Link-local (all nodes on the local network segment)
`ff01:0:0:0:0:0:0:2`	All routers address	Interface-local

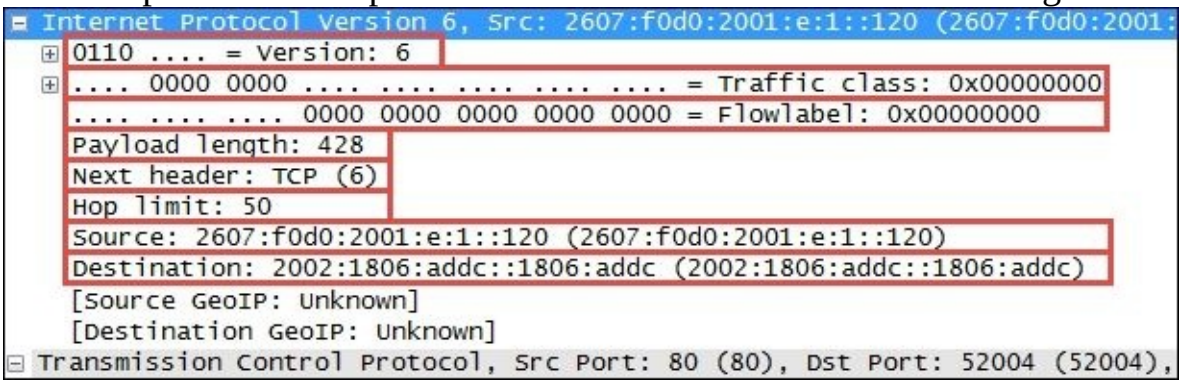
ff02:0:0:0:0:0:0:2	All routers address	Link-local
ff05:0:0:0:0:0:0:2	All routers address	Site-local (spans a single site)
ff02:0:0:0:0:0:1:2	DHCPv6 servers/agents	Link-local
ff05:0:0:0:0:0:1:3	DHCPv6 servers/agents	Site-local

- **Anycast**: These packets are from one to the nearest of a group of interfaces. There is no special addresses scheme for Anycast addresses; they are similar to Unicast addresses. An Anycast address is created automatically when a Unicast address is assigned to more than one interface. Anycast addresses can be used to set up a group of devices so that any one of the group devices can respond to a request sent to a single IPv6 address.

Further discussion of IPv6 addressing would cover quite a number of additional features, which are beyond the scope of this book. The reader is encouraged to research IPv6 addressing further online and/or by reading Request For Comments (RFC) 4291 (IP Version 6 Addressing Architecture).

IPv6 header fields

An example of an IPv6 protocol header is illustrated in the following screenshot:



The IPv6 header fields are similar to many IPv4 headers and the fields include:

- **Version**: This is the IP version number, 6 for IPv6.
- **Traffic class**: This is similar to the IPv4 **DiffServ** field; it is used to identify different classes or priorities of IPv6 packets.
- **Flow label**: These are used to identify sequences of packets that are labeled as a set. An IPv6 flow is defined by the 20-bit **Flow Label** field and the source and destination IPv6 address fields.
- **Payload length**: This is the length of the IPv6 payload, not including any packet padding.
- **Next header**: This field indicates what's coming next in the packet. This is equivalent to the IPv4 **Protocol** field. In the preceding example, the next layer is a normal **TCP (6)** header.
- **Hop limit**: This field is roughly equivalent to the **Time To Live** field in IPv4; it is decremented by one by each device that forwards the IPv6 packet. When the value reaches one, the packet cannot be forwarded.
- **Source and Destination addresses**: These are the 128-bit IPv6 source and destination addresses.

IPv6 supports extension headers that provide additional information fields and that also extend the length of the IPv6 header. There is specific **Next Header** code that indicates the presence of this added functionality.

IPv6 transition methods

As part of the transition to IPv6, the current TCP/IP devices support dual stacks (IPv4 and IPv6 simultaneously) and the ability to encapsulate and tunnel IPv6 packets inside IPv4 packets so that they can be routed by IPv4 networks. The three of the most popular encapsulation methods are:

- **6to4 tunneling**: In this tunneling method, an IPv6 header follows an IPv4 header; the **Protocol** field of the IPv4 header will contain `41` (IPv6), and the source IPv6 address in the IPv6 header will start with `2002`.
- **Teredo**: In this tunneling method, an IPv6 header is encapsulated inside a UDP packet. This method was developed to accommodate NAT devices that do not handle protocol 41. Teredo tunneling can be identified in the UDP packet header by a destination port of `3544`.
- **ISATAP**: This tunneling method uses a locally assigned IPv4 address to create a 64-bit interface identifier. For example, in ISATAP, the IPv4

address `24.6.173.220` becomes `::0:5EFE:1806:addc`. ISATAP encapsulates IPv6 headers within IPv4 as in 6to4 tunneling.

Wireshark IPv6 filters

Capture filter(s): `ip6 host fe80::1 ip proto 41 (capture IPv6-over-IPv4 tunneled traffic)`

Display filter(s): `ipv6 ipv6.addr == fe80::f61f:c2ff:fe58:7dcb ipv6.addr == ff02::1`

Internet Control Message Protocol Version 6

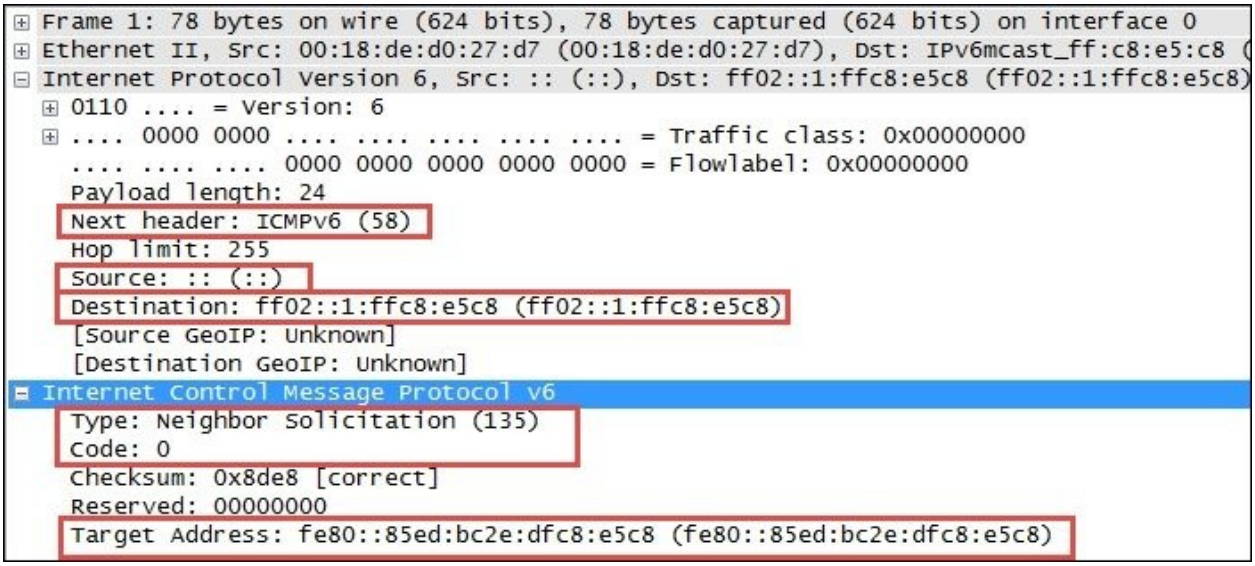
Internet Control Message Protocol Version 6 (**ICMPv6**) is an integral part of IPv6, and the base protocol must be fully implemented by every IPv6 node. ICMPv6 provides services for an IPv6 environment that are provided by other distinct protocols in an IPv4 environment, such as Neighbor Solicitation to replace ARP.

The following table contains some of the common ICMPv6 packet types:

ICMPv6 packet type	ICMPv6 type	Purpose
Echo request	128	Ping request
Echo response	129	Ping response
Multicast listener query	130	Sent by multicast router to poll a network segment for group members
Multicast listener report	131	Sent by a host when it joins a multicast group, or in response to a multicast listener query sent by a router
Multicast listener done	132	Sent by a host when it leaves a multicast group and might be the last member of that group on the network segment
Router solicitation	133	Discover the local router(s)
Router advertisement	134	Respond to Router Solicitation messages, as well as sending this packet after initialization and periodically afterwards
Neighbor solicitation	135	Used first for Duplicate Address Detection (using a source address of : :) and then to obtain the MAC address of the local router; this function replaces ARP
Neighbor advertisement	136	Response to Neighbor Solicitation messages

Redirect message	137	Redirect a device to the proper router to send packets to a specific network or host

An example of a Neighbor Solicitation ICMPv6 packet is shown in the following screenshot:



The significant fields in this packet include:

- **Next Header**: This field contains **58**, which indicates that the next protocol header is to be ICMPv6.
- **IPv6 Source Address**: The presence of an unspecified address (**::**) indicates this is a **Duplicate Address Detection** packet.
- **IPv6 Destination Address**: This is basically a multicast address.
- **ICMPv6 Type**: This is a Neighbor Solicitation message using Type **135**.
- **ICMPv6 Code**: This is the subtype for Neighbor Solicitation messages; this will be **0**.
- **ICMPv6 Target Address**: This is the address the host wants to use. If another node on the network is already using this address, they will respond accordingly.

Multicast Listener Discovery

Multicast Listener Discovery (**MLD**) is another component of the IPv6 suite used by IPv6 routers to discover multicast listeners on a directly attached link.

MLD is part of the ICMPv6 protocol and it replaces IGMP on IPv4 networks.

Wireshark ICMPv6 filters

Capture filter(s): `icmp6`

Display filter(s): `icmpv6 icmpv6.type==1135 && icmpv6.code==0 (Neighbor Solicitation)`

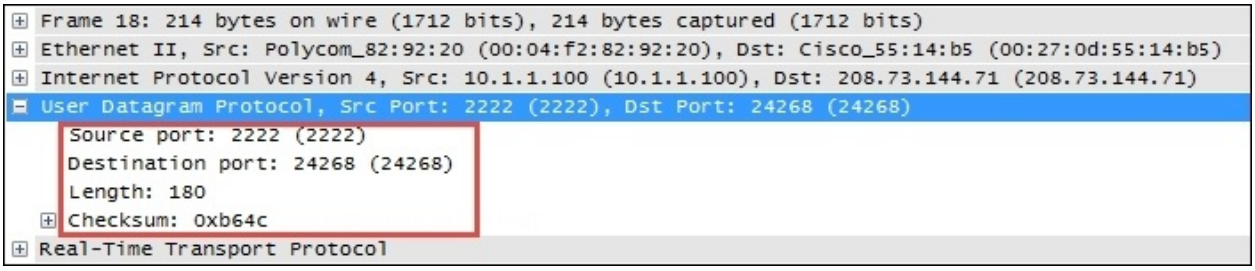
Transport layer protocols

The transport layer protocols include TCP and UDP used to transport application protocols.

User Datagram Protocol

The **User Datagram Protocol** (**UDP**) is considered an unreliable transport. In this, there's no guarantee of packet delivery or ordering, but it has a lower overhead and is used by time-sensitive applications such as voice and video traffic.

The following screenshot shows the fields contained in an UDP header:



The UDP header is only 8-bytes long, consisting of:

- **Source and Destination port number**: This is 2 bytes each.
- **Length**: This is the length of the UDP header plus the payload. This is a 2-byte field.
- **Checksum**: This is a 2-byte field used to check for errors in the UDP header and data. If no checksum was generated by the transmitter, this will be all zeroes.

Wireshark UDP filters

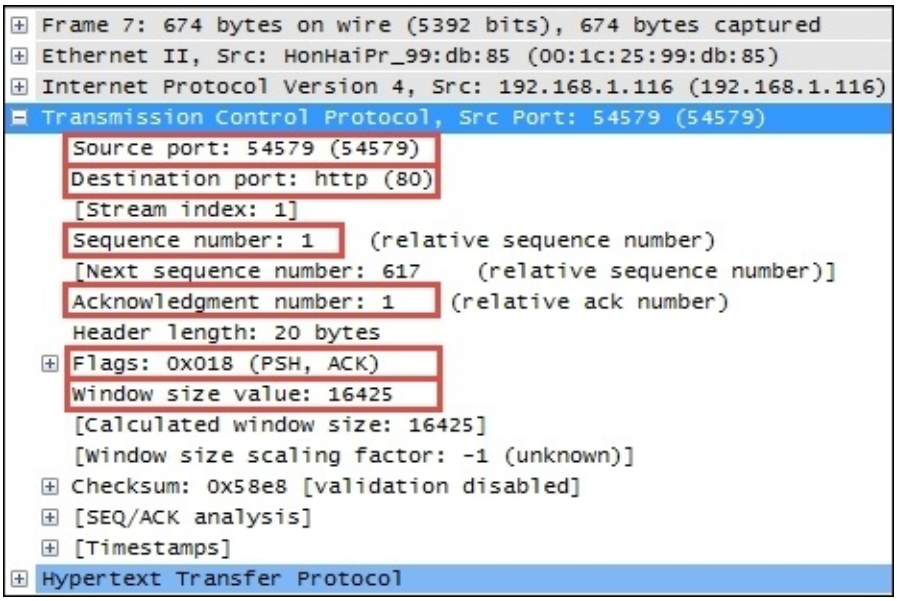
Capture filter(s): `udp udp port 2222`

Display filter(s): `udp udp.srcport == 161 (SNMP response) udp.length > 256`

Transmission Control Protocol

The **Transmission Control Protocol** (**TCP**) provides a reliable delivery of data by detecting lost, duplicated, or out-of-order packets, requesting retransmission of lost data, or rearranging packets in the right order before delivering them to the application. TCP can also accept a large chunk of data from an application and handle getting the data transported to the other end reliably using multiple packets and reassembling them at the other end.

The following screenshot highlights the significant fields of a basic TCP header:



The TCP header contents and length can vary depending on options that may be in use, but in its simplest implementation it consists of:

- **Source port and Destination port**: These are well-known and registered ports are used (on servers) to access standard application services such as HTTP, FTP, SMTP, databases, and so on. Port numbers assigned to client/user sessions are usually in a higher number range and assigned sequentially.
- **Sequence number**: This is a number that represents the first octet in any given segment. Sequence numbers are initialized at the beginning of new sessions as a random number, and then incremented as data bytes are sent.

- **Acknowledgment number**: When the ACK flag bit is set, this field contains the next sequence number expected from the sender, which in turn acknowledges receipt of all the bytes received up to that point.

  ### Note

  The use of sequence and acknowledgment numbers is how TCP ensures reliable delivery of data by tracking the number and order of received bytes.

  Sequence and acknowledgment numbers are large and difficult for humans to follow. Wireshark can convert and display these as relative values that start with 0 at the beginning of a session to make it easier to inspect them and relate the values to the number of bytes transmitted and received.

- **Flags**: These bits are used to control connection setups, terminations, and flow control mechanisms.
- **Window size**: This field indicates the current size of the buffer on this host used to store received data until it can be handed off to the receiving application. This information enables the sending host to adjust data flow rates in case of network or host congestion.

TCP flags

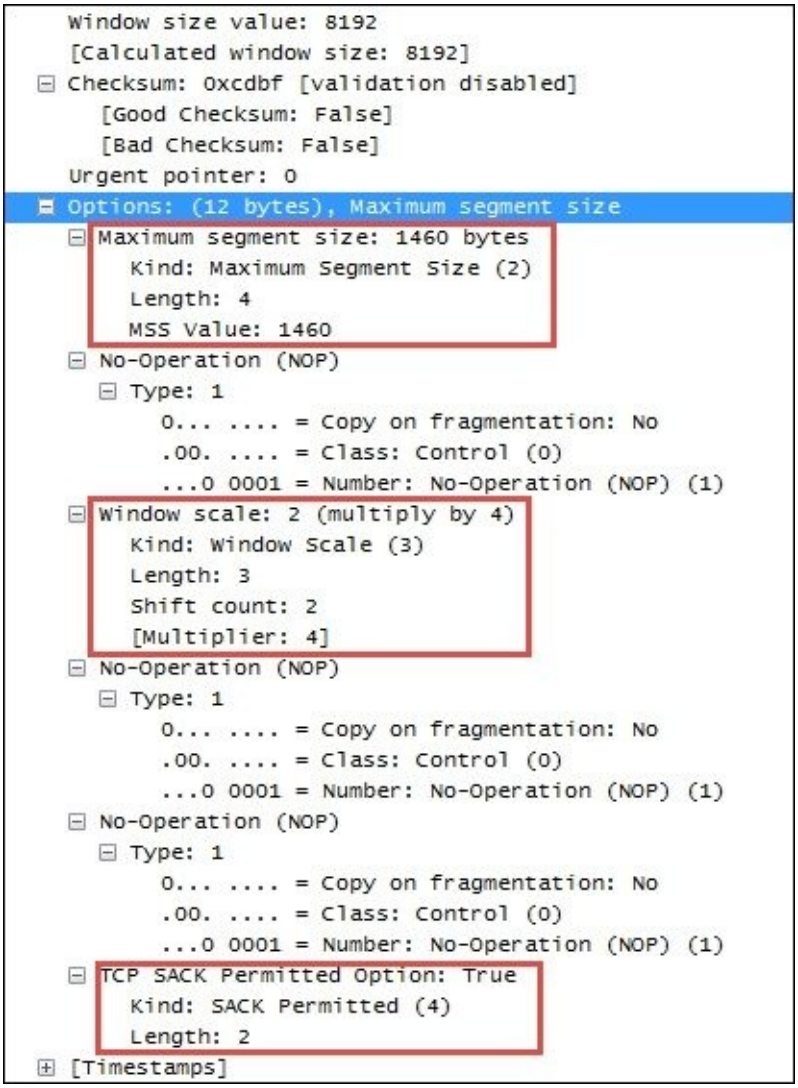
The following table lists the flags that are most commonly used in a TCP header:

Flag field name	Description
URG (urgent)	This indicates the **Urgent Pointer** field (after the TCP header checksum) that should be examined. This flag is normally 0; the **Urgent Pointer** field is only examined if this bit is set.
ACK (acknowledgment)	This is the acknowledgment packet.
PSH (push)	This indicates whether the sending node's TCP stack should bypass any buffering and pass the data directly to the network and on to the receiving application.
RST (reset)	This is used to close the connection explicitly.
SYN	This is used to synchronize sequence numbers and used in a three-way TCP session

(synchronize)	initiation handshake process.
FIN (finish)	This is used when the transaction is finished. This does not mean that the connection is to be closed explicitly, but is commonly seen at the end of sessions.

TCP options

The TCP also supports a number of additional options, several of which are in common use in modern networks that you should be aware of. The snippet of a TCP header illustrated in the following screenshot depicts several of the most popular options:



The TCP options highlighted in the preceding screenshot include:

- **Maximum Segment Size**: This option allows you to specify of the number of bytes that can follow the TCP header. This option exists to allow adjustment to accommodate VLAN tagging or **Multiprotocol Label Switching** (**MPLS**).
- **Window Scale**: This option overcomes the inability of the **Window Size** field in a standard TCP header to specify a window size greater than 65,535 bytes. Window scaling allows you to specify a factor to multiply the advertised window size to achieve a larger window size. Both sides of a session must be able to support this option for it to apply; this is determined during the session setup.
- **TCP SACK Permitted Option**: This option indicates that this node supports selective acknowledgments, which allows a node to acknowledge ongoing and incoming data packets while still asking for a specific missing packet. The recovery process only requires retransmission of the missing packet(s), instead of the missing packet and all the packets that followed. Both sides of a session must be able to support this option for it to apply, as determined during session setup.

Wireshark TCP filters

Capture filter(s): `tcp tcp port 80`

Display filter(s): `tcp tcp.port == 80 tcp.dstport == 8080 tcp.stream == 2`

Application layer protocols

The most common application layer protocols include DHCP used to obtain client IP addresses and configuration information, DNS for hostname resolution, HTTP, SMB, POP/SMTP, and FTP for the most common network services and SIP, RTP, and RTCP for VoIP and video conferencing.

Extensive coverage of all the upper layer protocols is beyond the scope of this book. A brief overview of DHCP and DNS will be provided, as these protocols universally support network operations and HTTP as an example of one of the most common application layer protocols. The reader is encouraged to research any or all of these protocols further depending on their scope of interest and need to meet the analysis tasks being addressed.

Dynamic Host Configuration Protocol

Dynamic Host Configuration Protocol (**DHCP**) allows a client to lease an IP address from a pool managed by a DHCP server. The client can receive other configuration options such as the default gateway, subnet mask, and one or more DNS server addresses as well. DHCP is derived from an older BOOTP protocol; Wireshark uses bootp in display filter syntax. DHCP works by the client sending a broadcast packet using UDP source port `67` to UDP destination port `68`. A DHCP server will respond to the requestor's IP address and using UDP source port `68` to UDP destination port `67`.

DHCP servers don't necessarily have to reside on the same local network segment as clients. A relay agent such as a router can forward DHCP requests and respond to/from a different network where a DHCP server resides.

Wireshark DHCP filters

Capture filter(s): port `67` (DHCP is between ports `67` and `68`; filtering on port `67` is sufficient to get both sides of the conversations)

Display filter(s): `bootp bootp.option.value == 0` (DHCP Discover message)

Dynamic Host Configuration Protocol Version 6

Dynamic Host Configuration Protocol Version 6 (**DHCPv6**) is the IPv6 version of DHCP. Since IPv6 doesn't use broadcasts, DHCPv6 clients use the multicast address for `All_DHCP_Relay_Agents_and_Servers (ff02::1:2)` to locate DHCPv6 servers or relay agents.

Wireshark DHCPv6 filters

Capture filter(s): port `546` (DHCPv6 is between ports `546` and `547`; either will work) **Display filter(s)**: `dhcpv6 dhcpv6.msgtype == 1`(DHCPv6 Solicit message)

Domain Name Service

Domain Name Service (**DNS**) is used to convert host names, such as www.wireshark.org to IP addresses. DNS can also be used to identify the hostname associated with an IP address (an inverse or pointer (PTR) query) and several other network information services. This is a good protocol to become familiar with as it is used extensively to locate nodes both within an enterprise and on the Internet using hostnames.

Wireshark DNS filters

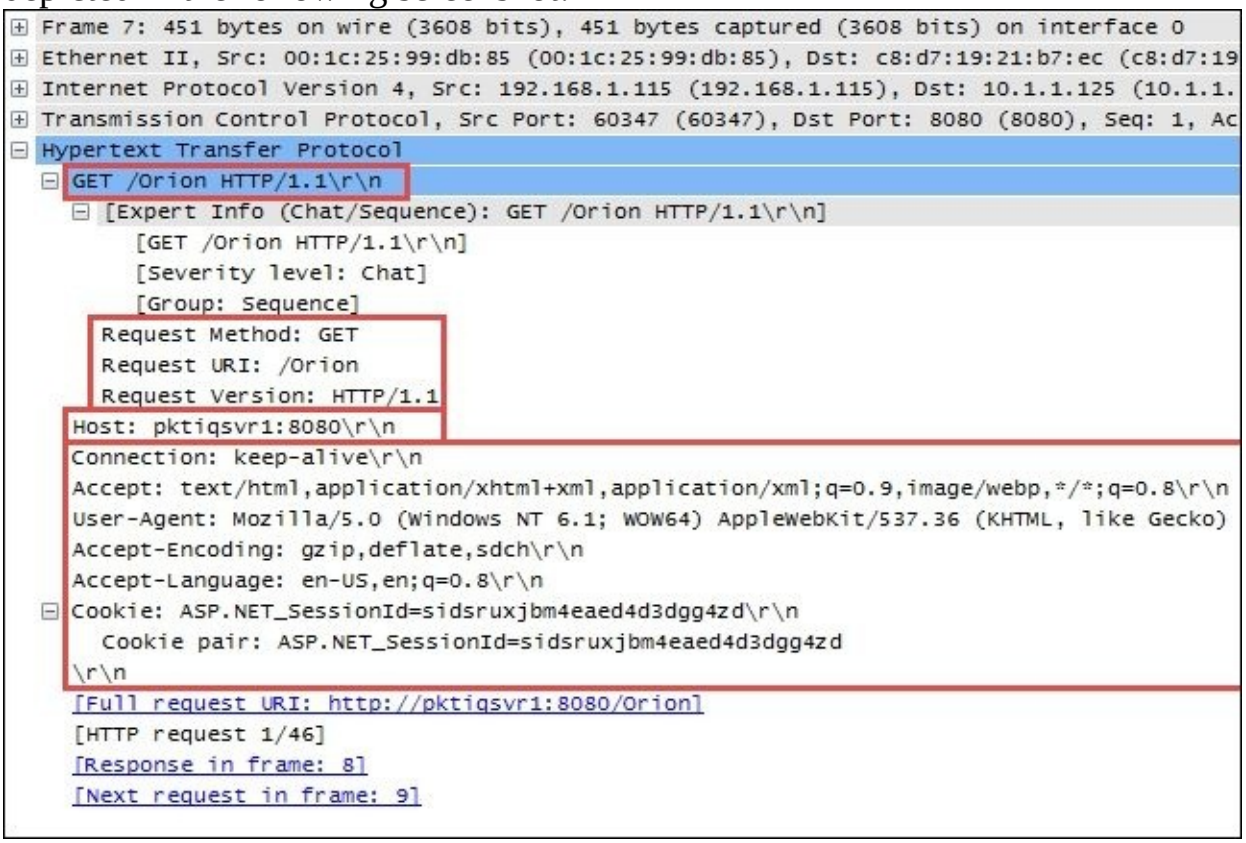
Capture filter(s): port `53`

Display filter(s): `dns` `dns.flags.response == 0`(DNS query) `dns.flags.response == 1`(DNS response) `dns.flags.rcode != 0`(DNS response contains an error)

Hypertext Transfer Protocol

Hypertext Transfer Protocol (**HTTP**) is the application protocol used when someone browses (unsecured) websites on the Internet, along with the secure version (HTTPS). HTTP/1.1 is the current version—although HTTP/2.0 is starting to appear in some environments. Be aware that some network devices such as proxy servers and gateways may not support HTTP/2.0 yet.

An example of a HTTP packet delivering a `GET` request to a web server is depicted in the following screenshot:



The most common features and fields of the HTTP protocol include HTTP Methods, Host, and Request Modifiers.

In the preceding screenshot, the HTTP header includes:

- **Request Method**: **GET**

- **Request URI**: **/Orion** (a home page on the web server)
- **Request Version**: **HTTP/1.1**

HTTP Methods

Some of the more common HTTP Methods are listed and described in the following table:

Method	Description
`GET`	This retrieves information defined by the **Uniform Resource Identifier** (**URI**) field
`HEAD`	This retrieves meta data related to the desired URI
`POST`	This sends data to the HTTP server/application
`OPTIONS`	This determines the options associated with a resource
`PUT`	This sends data to the HTTP server/application
`DELETE`	This deletes the resource defined by the URI
`CONNECT`	This is used to connect to a proxy device

Host

The **Host** field identifies the target host and port number of the resource being requested. In the preceding screenshot, **Host** is `pktiqsvr1` on port `8080`.

Request Modifiers

HTTP requests and responses use Request Modifiers to provide details for the request. In the preceding screenshot, Request Modifiers includes:

- **Connection**: This indicates the preference for a persistent connection (keep-alive).
- **Accept**: This is a list of data formats (`text`/`html` and `application`/`xhtml` plus `xml`) accepted.
- **User-agent**: This is a list of browser and operating system parameters

(Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit) for the requesting device.

- **Accept-encoding**: This is a list of the acceptable HTTP compression schemes (`gzip`, `deflate`, and `sdch`).
- **Accept-language**: The acceptable languages (en-US and en; q=0.8) where q=0.8 is a relative quality factor that specifies the language the user would prefer on a scale of 0 to 1.
- **Cookie**: This is a session ID cookie (`ASP.NET_SessionId=sidsruxjbm4eaed4d3dgg4zd`) that was previously stored on the user's browser in a cookie and is being provided to the website.

The following table lists some of the more commonly used modifiers:

Request Modifier	**Description**
Accept	Acceptable content types
Accept-charset	Acceptable character sets
Accept-encoding	Acceptable encodings
Accept-language	Acceptable languages
Accept-ranges	Server can accept range requests
Authorization	Authentication credentials for HTTP authentication
Cache-control	Caching directives
Connection	Type of connection preferred by the user agent
Cookie	HTTP cookie (a small piece of data sent from the website and stored in a user's browser,

	and/or sent back to the website the next time the user visits containing session information)
Content-length	Length of the request body in bytes
Content-type	Mime type of the body (used with `POST` and `PUT` requests)
Date	Date and time the message was sent
Expect	Defines server behavior expected by the client
If-match	Perform action if client-provided information matches
If-modified-since	Provide date/time of cached data; return **304 Not Modified** if the cached data is still current
If-range	Request for range of missing information
IF-unmodified-since	Only send if unmodified since the provided date/time
Max-forwards	Limit the number of forwards through proxies or gateways
Proxy-authorization	Authorization credential for a proxy connection
Range	Request only part of an entity
TE	Transfer encodings accepted
User-agent	A string containing browser and operating system information
Via	The proxies traversed

Wireshark HTTP filters

Capture filter(s): `tcp port http tcp port https`

Display filter(s): `http http.request.method == "GET" or http.request.method == "POST" http.response.code > 399` (identifies client or server error packets)

Additional information

Covering all the most common upper layer protocols or covering them to any great depth is obviously more than what can be included in a book of this size. I encourage you to spend some time studying those protocols that are of interest to you for personal or job-related reasons. The return on your investment in time will be well worth the effort.

Additional information for any of the protocols discussed in this chapter as well as all those not covered can be found online.

Wireshark wiki

If you are inspecting a protocol within the Wireshark's **Packet Details** pane, you can right-click on a protocol header or field within a header and select the **Wiki Protocol Page** from the menu to go to the specific page on the Wireshark wiki that contains information on that protocol. More information can be found at http://wiki.wireshark.org/ProtocolReference.

You can also get a complete list of Wireshark display filters on specific protocols by selecting a protocol header or a field within a header, right-clicking, and selecting **Filter Field Reference**.

Protocols on Wikipedia

You can find general information on various protocols on Wikipedia. Start with the Internet protocol. Additional links to the entire Internet protocol suite are also provided at http://en.wikipedia.org/wiki/Internet_Protocol.

Requests for Comments

The **Requests for Comment** (**RFC**) documents contain detailed information for all the Internet protocols. These documents are maintained by the **Internet Engineering Task Force** (**IETF**) and are the final word on how the protocols should be implemented and function (http://www.ietf.org/rfc.html). If you want to search for a specific RFC by title or keyword, use the link http://www.rfc-editor.org/search/rfc_search.php.

Summary

The topics covered in this chapter included protocol and field coverage of the network layer protocols IPv4, ARP, IGMP, ICMP, IPv6, and ICMPv6; the transport layer protocols UDP and TCP; an overview of the application layer protocols DHCP, DHCPv6, and DNS; and a more in-depth look at HTTP.

In the next chapter, we'll put all the topics covered so far to good use by using Wireshark to troubleshoot the functionality and performance issues.

Chapter 6. Troubleshooting and Performance Analysis

In this chapter, we will discuss the use of Wireshark for its primary purpose—troubleshooting network and application connectivity, functionality, and performance issues.

The topics that will be covered include:

- Troubleshooting methodology
- Troubleshooting connectivity issues
- Troubleshooting functional issues
- Performance analysis methodology
- Top five reasons for poor application performance
- Detecting and prioritizing delays
- Server processing time events
- Application turn's delay
- Network path latency
- Bandwidth congestion
- Data transport issues

These topics cover the majority of problems you'll come across in your analysis efforts.

Troubleshooting methodology

There are two fundamental reasons why you might be doing packet analysis:

- Troubleshooting a connectivity or functionality problem (a user can't connect, an application doesn't work, or doesn't work right), which we'll just call troubleshooting
- Analyzing a performance problem (the application works but is slow), which we'll call performance analysis

A third gray area is an application that basically works but is slow and occasionally times out, which could involve an underlying functional problem that causes the performance issue, or just simply be a really poor performance.

Troubleshooting a connectivity or functional issue is just a matter of comparing what normally works with what is going on, in the case you're working on.

A performance problem, on the other hand, requires determining where the majority of the time for a particular transaction to complete is being spent, measuring the delay and comparing that delay to what is normal or acceptable. The source and type of excessive delay usually points to the next area to investigate further or resolve.

In any case, you need to gather the information that allows you to determine whether this is a connectivity, functional, or performance issue and approach the problem according to its nature.

Gathering the right information

The most important thing you can do when approaching a problem is to determine what the real problem is so you can work on the right problem or the right aspect of the problem. In order to determine what the real problem is, or at least get close, you'll need to ask questions and interpret the answers. These questions could include the appropriate selections (depending on the complaint) from the following list:

- Define the problem:
  - What were you trying to do (connect to a server, log in, send/receive e-mails, general application usage, upload/download file, and specific transactions or functions)?
  - Is nothing working or is this just a problem with a specific application or multiple applications?
  - What website/server/application were you trying / connecting to? Do you know the hostname, URL, and/or IP address and port used to access the application?
  - What is the symptom/nature of the problem? Has this application or function/feature worked before, or is this the first time you've ever tried to use it?
  - Did you receive any error messages or other indications of a problem?
  - Is the issue consistent or intermittent? Depends? On what?
  - How long has this been happening?
  - Was there some recent change that did or could have had an impact?
  - What has been identified or suspected so far? What has been done to address this? Has it helped or changed anything?
  - Are there any other pertinent factors, symptoms, or recent changes to the user environment that should be considered?
- Determine the scope of the issue:
  - Is this problem occurring for a single user or a group of users?
  - Is this problem occurring within a specific office, region, or across the whole company?
  - Is this problem affecting different types of users differently?
- Collect system, application, and path information. For a more in-depth analysis (beyond single user or small group issues), the applicable questions from the following list might also need to be gathered and analyzed, as

appropriate to the complaint (some of this information may have to be obtained from network or application support groups):

- What is the browser type and version on the client (for web apps)? Is this different from clients that are working properly?
- What is the operating system type and version of the client(s) and server?
- What is the proper (vendor) application name and version? Are there any known issues with the application that match these symptoms (check the vendor's bug reports).
- What is the database type and server environment behind the application server?
- Are there other backend-supporting data sources such as an online data service or Documentum and SharePoint servers involved?
- What is the network path between the client and server? Are there firewalls, proxy servers, load balancers, and/or WAN accelerators in the path? Are they configured and working properly?
- Can you confirm the expected network path (and any WAN links involved) with a traceroute and verify the bandwidth availability?
- Can you measure the **round trip time** (**RTT**) path latency from the user to the application server with pings or TCP handshake completion times?

Establishing the general nature of the problem

At this point, you should be able to identify the general nature of the problem between one of the following three basic types:

- Determine whether this is a connectivity problem
  - User(s) cannot connect to anything
  - User(s) cannot connect to a specific server/application
- Determine whether this is a functionality or configuration problem
  - User(s) can connect (gets a login screen or other response from the application server) but cannot log in (or get the expected response)
  - User(s) can connect and log in but some or all functions are failing (for example, cannot send/receive e-mails)
- Determine whether this is a performance problem
  - User(s) can connect, log in, and use the application normally; but it's slow
  - The application works normally but sometimes it stalls and/or times out

Half-split troubleshooting and other logic

When I was doing component-level repair of electronic equipment early in my career, I learned to use the "half-split" troubleshooting method, which worked very well in almost every single case. Half-split troubleshooting is the process of cutting the problem domain (in my case, a piece of radio gear) in half by injecting or measuring signals roughly midway through the system. The idea is to see which half is working right and which half isn't, then shifting focus to the half that doesn't work, analyzing it halfway through, and so on. This process is repeated until you narrow the problem down to its source.

In the network and application world, the same half-split troubleshooting approach can be applied as well, in a general sense. If users are complaining that the network is slow, try to confirm or eliminate the network:

- Are users close to the server experiencing similar slowness? How about users in other remote locations?
- If a certain application is slow for a remote user, are other applications slow for that user as well?
- If users can't connect to a given server, can they connect to other servers nearby or at other locations?

By a process of logical examination of what does and doesn't work, you can eliminate a lot of guesswork and narrow your analysis down to just a few plausible possibilities.

It's usually much easier to determine the source of a connectivity or functionality problem if you have an environment where everything is working properly to compare with a situation that does not work. A packet capture of a working versus a non-working scenario can be compared to see what is different and if those differences are significant.

It is important not to make too many assumptions about a problem, even if the issue you're working on looks the same as the one that you've fixed before. Always verify the problem and the resolution that you should be able to apply and remove a fix and see the problem disappear/reappear reliably. Otherwise, you should question yourself about whether you've found the true source of the issue or are just affecting the symptoms.

Unless a reported problem is obviously a system-wide or specific server issue, it is better to conduct at least the initial analysis at or as close to the complaining user's workstation as possible. This has the advantages of offering the ability to perform the following actions:

- View and verify the actual problem that the user is reporting
- Measure round-trip times to the target server(s)
- Capture and view the TCP handshake process upon session initiation
- Capture and investigate the login and any other background processes and traffic
- Look for indications of network problems (lost packets and retransmissions) as they are experienced by the user's device
- Measure the apparent network throughput to the user's workstation during data downloads
- Eliminate the need to use a capture filter; the amount of traffic to/from a single workstation should not be excessive

A capture at a user workstation, server, or other device should be conducted with the use of an aggregating **Test Access Point** (**TAP**) versus using a switch SPAN port (as discussed in Chapter 3, *Capturing All the Right Packets*, or as a last resort by installing Wireshark on the user's workstation or server (if authorized).

Troubleshooting connectivity issues

Single user or small group connectivity issues can be resolved by confirming that the networking functions required for a user workstation to access local and remote network resources are functioning properly. The basic requirements or items to confirm include:

- Enabling the correct network interface(s) (workstation configuration)
- Confirming layer 1 (physical) connectivity
- Obtaining an IP address, subnet mask, and default gateway for each interface (DHCP)
- Obtaining the MAC address of the default gateway or other local network services (ARP)
- Obtaining the IP address of a network service (DNS)
- Connecting to a network service (TCP handshake or UDP response)

We'll briefly discuss each of these in order; while the first two steps will not involve using Wireshark, they are a necessary part in a troubleshooting approach. If the connectivity issue is affecting a group of users or a whole office, the first step is probably not applicable.

Enabling network interfaces

While it may seem obvious that network interfaces need to be enabled, the assumption that they are automatically enabled (especially for the wireless connectivity) by default upon device boot up may be false.

On Windows, you can use the command-line utility `ipconfig` to view the status and basic configuration (IP address, subnet mask, and default gateway) of network interfaces; on Linux or MAC devices, the equivalent command is `ifconfig` or `ip`.

Confirming physical connectivity

If a connectivity problem is isolated to a single user's workstation, the physical connections are suspected. There are a few items to check, and the troubleshooting steps that can be taken are as follows:

- If there is a problem with the Ethernet cable from the workstation to a wall jack, you need to swap the cable with a different one.
- If there is a problem with the cabling from the user's wall jack to the switch port, you need to temporarily plug the user's Ethernet cable into another (known good) wall jack.
- If there is a problem with the switch, switch port, or port configuration, you need to temporarily plug the user's port cable into another (known good) port. Be aware that some network security policies call to disable switch ports until they are needed or configuring the port to be associated with a single, specific MAC address. If so, a port may not work when you plug into it although there is nothing physically wrong with it.

Obtaining the workstation IP configuration

Unless the workstation was manually configured, it will need to get its IP address, subnet mask, default gateway, and DNS server settings from a DHCP server. If this does not appear to be working properly (after checking the configuration using `ipconfig` (Windows) or `ifconfig`, (Linux or Mac OS X)), you need to perform a packet capture during the workstation initialization/boot-up process using a TAP or SPAN port and investigate the DHCP requests and responses.

There are eight DHCP message types (not to be confused with the two Bootstrap Protocol types, Boot Request and Boot Reply):

Message type number	**Message type**	**Description**
1	DHCP Discover	A client broadcast to locate an available DHCP server
2	DHCP Reply	A server to client response to a DHCP Discover to offer configuration parameters
3	DHCP Request	A client message to a DHCP server to either one of the following conditions: • Request offered parameters from one server and decline offers from other DHCP servers • Confirm correctness of previously allocated address after a reboot • Extending the lease on an IP address
4	DHCP Decline	Client message to DHCP server indicating the offered address is not acceptable
5	DHCP Acknowledgment	Server to client with configuration parameters including a committed network address
6	DHCP Negative Acknowledgement	Server to client indicating client's address is incorrect or expired
7	DHCP Release	Client to server releasing a network address and canceling a

		lease
8	DHCP Informational	Client to server asking for local configuration parameters only

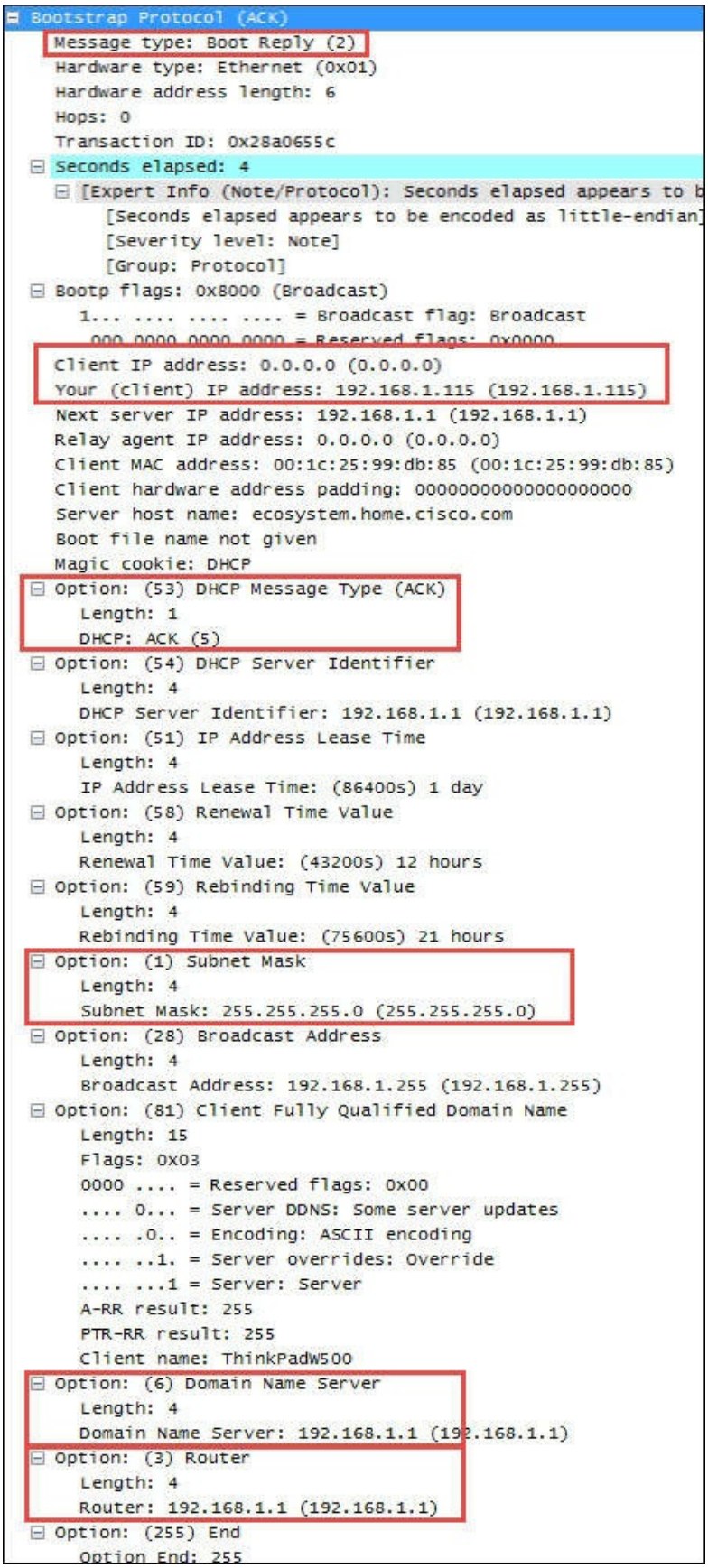
For a workstation that is booting up and was previously working on the network, you'll generally see the DHCP Request and Acknowledgment packets verifying that the workstation can still use a previously leased address. On an entirely cold start up, the first two DHCP packets will be DHCP Discover and DHCP Offer packets, followed by the Request and ACK packets.

In a DHCPv6 environment, the typical packet sequence is DHCPv6 Solicit, DHCPv6 Advertise, DHCPv6 Request, and DHCPv6 Reply.

The fields to verify in a DHCP Response packet (or similar fields in a DHCPv6 Advertise packet) include the following four fields:

- **Your (client) IP Address**: This is the offered IP address for this workstation
- **Subnet Mask**: This is the subnet mask to use on this network
- **Domain Name Server**: This is the DNS server IP address
- **Router**: This is the IP address of the default gateway to use

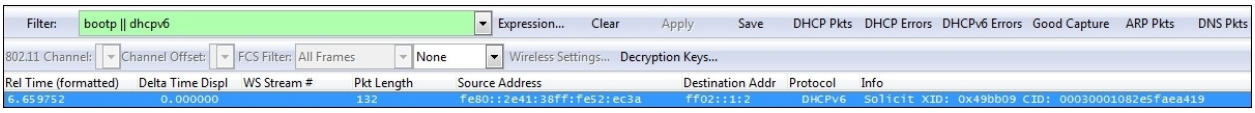
This is minimum data required for any network communications; an example of these fields being provided in a DHCP Reply packet is illustrated in the following screenshot:


Bootstrap Protocol (ACK)
Message type: Boot Reply (2)
Hardware type: Ethernet (0x01)
Hardware address length: 6
Hops: 0
Transaction ID: 0x28a0655c
Seconds elapsed: 4
[Expert Info (Note/Protocol): Seconds elapsed appears to b
[Seconds elapsed appears to be encoded as little-endian]
[Severity level: Note]
[Group: Protocol]
Bootp flags: 0x8000 (Broadcast)
1... .... = Broadcast flag: Broadcast
.000 0000 0000 0000 = Reserved flags: 0x0000
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 192.168.1.115 (192.168.1.115)
Next server IP address: 192.168.1.1 (192.168.1.1)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: 00:1c:25:99:db:85 (00:1c:25:99:db:85)
Client hardware address padding: 00000000000000000000
Server host name: ecosystem.home.cisco.com
Boot file name not given
Magic cookie: DHCP
Option: (53) DHCP Message Type (ACK)
Length: 1
DHCP: ACK (5)
Option: (54) DHCP Server Identifier
Length: 4
DHCP Server Identifier: 192.168.1.1 (192.168.1.1)
Option: (51) IP Address Lease Time
Length: 4
IP Address Lease Time: (86400s) 1 day
Option: (58) Renewal Time Value
Length: 4
Renewal Time Value: (43200s) 12 hours
Option: (59) Rebinding Time Value
Length: 4
Rebinding Time Value: (75600s) 21 hours
Option: (1) Subnet Mask
Length: 4
Subnet Mask: 255.255.255.0 (255.255.255.0)
Option: (28) Broadcast Address
Length: 4
Broadcast Address: 192.168.1.255 (192.168.1.255)
Option: (81) Client Fully Qualified Domain Name
Length: 15
Flags: 0x03
0000 .... = Reserved flags: 0x00
.... 0... = Server DDNS: Some server updates
.... .0.. = Encoding: ASCII encoding
.... ..1. = Server overrides: Override
....1 = Server: Server
A-RR result: 255
PTR-RR result: 255
Client name: ThinkPadW500
Option: (6) Domain Name Server
Length: 4
Domain Name Server: 192.168.1.1 (192.168.1.1)
Option: (3) Router
Length: 4
Router: 192.168.1.1 (192.168.1.1)
Option: (255) End
Option End: 255

You can apply Wireshark display filters to isolate DHCP packets; the filter is `bootp`, as this is the legacy name for DHCP:

- **DHCP display filter**: `bootp bootp.option.dhcp == 5 (DHCP Message Type 'ACK')`
- **DHCPv6 display filter**: `dhcpv6 dhcpv6.msgtype == 2 (DHCPv6 'Advertise')`

You can save the basic `bootp` and `dhcpv6` display filters as a **Filter Expression Button** (**FEB**) after entering the filter string in the textbox on the **Display Filter** toolbar, clicking on **Save**, and giving the button a name such as `DHCP Pkts` and `DHCPv6 Pkts` respectively. Alternatively, you could combine both filters with an `or` (`||`) in one button, as shown in the following screenshot:



You might want to save another FEB that displays an abnormal DHCP condition packets using the following display filter string and call the **DHCP Errors** button or a similar as follows:

```
bootp.option.dhcp == 4 || bootp.option.dhcp == 6 ||
bootp.option.dhcp == 7
```

Similar abnormal event display filters for DHCPv6 could include:

```
dhcpv6.msgtype == 8 || dhcpv6.msgtype == 9 || dhcpv6.msgtype == 10
```

You can research more about DHCP, DHCPv6, and the various DHCPv6 message types online or from other sources if you need to analyze these in more detail.

Obtaining MAC addresses

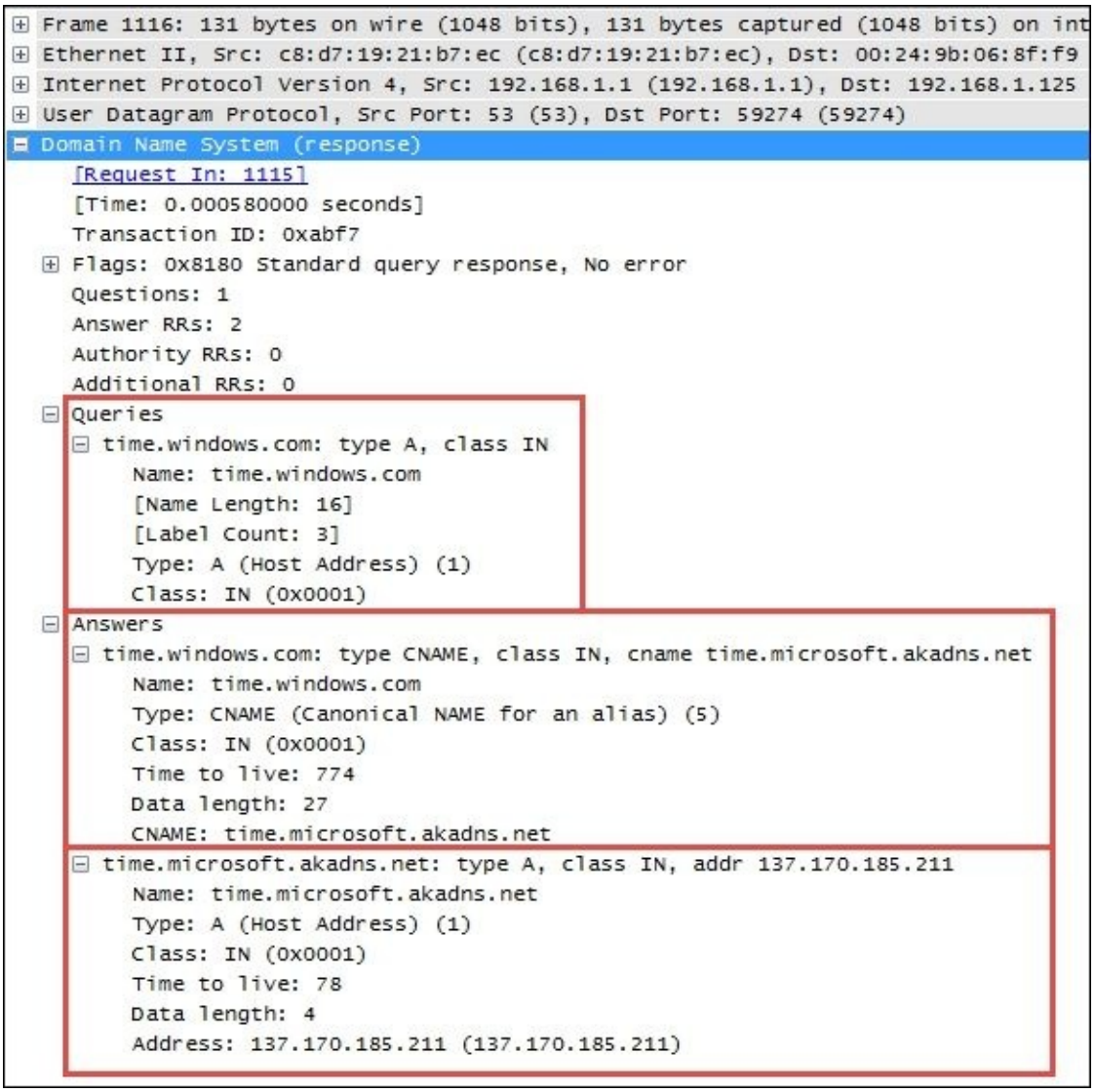
A workstation will utilize the ARP protocol to obtain a MAC address for known IP addresses of network services, such as its default gateway or the DNS server if it's located on the same network segment. The ARP protocol and how it typically functions has already been covered in Chapter 2, *Networking for Packet Analysts*.

You may want to create an ARP FEB using the `arp` display filter syntax to make it quick and easy to inspect those packets.

Obtaining network service IP addresses

A client workstation sends queries to a DNS server to obtain an IP address for a given hostname; the DNS server responds with the information or asks other DNS servers for the information on behalf of the client.

The format of the DNS query and response packet fields as displayed in the Wireshark **Packet Details** pane is fairly intuitive. An example of a DNS response packet containing a resolved IP address for `time.windows.com`, which actually provided the IP address (`137.170.185.211`) for the alias `time.microsoft.akadns.com` is shown in the following screenshot:



If a client workstation cannot obtain the IP address of a web service or application server, a packet-level investigation of the request (which URL or hostname is being requested), and what the response is from the DNS server (if any) should be revealing. A comparison of a failing query with queries that work properly for other hostnames or from other workstations should reveal the root of the problem (if DNS is the problem). Failure to obtain an IP address can be caused by an inoperable DNS server, improper hostname or URL, or a problem with connectivity from the user to other parts of the network, which we'll check next.

Basic network connectivity

A few simple tests can confirm that basic network connectivity is working, or reveal a routing issue or another issue that needs to be addressed by the network support team.

Capturing and analyzing the ICMP packets sent and received during the following tests can be revealing; although, the test results themselves are often telling enough:

- Ping the user's default gateway using the default gateway IP address obtained from using `ipconfig /all` (Windows) or `ip addr show` (Linux) to confirm that the user workstation has basic connectivity on the local network.
- Ping the hostname or URL of the target server. If this fails (request timed out message), try to ping other hosts or URLs. If necessary, inspect the DNS and/or ICMP responses in a packet capture of these tests to determine the nature of the failure. Otherwise, take note of the average round trip times.
- If a ping works to the default gateway but pinging other targets fails, a traceroute to a target server can reveal where in the network path connectivity ceases to function or is blocked.

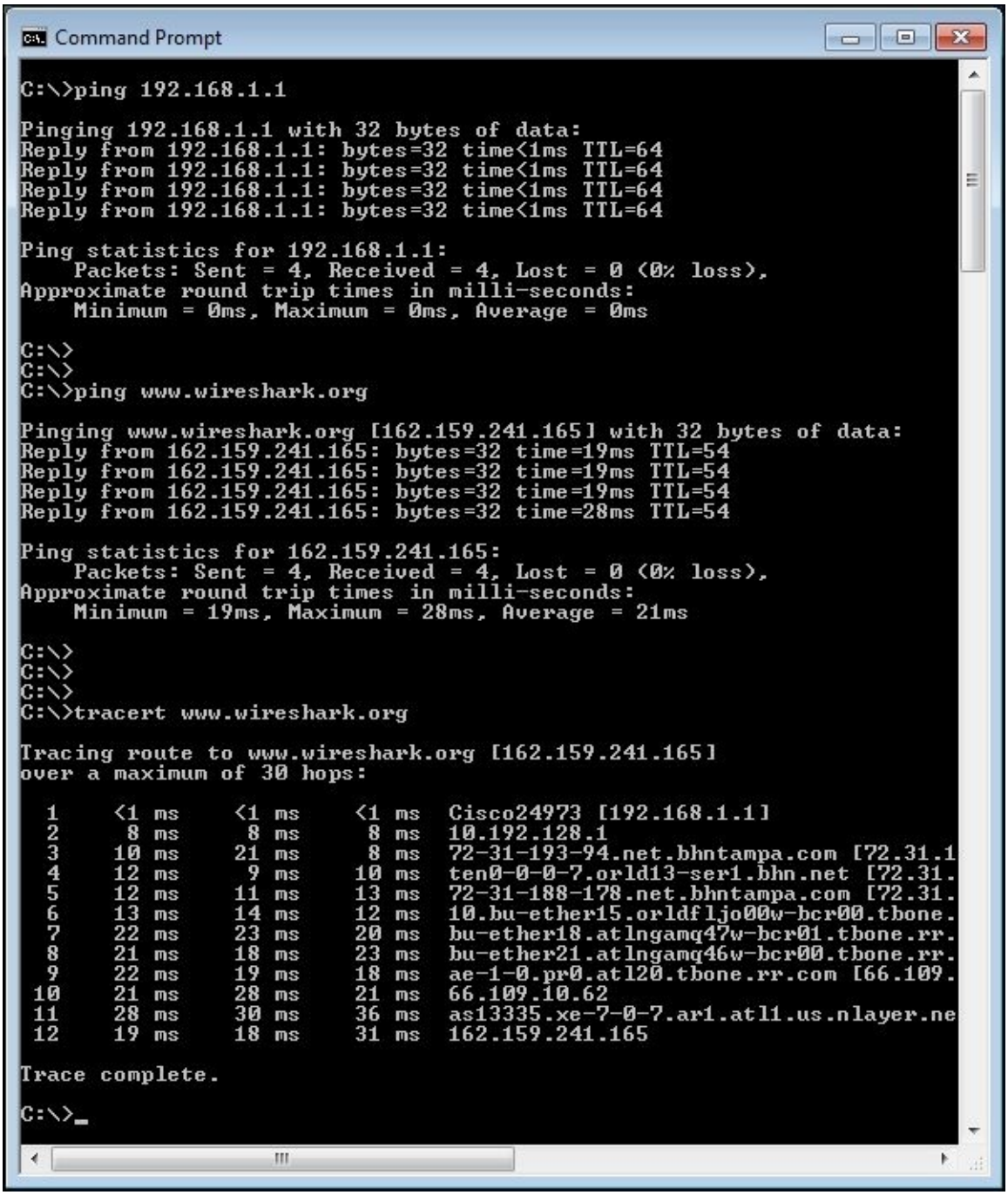
  ### Note

  The traceroute command-line utility in Windows is `tracert`, whereas for traceroutes on Linux/Unix and Mac OS X machines, the command is `traceroute`. To do a traceroute in Windows, open a **Command Prompt** (**CMD**) window and type `tracert <hostname or IP Address of target>`. In most other environments, open a terminal window and type `traceroute <hostname or IP address of target>`.

If you can ping the target server and network connectivity is functioning, you can move on to the next step in the troubleshooting process. If not, be aware that some hosts may be configured to not respond to ICMP ping requests, and/or ICMP is blocked by a firewall between the user and server for security reasons. So, the inability to ping a device is not necessarily a sign of a network problem. Traceroute results should help determine how far and to what extent network

connectivity is functioning in the path towards the target server; testing to other targets should be revealing as well.

An example of pinging a default gateway, then a URL, and finally performing a traceroute to the target URL is depicted in the following screenshot:



Connecting to the application services

If network connectivity from a user workstation to a target server is functional (as proven by the ability to ping the host), a problem connecting to a specific application hosted on that server may be caused by a number of factors:

- The URL or port used by the client to access the application is wrong
- The port used to access the application is blocked by a firewall
- The application service is not turned up or is not working properly

The first of these factors is far more likely for a single user issue. Any of the last two factors would prevent anyone in a group or the whole organization from accessing the application. A packet-level analysis (from the client side) of a user attempting to connect to an application that is blocked should result in ICMP messages: **Destination Host is Unreachable** or **Destination Port is Unreachable**, or there will be no response at all if ICMP messages are being blocked by a firewall.

If the server is up, the application is reportedly operational but cannot be accessed; a client-side capture does not offer any solid clues, but a packet capture of the TCP session setup (if any) from or near the server end should be revealing.

Troubleshooting functional issues

If a user is able to connect and set up a TCP session with an application server, but the application does not function otherwise, or function correctly, then, there are a number of areas that can be investigated. These areas can be investigated using a combination of packet-level analysis, error reports, and configuration comparisons with captures and configurations from other users' machines:

- **User credentials**: The most common reason for specific-user issues with application functionality is the lack of proper credentials, authorization, rights, and so on. This is the first thing to check whether other users are working normally.
- **Application settings on the user machine**: Some applications require specific configuration files to be placed on a user's machine in a specific location. Applications may also require certain version levels of application-specific utilities, Java, .NET frameworks, and so on. Usually, an application will provide an error message indicating at least the general nature of a configuration problem.
- **Application reported errors**: You can look for the error code within response packets or on the user screen that may reveal the nature of application errors:
    - Status code greater than 400 in HTTP, FTP, or SIP response packets
    - Error code in SMB response packets
    - Other application-specific exceptions, error codes, and messages
- **Differences in web browsers**: Some web applications are designed to work with specific browsers (Chrome, Internet Explorer, Firefox, Opera, and so on) and may not work properly or at all on other browsers and there may not be any error messages provided that indicate this is the case. A comparison of the browser type and version with other working users may be revealing.

The causes of network connectivity and application functionality issues can vary widely, so it is impossible to draw a clear roadmap for every possibility. The best approach to successfully address these problems is not to make too many assumptions without proving those assumptions correct with systematic, logical troubleshooting steps, but try to find or create a scenario where the system, or at least part of the system, works properly and compare the appropriate packet-level details of the working environment to the one that doesn't work.

Performance analysis methodology

Analyzing an application's performance problem is basically a case of identifying where the majority of the time for a particular task to complete is being spent, and measuring/comparing that time to what is normal and/or acceptable for that type of task.

Top five reasons for poor application performance

Generally speaking, performance issues can be attributed to one of the following five areas, in order of decreasing likelihood:

- Server processing time delay
- Application turns delay
- Network path latency
- Bandwidth congestion
- Data transport (TCP) issues

Client processing time is usually a relatively small component of overall response time—except perhaps for some compute-extensive desktop applications, which leaves the focus on the network and server environments and any performance-affecting application design characteristics.

Preparing the tools and approach

As was done when preparing to troubleshoot a connectivity or functionality problem, you'll need to gather the right information about the application environment and problem domain. You'll also want to determine which tools you may need to use during the analysis: Wireshark, TAPs to facilitate packet captures, and any other analysis tools.

You will also need to determine where to perform the first packet capture:

- A client-side capture is the best place to begin a performance analysis effort. From this vantage point, you can view and verify what the user is complaining about, view any error messages presented to the user or evident in the packet capture, measure network round-trip times, and capture the performance characteristics to study within a packet capture without the need to use a capture filter so you know you won't miss anything.
- A server-side capture may be needed because a client-side capture may not be possible for a user that is at a long distance, or to analyze server-to-server transactions to backend databases or other data sources.
- A packet capture at some intermediate point in the network path may be

needed to isolate the source of excessive packet loss/errors and the associated retransmissions.

Remember that the use of an aggregating TAP is preferable over using SPAN ports, or you can install Wireshark on the client workstation or server as a last resort, but get the capture done any way you have to.

Performing, verifying, and saving a good packet capture

After performing the capture and saving the bulk capture file, confirm the following:

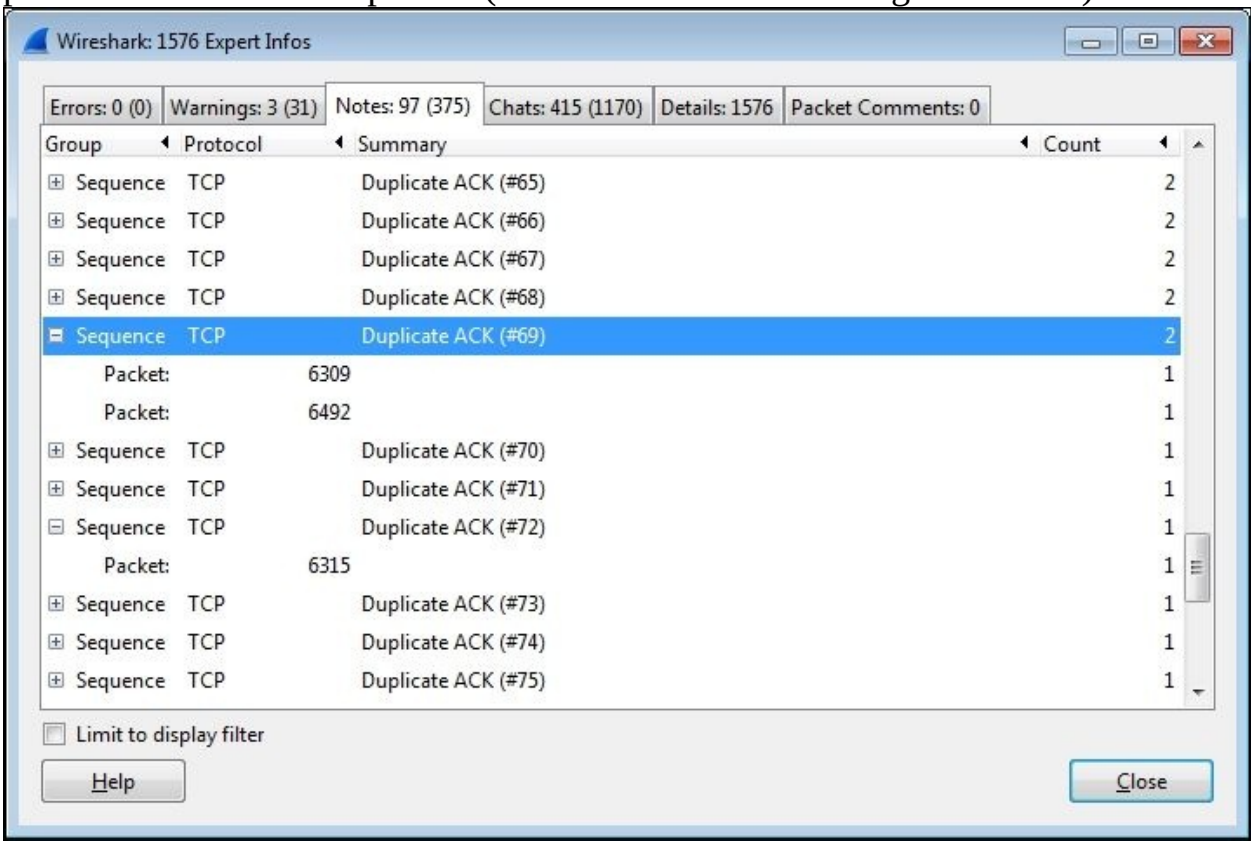
1. Check the file to ensure there are no packets with the **ACKed Unseen Segment** messages in the Wireshark **Warnings** tab in the **Expert Info** menu, which means Wireshark saw a packet that was acknowledged but didn't see the original packet; an indication that Wireshark is missing packets due to a bad TAP or SPAN port configuration or excessive traffic levels. In any case, if more than just a few of these show up, you'll want to do the capture again after confirming the capture setup.
2. Next, you'll want to review the captured conversations in **IPv4** in the **Conversations** window and sort the **Bytes** column. The IP conversation between the user and application server should be at or near the top so you can select this conversation, right-click on it, and select **A <-> B** in the **Selected** menu.
3. After reviewing the filtered data to ensure it contains what you expected, select **Export Specified Packets** from the **File** menu and save the filtered capture file with a filename that reflects the fact that this is a filtered subset of the bulk capture file.
4. Finally, open the filtered file you just saved so you're working with a smaller, faster file without any distracting packets from other conversations that have nothing to do with your analysis.

Initial error analysis

At the onset of your analysis, you should take a look through the **Errors**, **Warnings**, and **Notes** tabs of Wireshark's **Expert Info** window (**Analyze** | **Expert Info**) for significant errors such as excessive retransmissions, Zero Window conditions, or application errors. These are very helpful to provide clues to the source of reported poor performance.

Although a few lost packets and retransmissions are normal and of minimal consequence in most packet captures, an excessive number indicates that network congestion is occurring somewhere in the path between user and server, packets are being discarded, and that an appreciable amount of time may be lost recovering from these lost packets.

Seeing a high count number of Duplicate ACK packets in the **Expert Info Notes** window may be alarming, but can be misleading. In the following screenshot, there was up to 69 Duplicate ACKs for one lost packet, and for a second lost packet the count went up to 89 (not shown in the following screenshot):



However, upon marking the time when the first Duplicate ACK occurred in Wireshark using the **Set/Unset Time Reference** feature in the **Edit** menu and then going to the last Duplicate ACK in this series by clicking the packet number in the **Expert Info** screen and inspecting a **Relative time** column in the **Packet List** pane, only 30 milliseconds had transpired. This is not a significant amount of time, especially if **Selective Acknowledgment** is enabled (as it was in this

example) and other packets are being delivered and acknowledged in the meantime. Over longer latency network paths, the Duplicate ACK count can go much higher; it's only when the total number of lost packets and required retransmissions gets excessively high that the delay may become noticeable to a user.

Another condition to look for in the **Expert Info Notes** window includes the **TCP Zero Window** reports, which are caused by a receive buffer on the client or server being too full to accept any more data until the application has time to retrieve and process the data and make more room in the buffer. This isn't necessarily an error condition, but it can lead to substantial delays in transferring data, depending on how long it takes the buffer to get relieved.

You can measure this time by marking the TCP Zero Window packet with a time reference and looking at the elapsed relative time until a **TCP Window Update** packet is sent, which indicates the receiver is ready for more data. If this occurs frequently, or the delay between Zero Window and Window Update packets is long, you may need to inspect the host that is experiencing the full buffer condition to see whether there are any background processes that are adversely affecting the application that you're analyzing.

Note

If you haven't added them already, you need to add the **Relative time** and **Delta time** columns in the **Packet List** pane. Navigate to **Edit** | **Preferences** | **Columns** to add these. Adding time columns was also explained in Chapter 4, *Configuring Wireshark*.

You will probably see the connection reset (RST) messages in the **Warnings** tab. These are not indicators of an error condition if they occur at the end of a client-server exchange or session; they are normal indicators of sessions being terminated.

A very handy **Filter Expression** button you may want to add to Wireshark is a **TCP Issues** button using this display filter string as follows:

```
tcp.analysis.flags && !tcp.analysis.window_update &&
!tcp.analysis.keep_alive && !tcp.analysis.keep_alive_ack
```

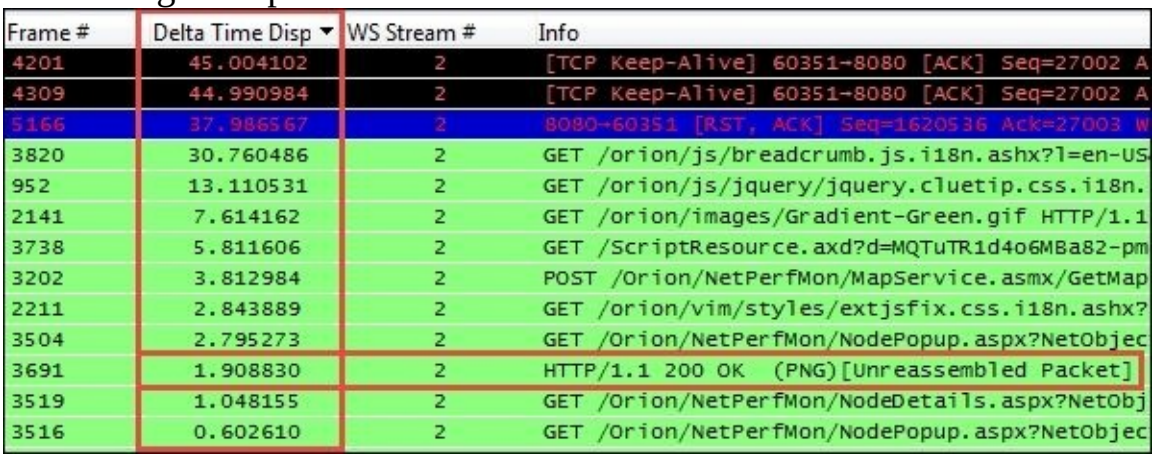
This will filter and display most of the packets for which you will see the messages in the **Expert Info** window and provide a quick overview of any significant issues.

Detecting and prioritizing delays

Since we're addressing application performance, the first step is to identify any delays in the packet flow so we can focus on the surrounding packets to identify the source and nature of the delay.

One of the quickest ways to identify delay events is to sort a **TCP Delta time** column (by clicking on the column header) so that the highest delay packets are arranged at the top of the packet list. You can then inspect the **Info** field of these packets to determine which, if any, reflect a valid performance affecting the event as most of them do not.

In the following screenshot, a **TCP Delta time** column is sorted in order of descending inter-packet times:



Frame #	Delta Time Disp ▾	WS Stream #	Info
4201	45.004102	2	[TCP Keep-Alive] 60351→8080 [ACK] Seq=27002 A
4309	44.990984	2	[TCP Keep-Alive] 60351→8080 [ACK] Seq=27002 A
5166	37.986567	2	8080→60351 [RST, ACK] Seq=1620536 Ack=27003 W
3820	30.760486	2	GET /orion/js/breadcrumb.js.i18n.ashx?l=en-US
952	13.110531	2	GET /orion/js/jquery/jquery.cluetip.css.i18n.
2141	7.614162	2	GET /orion/images/Gradient-Green.gif HTTP/1.1
3738	5.811606	2	GET /ScriptResource.axd?d=MQTuTR1d4o6MBa82-pm
3202	3.812984	2	POST /Orion/NetPerfMon/MapService.asmx/GetMap
2211	2.843889	2	GET /orion/vim/styles/extjsfix.css.i18n.ashx?
3504	2.795273	2	GET /Orion/NetPerfMon/NodePopup.aspx?NetObjec
3691	1.908830	2	HTTP/1.1 200 OK (PNG)[Unreassembled Packet]
3519	1.048155	2	GET /Orion/NetPerfMon/NodeDetails.aspx?NetObj
3516	0.602610	2	GET /Orion/NetPerfMon/NodePopup.aspx?NetObjec

Let's have a detailed look at all the packets:

- The first two packets are the **TCP Keep-Alive** packets, which do just what they're called. They are a way for the client (or server) to make sure a connection is still alive (and not broken because the other end has gone away) after some time has elapsed with no activity. You can disregard these; they usually have nothing to do with the user experience.
- The third packet is a Reset packet, which is the last packet in the

conversation stream and was sent to terminate the connection. Again, it has no impact on the user experience so you can ignore this.

- The next series of packets listed with a high inter-packet delay were **GETs** and a **POST**. These are the start of a new request and have occurred because the user clicked on a button or some other action on the application. However, the time that expired before these packets appear were consumed by the user think time—a period when the user was reading the last page and deciding what to do next. These also did not affect the user's response time experience and can be disregarded.
- Finally, **Frame # 3691**, which is a **HTTP/1.1 200 OK**, is a response from the server to a previous request; this is a legitimate response time of 1.9 seconds during which the user was waiting. If this response time had consumed more than a few seconds, the user may have grown frustrated with the wait and the type of request and reason for the excessive delay would warrant further analysis to determine why it took so long.

The point of this discussion is to illustrate that not all delays you may see in a packet trace affect the end user experience; you have to locate and focus on just those that do.

You may want to add some extra columns to Wireshark to speed up the analysis process; you can right-click on a column header and select **Hide Column** or **Displayed Columns** to show or hide specific columns:

- **TCP Delta** (**tcp.time_delta**): This is the time from one packet in a TCP conversation to the next packet in the same conversation/stream
- **DNS Delta** (**dns.time**): This is the time between DNS requests and responses
- **HTTP Delta** (**http.time**): This is the time between the HTTP requests and responses

  ## Note

  You should ensure that **Calculate conversation timestamps** is enabled in the **TCP** option, which can be found by navigating to **Edit** | **References** | **Protocols**, so that the delta time columns will work properly.

While you're adding columns, the following can also be helpful during a performance analysis:

- **Stream #** (**tcp.stream**): This is the TCP conversation stream number. You can right-click on a stream number in this column, and select **Selected** from the **Apply as a filter** menu to quickly build a display filter to inspect a single conversation.
- **Calc Win Size** (**tcp.window_size**): This is the calculated TCP window size. This column can be used to quickly spot periods within a data delivery flow when the buffer size is decreasing to the point where a Zero Window condition occurred or almost occurred.

Server processing time events

One of the most common causes of poor response times are excessively long server processing time events, which can be caused by processing times on the application server itself and/or delays incurred from long response times from a high number of requests to backend databases or other data sources.

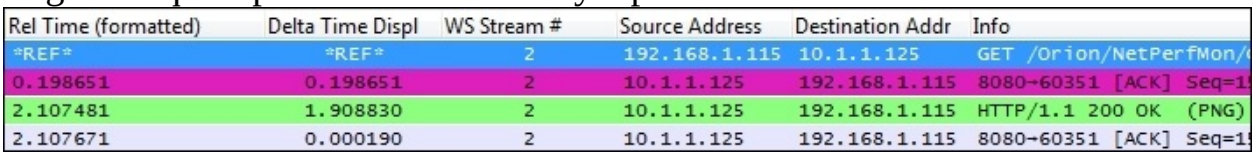
Confirming and measuring these response times is easy within Wireshark using the following approach:

1. Having used the sorted **Delta Time** column approach discussed in the previous section to identify a legitimate response time event, click on the suspect packet and then click on the **Delta Time** column header until it is no longer in the sort mode. This should result in the selected packet being highlighted in the middle of the **Packet List** pane and the displayed packets are back in their original order.
2. Inspect the previous several packets to find the request that resulted in the long response time. The pattern that you'll see time and again is:
    1. The user sends a request to the server.
    2. The server fairly quickly acknowledges the request (with a **[ACK]** packet).
    3. After some time, the server starts sending data packets to service the request; the first of these packets is the packet you saw and selected in the sorted **Delta Time** view.

The time that expires between the first user request packet and the third packet when the server actually starts sending data is the **First Byte** response time. This is the area where you'll see longer response times caused by server processing time. This effect can be seen between users and servers, as well as between

application servers and database servers or other data sources.

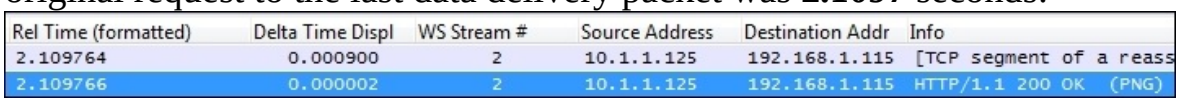
In the following screenshot, you can see a **GET** request from the client followed by an ACK packet from the server 198 milliseconds later (**0.198651** seconds in the **Delta Time Displ** column); **1.9** seconds after that the server sends the first data packet (**HTTP/1.1 200 OK** in the **Info** field) followed by the start of a series of additional packets to deliver all of the requested data. In this illustration, a **Time Reference** has been set on the request packet. Looking at the **Rel Time** column, it can be seen that **2.107481** seconds transpired between the original request packet and the first byte packet:



Rel Time (formatted)	Delta Time Displ	WS Stream #	Source Address	Destination Addr	Info
REF	*REF*	2	192.168.1.115	10.1.1.125	GET /Orion/NetPerfMon/
0.198651	0.198651	2	10.1.1.125	192.168.1.115	8080→60351 [ACK] Seq=1
2.107481	1.908830	2	10.1.1.125	192.168.1.115	HTTP/1.1 200 OK (PNG)
2.107671	0.000190	2	10.1.1.125	192.168.1.115	8080→60351 [ACK] Seq=1

It should be noted that how the First Byte data packet is summarized in the **Info** field depends upon the state of the **Allow subdissector to reassemble TCP streams** setting in the **TCP** menu, which can be found by navigating to **Edit** | **Preferences** | **Protocols**, as follows:

- If this option is disabled, the First Byte packet will display a summary of the contents of the first data packet in the **Info** field, such as **HTTP/1.1 200 OK** shown in the preceding screenshot, followed by a series of data delivery packets. The end of this delivery process has no remarkable signature; the packet flow just stops until the next request is received.
- If the **Allow subdissector to reassemble TCP streams** option is enabled, the First Byte packet will be summarized as simply a **TCP segment of a reassembled PDU** or similar notation. The **HTTP/1.1 200 OK** summary will be displayed in the **Info** field of the last data packet in this delivery process, signifying that the requested data has been delivered. An example of having this option enabled is illustrated in the following screenshot. This is the same request/response stream as shown in the preceding screenshot. It can be seen in the **Rel Time** column that the total elapsed time from the original request to the last data delivery packet was **2.1097** seconds:



Rel Time (formatted)	Delta Time Displ	WS Stream #	Source Address	Destination Addr	Info
2.109764	0.000900	2	10.1.1.125	192.168.1.115	[TCP segment of a reass
2.109766	0.000002	2	10.1.1.125	192.168.1.115	HTTP/1.1 200 OK (PNG)

Note

The **Reassemble SMB Transaction payload** setting in the SMB protocol preferences will affect how SMB and SMB2 responses are summarized in the **Info** field in like fashion to the related setting in the TCP protocol preferences.

In either case, the total response time as experienced by the user will be the time that transpires from the client request packet to the end of the data delivery packet plus the (usually) small amount of time required for the client application to process the received data and display the results on the user's screen.

In summary, measuring the time from the first request to the First Byte packets is the server response time. The time from the first request packet to the final data delivery packet is a good representation of the user response time experience.

Application turn's delay

The next, most likely source of poor response times—especially for remote users accessing applications over longer distances—is a relatively high number of what is known as application turns. An app turn is an instance where a client application makes a request and nothing else can or does happen until the response is received, after which another request/response cycle can occur, and so on.

Every client/server application is subject to the application turn effects and every request/response cycle incurs one. An application that imposes a high number of app turns to complete a task—due to poor application design, usually—can subject an end user to poor response times over higher latency network paths as the time spent waiting for these multiple requests and responses to traverse back and forth across the network adds up, which it can do quickly.

For example, if an application requires 100 application turns to complete a task and the **round trip time** (**RTT**) between the user and the application is 50 milliseconds (a typical cross-country value), the app turns delay will be 5 seconds:

```
100 App Turns X 50 ms RTT network latency = 5 seconds
```

This app turns' effect is additional wait (response) time on top of any server processing and network transport delays that is 5 seconds of totally wasted time. The resultant longer time inevitably gets blamed on the network; the network support teams assert that the network is working just fine and the application team points out that the application works fine until the network gets involved. And on it goes, so it is important to know about the app turns effects, what causes them, and how to measure and account for them.

Web applications can incur a relatively high app turn count due to the need to download one or more CSS files, JavaScript files, and multiple images to populate a page. Web designers can use techniques to reduce the app turn and download times, and modern browsers allow numerous connections to be used at the same time so that multiple requests can be serviced simultaneously, but the effects can still be significant over longer network paths. Many older, legacy applications and Microsoft's **Server Message Block** (**SMB**) protocols are also known to impose a high app turn count.

The presence and effects of application turns are not intuitively apparent in a packet capture unless you know they exist and how to identify and count them. You can do this in Wireshark for a client-side capture using a display filter:

```
ip.scr == 10.1.1.125 && tcp.analysis.ack_rtt > .008 &&
tcp.flags.ack == 1
```

You will need to replace the `ip.src` IP address with that of your server, and adjust the `tcp.analysis.ack_rtt` value to the RTT of the network path between the user and server. Upon applying the filter, you will see a display of packets that represent an application turn, and you can see the total app turns count in the **Displayed** field in the center section of the Wireshark's **Status Bar** option at the bottom of the user interface.

If you measure the total time required to complete a task (first request packet to last data delivery packet) and divide that time into the time incurred for application turns (number of app turns X network RTT), you can derive an approximate app turn time percentage:

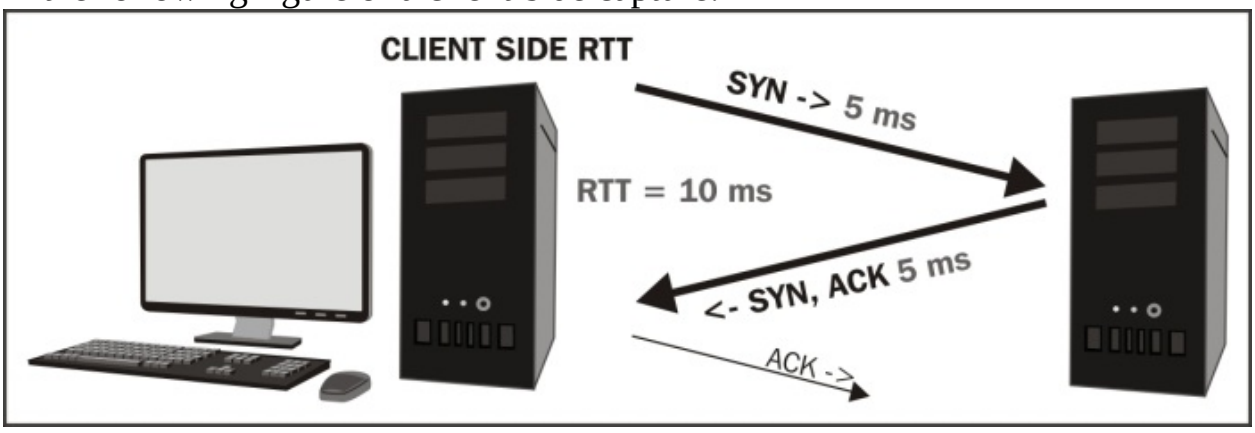
`5 seconds app turns delay / 7.5 seconds total response time = 66% of RT`

Any percentage over 25 percent warrants further investigation into what can be done to reduce either the RTT latency (server placement) or the number app turns (application design).

Network path latency

The next leading cause of high response times is network path latency, which compounds the effects of application turns as discussed in the preceding section, as well as affecting data transport throughput and how long it takes to recover from packet loss and the subsequent retransmissions.

You can measure the network path latency between a client and server using the ICMP ping packets, but you can also determine this delay from a packet capture by measuring the time that transpires from a client SYN packet to the server's SYN, ACK response during a TCP three-way handshake process, as illustrated in the following figure of a client-side capture:



In a server-side capture, the time from the SYN, ACK to the client's ACK (third packet in the three-way handshake), also reflects the RTT. In practice, from any capture point, the time from the first SYN packet to the third ACK packet is a good representation of the RTT as well assuming the client and server response times during the handshake process are small. Be aware that the server response time to a SYN packet, while usually short, can be longer than normal during periods of high loading and can affect this measurement.

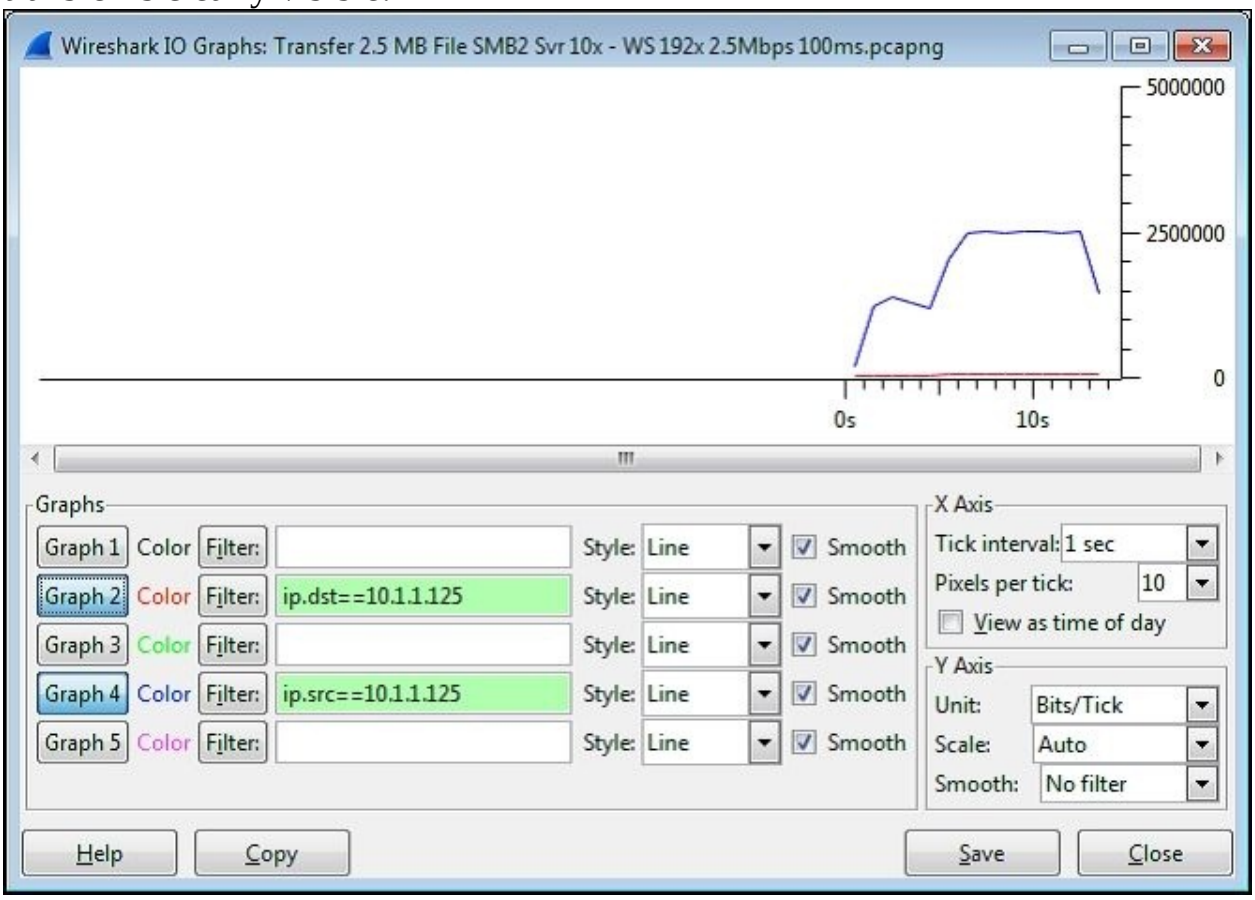
High network path latency isn't an error condition by itself, but can obviously have adverse effects on the application's operation over the network as

previously discussed.

Bandwidth congestion

Bandwidth congestion affects the application's performance by extending the amount of time required to transmit a given amount of data over a network path; for users accessing an application server over a busy WAN link, these effects can become significant. A network support team should be able to generate bandwidth usage and availability reports for the in-path WAN links to check for this possibility, but you can also look for evidence of bandwidth congestion by using a properly configured Wireshark IO Graph to view network throughput during larger data transfers.

The following screenshot illustrates a data transfer that is affected by limited bandwidth; the flatlining at the 2.5 Mbps mark (the total bandwidth availability in this example), because no more bandwidth is available to support a faster transfer is clearly visible:



You can determine the peak data transfer rate in **bits-per-second** (**bps**) from an IO Graph by configuring the graph as follows:

- **X Axis Tick interval**: **1 sec**
- **Y Axis Unit**: **Bits/tick**
- **Graph 2 Filter**: `ip.dst == <IP address of server>`
- **Graph 4 Filter**: `ip.src == <IP address of server>`

These settings result in an accurate bits-per-second display of network throughput in client-to-server (red color) and server-to-client (blue color) directions. The **Pixels per tick** option in the **X Axis** panel, the **Scale** option in the **Y Axis** panel, and other settings can be modified as desired for the best display without affecting the accuracy of the measurement.

Be aware that most modern applications can generate short-term peak bandwidth demands (over an unrestricted link) of multiple Mbps. The WAN links along a network path should have enough spare capacity to accommodate these short term demands or response time will suffer accordingly. This is an important performance consideration.

Data transport

There are a number of TCP data transport effects that can affect application performance; these can be analyzed in Wireshark.

TCP StreamGraph

Wireshark provides TCP **StreamGraphs** to analyze several key data transport metrics, including:

- **Round-trip time**: This graphs the RTT from a data packet to the corresponding ACK packet.
- **Throughput**: These are plots throughput in bytes per second.
- **Time/sequence (Stephen's-style)**: This visualizes the TCP-based packet sequence numbers (and the number of bytes transferred) over time. An ideal graph flows from bottom-left to upper-right in a smooth fashion.
- **Time/sequence (tcptrace)**: This is similar to the Stephen's graph, but provides more information. The data packets are represented with an I-bar

display, where the taller the I-bar, the more data is being sent. A gray bar is also displayed that represents the receive window size. When the gray bar moves closer to the I-bars, the receive window size decreases.

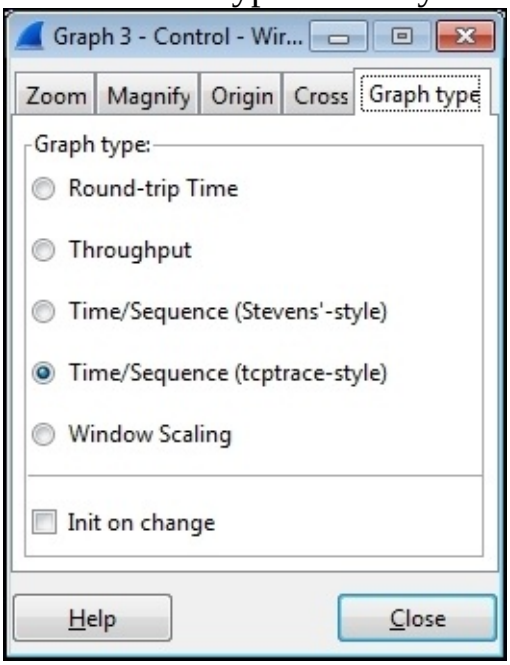
- **Window Scaling**: This plots the receive window size.

  ## Note

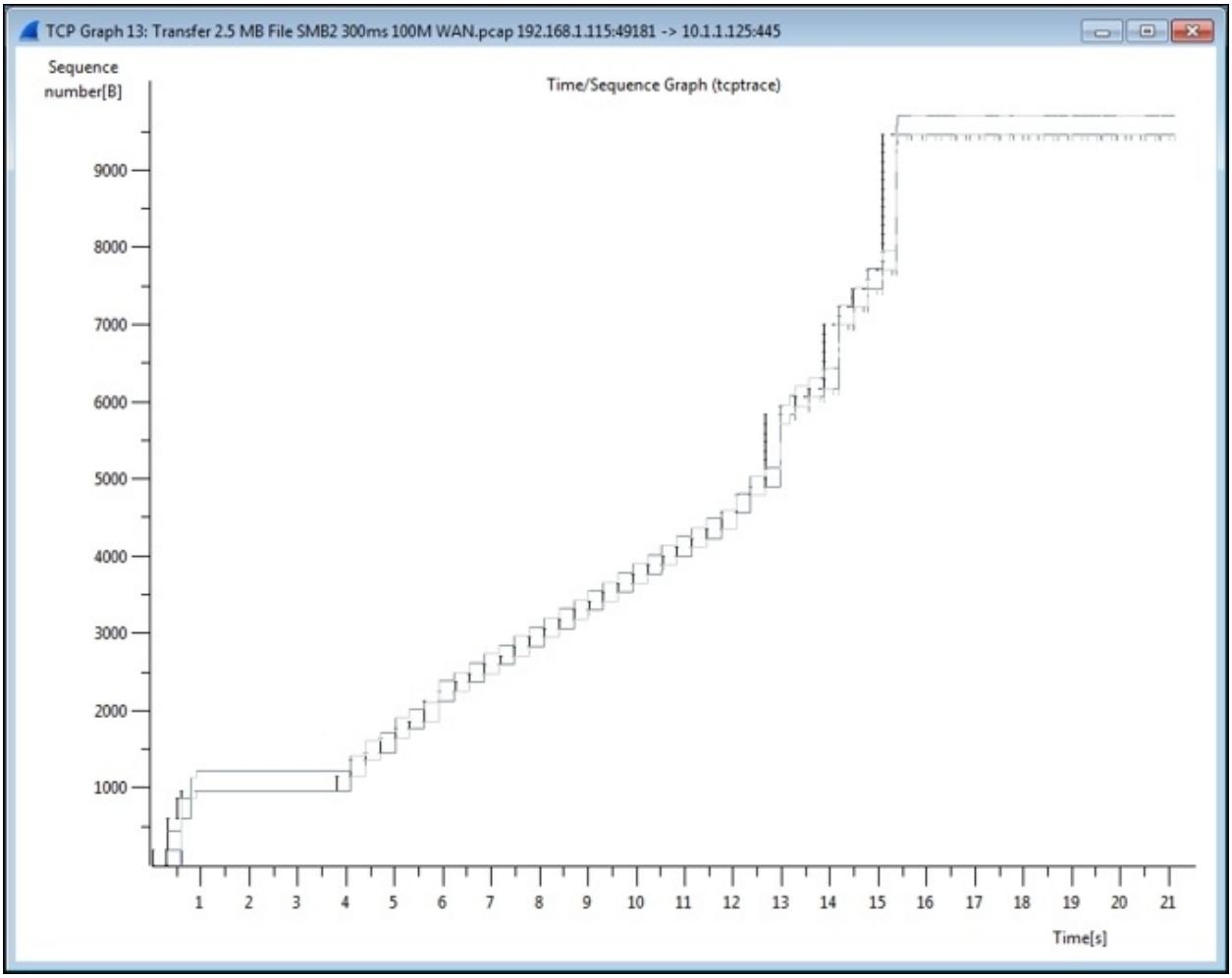
  The TCP StreamGraphs are unidirectional. You want to select a packet for the direction that is transporting data to get the proper view.

These analysis graphs can be utilized by selecting one of the packets in a TCP stream in the **Packet List** pane and selecting **TCP StreamGraph** from the **Statistics** menu and then one of the options such as the **Time-Sequence Graph (tcptrace)**.

The selected graph and **Control Window** will appear from the **Graph type** tab of the **Control Window** that you can select one of the other types of analysis graphs, as shown in the following screenshot:



The **Time/Sequence Graph (tcptrace)** shown in the following screenshot plots sequence numbers as they increase during a data transfer, along with the gray receive window size line:



You can click and drag the mouse over a section of the graph to zoom into a particular section, or press the + key to zoom in and the - key to zoom out. Clicking on a point in any of the graphs will take you to the corresponding packet in the Wireshark's **Packet List** pane.

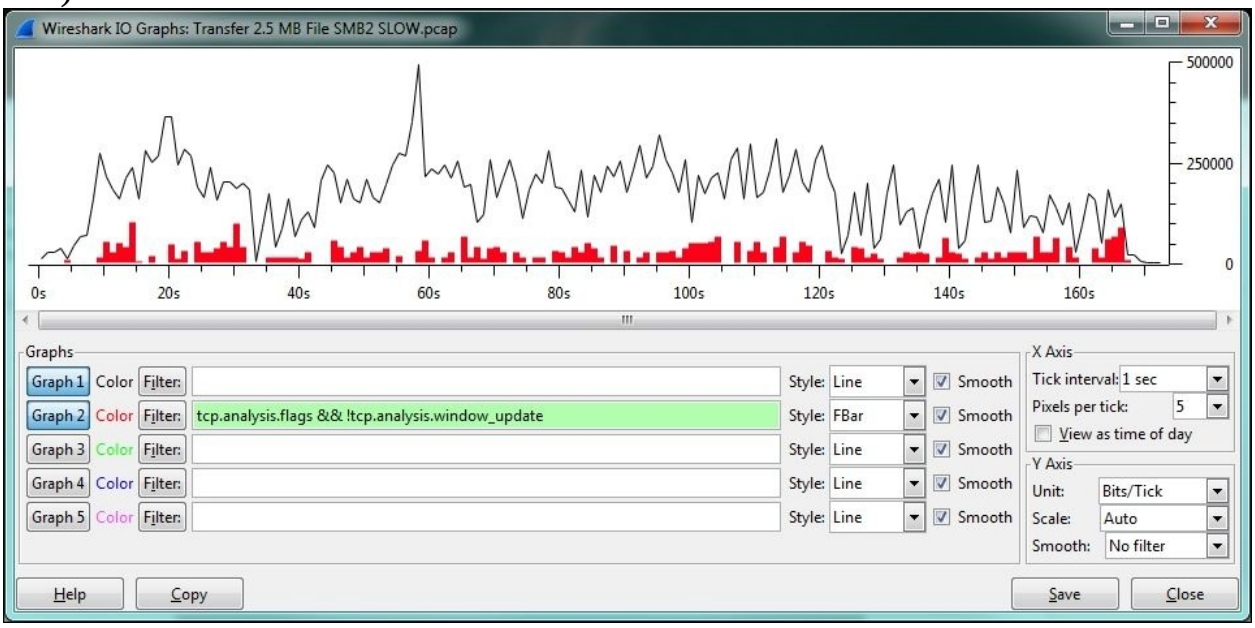
IO Graph

You can also analyze a the effects of TCP issues on network throughput by applying TCP analysis display filter strings to Wireshark's IO Graph, such as:

```
tcp.analysis.flags && !tcp.analysis.window_update
```

In the following screenshot of a slow SMB data transfer, it can be seen that the multiple TCP issues (in this case, packet loss, Duplicate ACKs, and

retransmissions) in the red line correspond to a decrease in throughput (the black line):

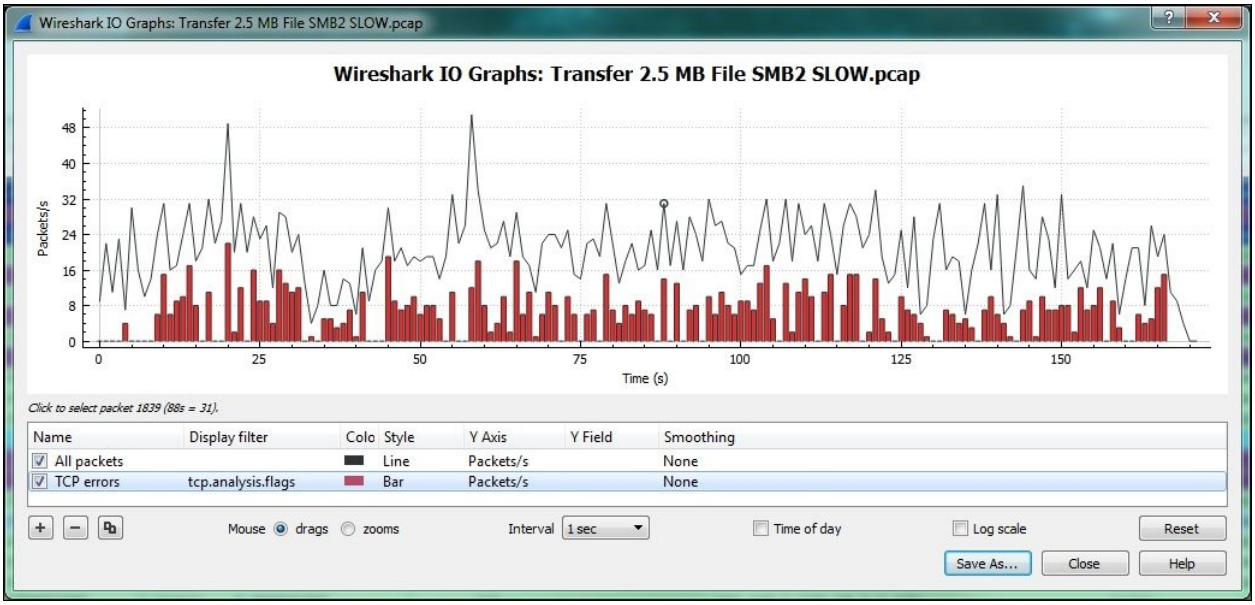


Clicking on a point in the IO Graph takes you to the corresponding packet in the Wireshark's **Packet List** pane so you can investigate the issue.

IO Graph – Wireshark 2.0

Wireshark 2.0, also known as Wireshark Qt, is a major change in Wireshark's version history due to a transition from the GTK+ user interface library to Qt to provide better ongoing UI coverage for the supported platforms. Most of the Wireshark features and user interface controls will remain basically the same, but there are changes to the IO Graph.

These are shown in the following screenshot, which shows the same TCP issues that were seen in the preceding screenshot:



The new IO Graph window features the ability to add as many lines as desired (using the + key) and to zoom in on a graph line, as well as the ability to save the graph as an image or PDF document.

Summary

The topics covered in this chapter included troubleshooting methodology, how to use Wireshark to troubleshoot connectivity and functionality issues, performance analysis methodology, and the top five causes of poor application performance and how to use Wireshark to analyze those causes.

In the next chapter, we will review some of the common types and sources of malicious traffic and introduce how a security professional can use Wireshark to detect these threats.

Chapter 7. Packet Analysis for Security Tasks

With the increasing threat of hackers, identity thieves, and corporate data theft, you need to be able to analyze the security of your network at the packet level.

The topics that will be covered in this chapter include:

- Security analysis methodology
- Scans and sweeps
- OS fingerprinting
- Malformed packets
- Phone home traffic
- Password cracking traffic
- Unusual traffic

Security analysis methodology

Security analysis at the packet level is based on detecting and analyzing suspect traffic, that is, the traffic that does not match normal patterns because of the presence of unusual protocol types or ports, or unusual requests, responses, or packet frequency. Suspicious traffic may include reconnaissance (discovery) sweeps, phone home behavior, denial of service attacks, botnet commands, or other types of behavior from direct attacks or virus- or botnet-based agents.

Wireshark captures strategic points in the network to investigate suspicious packets from specific hosts or on network segments and egress points can also complement any **Intrusion Detection System** (**IDS**) systems that may be in place to alert the IT staff about the suspicious traffic.

The importance of baselining

The ability to identify abnormal traffic patterns that bear investigation versus traffic caused by poorly behaving applications, misconfigurations, or faulty devices can be made much easier if you have a baseline of what is normal. A baseline is a snapshot capture of typical conversations with your primary applications and servers and the background traffic on the network segments that they reside on. In a potential security breach situation, you can compare the normal protocols, traffic patterns, and user sessions from a baseline with a current capture, filter out the normal traffic, and then inspect the differences.

To allow the comparison of baselines in your security analysis, you need to periodically capture and store packet trace files that cover a sufficient period of time to provide a good sample of typical user and background traffic patterns while keeping the file sizes manageable for use within Wireshark, for example, 100 MB to 1 GB per file. You can configure the **Ring Buffer** option within Wireshark's **Capture Options** window to save a series of reasonably sized files for longer captures or busier network segments.

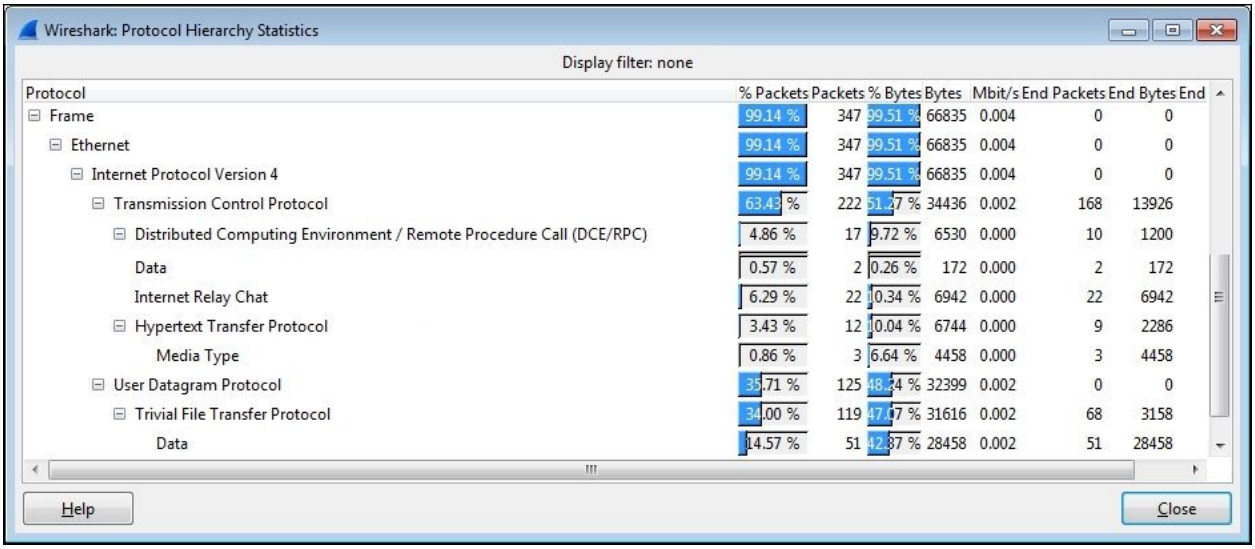
Although your baselining needs and practices will depend on your environment, some of the traffic aspects that you should inspect include:

- Broadcast and multicast types and rates:
  - What devices and applications are using broadcasts and multicasts?
  - What are the typical broadcast and multicast packet rates?
- Applications and protocols:
  - What applications are running over the network?
  - What protocols and ports are they using?
  - Application launch sequences and typical tasks
  - Are application sessions encrypted?
  - Are all users forced to use encryption? Any exceptions?
  - What are the login/logout sequences and dependencies?
- Routing protocol(s) and routing updates
- ICMP traffic
- Boot-up sequences
- Name resolution sessions
- Wireless connectivity includes normal management, control, and data frame

contents

- VoIP and video communications
- Idle time traffic is the host communicating with other hosts when there are no users logged in
- What backup processes are running at night and for how long?
- Are there any suspect protocols or broadcasts/scans taking place?

As you inspect your baseline captures, it is helpful to view a summary of the protocols being used by selecting **Protocol Hierarchy** from the Wireshark's **Statistics** menu. In the following screenshot, for example, you can see that there is some **Internet Relay Chat** (**IRC**) traffic, as well as the **Trivial File Transfer Protocol** (**TFTP**) traffic, neither of which might be normal on your network and could be an indication of rogue communications with outside entities:



Analyzing baselines of normal traffic levels and patterns is also an excellent way of getting familiar with your network environment and its typical packet flows and protocols, which better prepares you to spot abnormal traffic.

Security assessment tools

There are several popular tools that are used by security professionals to perform security assessment and vulnerability testing. As these tools can generate the same types of scans, fingerprinting, and other exploitive activities, as might be used by hackers and malicious agents, they can be useful to a packet analyst to analyze the packets that they generate with Wireshark to build familiarity with how different types of activities appear in a packet trace and also to build display filters to detect them.

One of the most popular tools is **Network Mapper** (**Nmap**), a free and open source utility for network discovery and security auditing. Nmap runs on all major computer operating systems and offers a command-line and GUI version (**Zenmap**).

Note

You can find more information about Nmap at http://nmap.org and information on other top security tools can be found at http://sectools.org.

Identifying unacceptable or suspicious traffic

Wireshark can be used to identify unusual patterns or packet contents in the network traffic including network scans, malformed packets, and unusual protocols, applications, and or conversations that should not be running on your network. The following is a general list of traffic types that may not be acceptable and/or warrant investigation to validate their legitimacy in your environment:

- **MAC or IP address scans**: These attempt to identify active hosts on the network
- **TCP or UDP port scans**: These attempt to identify active applications and services

IP address and port scans can be generated from network management applications to build or maintain their list of devices and applications to monitor/manage, but that's usually the only legitimate source of these types of traffic.

- **Clear text passwords**: These are passwords that you can see in the Wireshark's **Packet Details** or **Packet Bytes** fields. These are typical for **File Transfer Protocol** (**FTP**) logins, but not typical or acceptable elsewhere.
- **Clear text data**: This is the data in packet payloads that can be read. This is typical for HTTP requests and responses and commonly seen in application server to database requests and responses, but these database exchanges should be between hosts on isolated, nonpublic network segments and otherwise physically secure environments.
- **Password cracking attempts**: These are repeated, systematic attempts to discover a working password, usually from a single device.
- **Maliciously formed packets**: These are packets with intentionally invalid or improperly formatted data in protocol fields that are intended to exploit vulnerabilities in applications.
- **Phone home traffic**: This is the traffic from a rogue agent that may be resident on a server or workstation that periodically checks in with a remote (usually off-network) host.

- **Flooding or Denial of Service (DOS) attacks**: This is the traffic that is intentionally sent at a very high packet-per-second rate to one or more hosts in an attempt to flood the host(s) or network with so much traffic that no one else can access their services.
- **Subversive activities**: These include a number of techniques to prepare for and facilitate the man-in-the-middle attacks where a device is tricked into sending packets to a malicious host for the purpose of intercepting data.

This is only a sampling of types of malicious traffic that you might see on your network; network security is an ever evolving exchange of increasingly sophisticated attacks and subsequent countermeasures.

As you develop your security analysis skills, you might want to build a special security profile in Wireshark that includes packet coloring rules based on display filters to help identify suspicious or malformed packets, as well as a set of **Filter Expression Buttons** that isolate and display various types of questionable traffic you might be looking for.

Some examples of display filters to isolate and inspect suspicious packets include:

Filter description	**Display filter string**
Detect ICMP pings and possible ping sweep	`icmp.type == 8 || icmp.type == 0`
ICMP destination unreachable filter (included redirects)	`(icmp.type >= 3 && icmp.type <= 5) || icmp.type == 11 || (icmpv6.type >= 1 && icmpv6.type <= 4)`
Unusual ICMP echo requests	`(icmp.type == 8) && !(icmp.code == 0x00)`
TCP handshakes useful for detecting TCP scans as well as inspecting normal session setups/tear-downs/resets	`(tcp.flags&02 && tcp.seq==0) || (tcp.flags&12 && tcp.seq==0) || (tcp.flags.ack && tcp.seq==1 && !tcp.nxtseq > 0 && !tcp.ack >1) || tcp.flags.fin == 1 || tcp.flags.reset ==1`
Detect Xmas scan (URG, FIN, and PUSH flags set)	`tcp.flags == 0x029`
Other suspicious TCP settings: TCP SYN/ACK	`((tcp.flags == 0x02) && (tcp.window_size < 1025)) || tcp.flags == 0x2b || tcp.flags == 0x00`

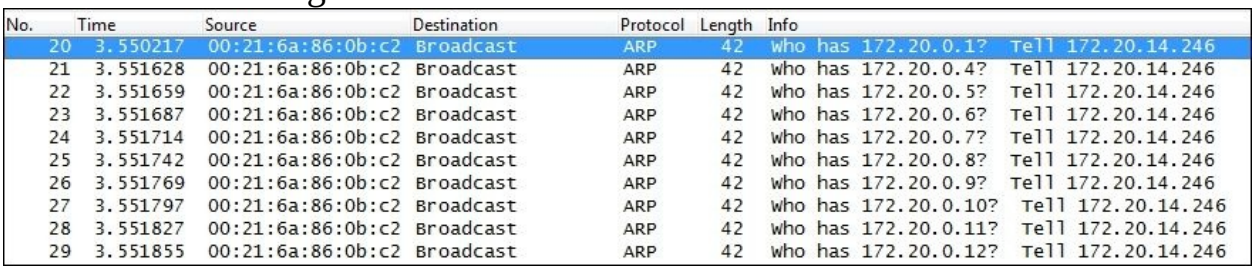
w/ Win size greater than 1025, SYN, FIN, PSH, URG bits set, no TCP flags set, TCP max segment size set to less than 1460	`|| tcp.options.mss_val < 1460`
Internet Relay Chat (**IRC**) traffic (is this normal in your network?)	`tcp.port == 194 || (tcp.port >= 6660 && tcp.port <= 6669) || tcp.port == 7000`
High number of DNS answers (could be a list of command and control servers)	`dns.count.answers > 5`

Scans and sweeps

Malicious programs and rogue processes might investigate a network environment for available ports and hosts using various scanning processes before launching an exploit. Identifying the presence of these reconnaissance processes may allow thwarting the attack before it is launched, as well as tracking down and/or blocking the source of the malicious activity—especially if that source is inside the company as some of them are.

ARP scans

ARP scans, also called as ARP sweeps, are used to discover active localhosts on a network segment. An ARP sweep can be difficult to detect unless you apply a display filter and observe a steady, incremental sweep from the same device, as seen in the following screenshot:



No.	Time	Source	Destination	Protocol	Length	Info
20	3.550217	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.1? Tell 172.20.14.246
21	3.551628	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.4? Tell 172.20.14.246
22	3.551659	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.5? Tell 172.20.14.246
23	3.551687	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.6? Tell 172.20.14.246
24	3.551714	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.7? Tell 172.20.14.246
25	3.551742	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.8? Tell 172.20.14.246
26	3.551769	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.9? Tell 172.20.14.246
27	3.551797	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.10? Tell 172.20.14.246
28	3.551827	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.11? Tell 172.20.14.246
29	3.551855	00:21:6a:86:0b:c2	Broadcast	ARP	42	Who has 172.20.0.12? Tell 172.20.14.246

As ARP packets cannot pass through a router, the source device conducting the ARP sweep must be on the same network segment that the ARP packets are seen on.

ICMP ping sweeps

ICMP ping sweeps are used to discover active hosts on local or remote network segments (since ICMP uses IP and is routable) using ICMP Type 8 Echo Requests and Type 0 Echo Replies for a range of IP addresses. You can easily detect ping sweeps by using a display filter `icmp.type == 8 || icmp.type == 0`.

TCP port scans

TCP port scans allow a malicious agent to discover which TCP ports are open on a target host. Network ports are the entry points to a server or workstation; a service that listens on a given port is able to service requests from a client. Malicious agents can sometimes exploit vulnerabilities in server code to gain access to sensitive data or execute malicious code on the machine, which is why testing all active ports is necessary for a complete coverage of any security validation.

Some of the most common ports used for TCP-based services include:

- 80 HTTP
- 443 HTTPS
- 8080 HTTP proxy
- 8000 HTTP alternate
- 21 FTP
- 22 SSH
- 23 Telnet
- 3389 Microsoft Remote Desktop
- 5900 VNC
- 25 SMTP
- 110 POP3
- 143 IMAP
- 3306 MySQL
- 1433 Microsoft SQL Server
- 1720 H.323
- 5060 SIP

A TCP port scan device will send a TCP SYN packet to a port on a target host, which will respond with either SYN, or ACK if the port is open, or RST if the port is closed. Similar to an ARP scan, a TCP scan can be detected by a series of SYN packets from a single IP address to a target IP address over a range of port numbers. A display filter can make detecting these types of scans easier:

```
ip.dest == <IP Address of target host> && tcp.flags.syn
```

UDP port scans

UDP port scans are like TCP scans, but they are run against typical UDP-based services, the most common of which include:

- 53 DNS
- 161/162 SNMP
- 67/68 DHCP
- 5060 SIP
- 135 Microsoft Endpoint Mapper
- 137/139 NetBIOS Name Service

The preceding topics cover just a sampling of the most common scans used by malicious agents. Security analysts should research this topic further to identify all the types of scans that may be used to exploit their particular environment's vulnerabilities.

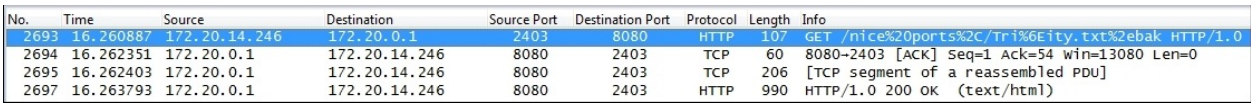
OS fingerprinting

OS fingerprinting is a technique wherein a remote machine sends various types of commands to a target device and analyzes the responses to attempt to identify the target devices' operating system and version. Knowing which operating system a device is running makes it possible to use exploits specific to that operating system.

Nmap detects operating systems based on a series of port scans, ICMP pings, and numerous other tests, and then runs a set of follow-up tests based on the results to further define the OS version running.

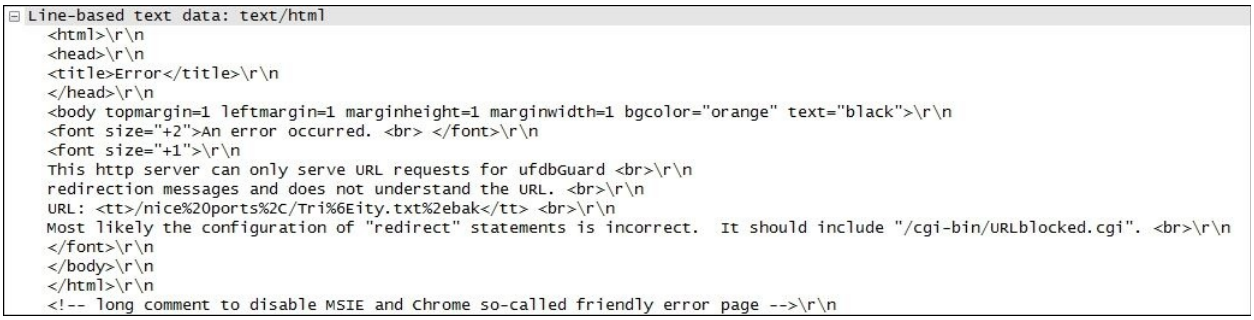
In the following screenshot, you can see the test results verbiage from the GUI version of Nmap (Zenmap) as it completes an OS detection scan, as well as its best estimate of the operating system and version:

A Wireshark capture of the OS detection activity described earlier included as an example of one of the OS fingerprinting scripts that are run, a `bogus` HTTP request to the target device (`172.20.0.1`) for `/nice%20ports%2C/Tri%6Eity.txt%2ebak` to see exactly what kind of error response was generated, which is used to help pinpoint the OS version:



No.	Time	Source	Destination	Source Port	Destination Port	Protocol	Length	Info
2693	16.260887	172.20.14.246	172.20.0.1	2403	8080	HTTP	107	GET /nice%20ports%2C/Tri%6Eity.txt%2ebak HTTP/1.0
2694	16.262351	172.20.0.1	172.20.14.246	8080	2403	TCP	60	8080→2403 [ACK] Seq=1 Ack=54 Win=13080 Len=0
2695	16.262403	172.20.0.1	172.20.14.246	8080	2403	TCP	206	[TCP segment of a reassembled PDU]
2697	16.263793	172.20.0.1	172.20.14.246	8080	2403	HTTP	990	HTTP/1.0 200 OK (text/html)

The exact format of the HTML response from the preceding request could be used to identify the OS and/or web server version, as seen in the following Wireshark packet details screenshot:



```
Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>Error</title>\r\n
    </head>\r\n
    <body topmargin=1 leftmargin=1 marginheight=1 marginwidth=1 bgcolor="orange" text="black">\r\n
    <font size="+2">An error occurred. <br> </font>\r\n
    <font size="+1">\r\n
    This http server can only serve URL requests for ufdbGuard <br>\r\n
    redirection messages and does not understand the URL. <br>\r\n
    URL: <tt>/nice%20ports%2C/Tri%6Eity.txt%2ebak</tt> <br>\r\n
    Most likely the configuration of "redirect" statements is incorrect.  It should include "/cgi-bin/URLblocked.cgi". <br>\r\n
    </font>\r\n
    </body>\r\n
    </html>\r\n
    <!-- long comment to disable MSIE and Chrome so-called friendly error page -->\r\n
```

Analyzing packet captures of these kinds of OS fingerprinting requests and responses will make it much easier to spot similar activities from malicious entities.

Malformed packets

Maliciously malformed packets take advantage of vulnerabilities in operating systems and applications by intentionally altering the content of data fields in network protocols. These vulnerabilities may include causing a system crash (a form of denial of service) or forcing the system to execute the arbitrary code.

An example of malformed packet vulnerability is Cisco Security Advisory *cisco-sa-20140611-ipv6*, wherein vulnerability in parsing malformed IPv6 packets in a certain series of routers could cause a reload (reboot) of a certain card that carries network traffic, which could intermittently cause service outages.

Another example of this kind of vulnerability is in some unpatched Windows or Linux systems that will crash if they receive a series of fragmented packets where the fragments overlap each other.

The types and possibilities of malformed packets are endless, but vulnerabilities are usually announced as they are discovered and some may provide packet details. You can build display filters and/or build coloring rules in Wireshark to detect these packets. It also helps to study and understand what range of values the different protocol fields normally and legally contain, and what TCP and other protocol sequences normally look like so you can spot suspicious contents in packet flows.

Phone home traffic

Phone home traffic originates from a rogue application on a device that periodically connects to a remote (usually off-network) host to receive updates or commands or deliver data collected from the infected host. The majority of phone home traffic will be the operating system and virus protection updates, Dropbox or other external services, and similar authorized and appropriate services, so it will take some effort to identify malicious traffic out of this mix.

It is important to understand the risk that phone home traffic can represent: many botnet **Distributed Denial of Service** (**DDoS**) attacks are supported by a "zombie army" of hijacked computers running software that may lie undetected for some period of time except for periodic communications with their **Command and Control** (**C&C**) servers awaiting instructions to attack a target. In a similar fashion, keylogging traffic will send periodic reports of video screenshots and keystroke data to the collecting host.

One way to identify potentially malicious phone home traffic is to capture and inspect the DNS queries as these sessions start up, looking at two distinct areas:

- The hostname(s) of legitimate services are often reasonably recognizable.
- DNS queries for illegitimate applications contacting C&C servers will often return a long list of aliases with IP addresses that are not all in the same general range (that is, from all over the world). A display filter that helps identify DNS responses with long response lists is `dns.count.answers > 5`.

It also helps to have a baseline that includes the idle period traffic and a sample of known updates/services dialogs to compare a questionable capture to.

Password-cracking traffic

Password-cracking traffic can be detected by observing numerous error messages from a target host directed to a client that repeatedly and unsuccessfully attempts to log in. There are two general types of password cracking attempts:

- Dictionary attacks work from a list of common words, names, and numbers
- Brute force attacks use a sequence of characters, numbers, and key values

Both of these types are often thwarted by login security measures that lock out an account after a short number of failed login attempts.

Unusual traffic

While it is difficult to anticipate what methods a hacker may use in an attempt to infiltrate a network or host, there are a few things that should probably never happen on a normal, healthy network. Due to their usefulness in testing and conveying error conditions, ICMP packets are a likely target for malicious redirection. Since TCP is the predominant transport protocol in use for most applications, you should look out for abnormalities in TCP headers or payloads that could be a sign of malicious intent.

Some examples of abnormalities to look out for are discussed in the following table:

Suspicious content	**Description**
TCP bad flags	An illegal or unlikely combination of TCP flags. The SYN, SYN/ACK, ACK, PSH, FIN, and RST flags are normal when they're used in the appropriate places; anything otherwise warrants investigation.
SYN packet contains data	The initial TCP SYN packet should never contain payload data; it is used to establish a session only. Note, however, that the third ACK packet in the TCP can contain data.
Suspicious datagram payload contents	References to the operating system or other non-application directories, strange executables, or other payload data that doesn't seem to fit the purpose of the application being used to send the data.
Suspicious ping payload text	The text used to fill in the payload of an ICMP Echo Request packet is usually a benign sequential series of letters and numbers or similar meaningless text. If this text appears to carry commands or meaningful data, it warrants investigation.
Clear text passwords in FTP or Telnet sessions	Seeing FTP used to transport sensitive business data, or Telnet to administer switches and routers, isn't malicious intent by a hacker. It's negligent practice by employees as both protocols, by design, transmit clear text login IDs and passwords over the network, making it easy for even an unsophisticated hacker to capture them. There are **Secure FTP** (**sftp**) and **Secure Shell** (**SSH**) (Telnet alternative) solutions for all platforms available on the Web.

Summary

The topics covered in this chapter on security analysis included detecting scans and sweeps to identify targets for planned attacks, operating system fingerprinting, detecting malformed packets, and packets that are suspiciously fragmented or sent out of order, phone home traffic from malicious agents, identifying password cracking attempts, and identifying other abnormal packets and payloads.

In the next chapter, we'll review several key command-line utilities provided in a Wireshark installation, as well as a few additional packet analysis tools that can complement your toolset.

Chapter 8. Command-line and Other Utilities

Wireshark includes a number of command-line utilities to manipulate packet trace files and offer GUI-free packet captures, and there are a few other tools that can help round out your analysis toolset.

The topics that will be covered in this chapter include:

- Capturing traffic with Dumpcap and Tshark
- Editing trace files with Editcap
- Merging trace files with Mergecap
- Other helpful tools

Wireshark command-line utilities

When you install Wireshark, a range of command-line tools also gets installed, including:

- `capinfos.exe`: This prints information about trace files
- `dumpcap.exe`: This captures packets and saves to a libpcap format file
- `editcap.exe`: This splits a trace file, alters timestamps, and removes duplicate packets
- `mergecap.exe`: This merges two or more packet files into one file
- `rawshark.exe`: This reads a stream of packets and prints field descriptions
- `text2pcap.exe`: This reads an ASCII hex dump and writes a libpcap file
- `tshark.exe`: This captures network packets or displays data from a saved trace file

The `Wireshark.exe` file launches the GUI version you're familiar with, but you can also launch Wireshark from the command line with a number of parameters; type `Wireshark –h` for a list of options and/or create shortcuts to launch Wireshark with any of those options.

Note

It is very helpful to add the Wireshark program directory to your system's `PATH` statement so that you can execute any of the command-line utilities from any working directory.

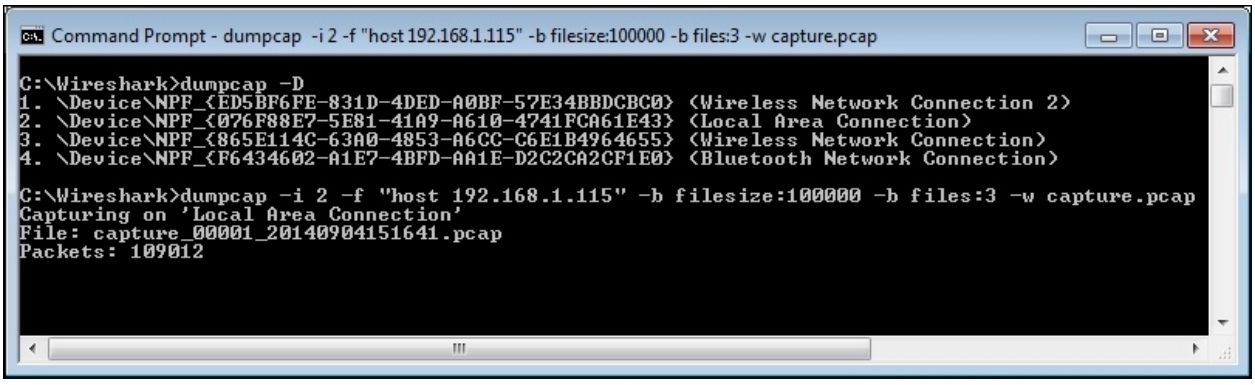
Capturing traffic with Dumpcap

The `dumpcap.exe` file is the executable that Wireshark actually runs under the covers to capture packets and save them to a trace file in libpcap format. You can run Dumpcap on the command line to circumvent using the Wireshark GUI and use fewer resources. A list of command-line options is available by typing `dumpcap.exe -h`.

Some of the most useful options are as follows:

- `-D`: This prints a list of available interfaces and exits
- `-i <interface>`: This specifies a name or index number of an interface to capture on
- `-f <capture filter>`: This applies a capture filter in the **Berkeley Packet Filter** (**BPF**) syntax
- `-b filesize`: This is the file size
- `-w <outfile>`: This is the name of the file where the files will be saved

An example of viewing a list of interfaces and then running Dumpcap to capture a specific interface with an IP address capture filter (note the use of quotes around the filter syntax) configured to use a three-file ring buffer with file sizes of 100 MB and an output filename derived from `capture.pcap` is illustrated in the following screenshot:



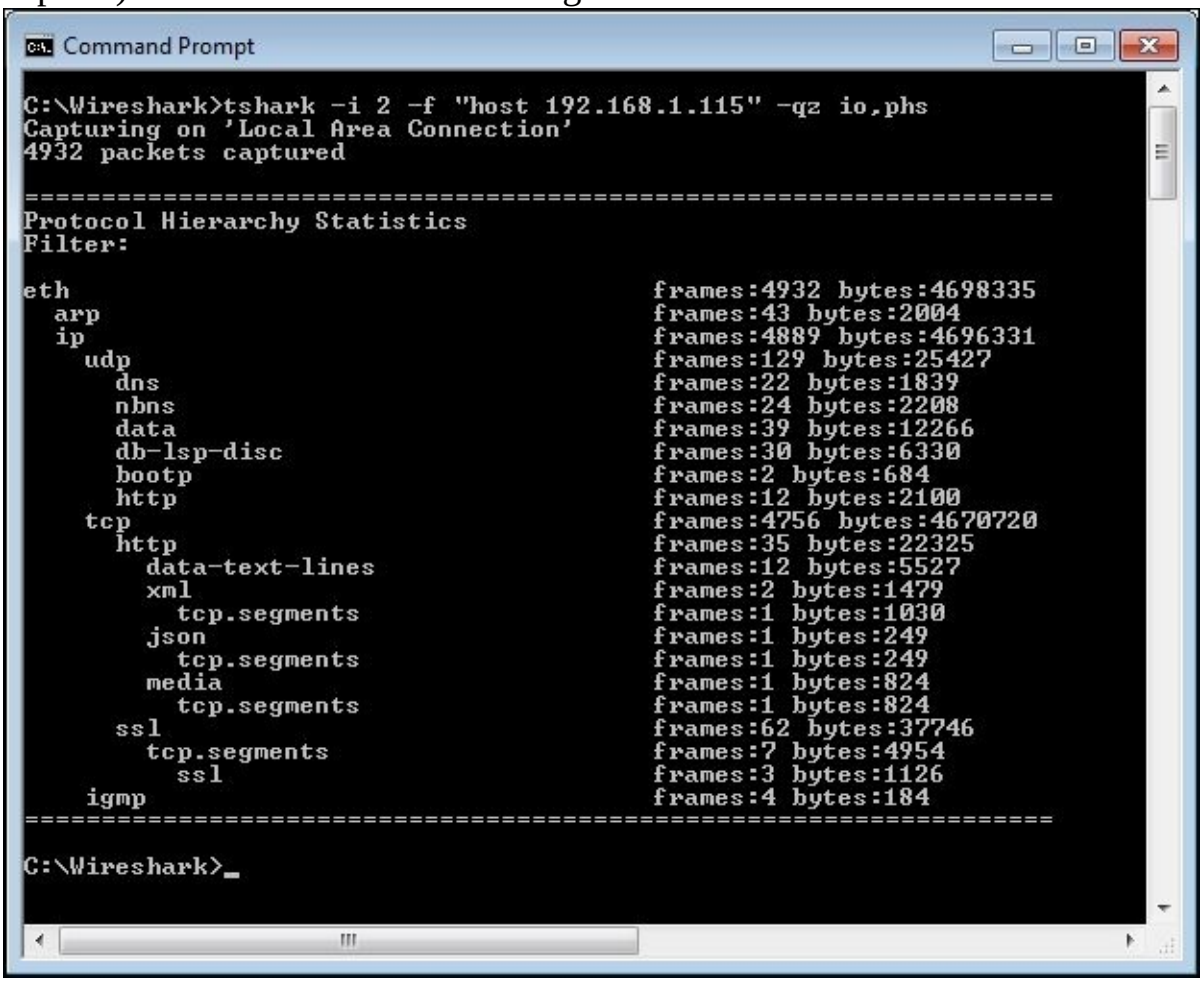
You can get more information on Dumpcap options at https://www.wireshark.org/docs/man-pages/dumpcap.html.

Capturing traffic with Tshark

Tshark can be used to capture network packets and/or display data from the capture or a previously saved packet trace file; packets can be displayed on the screen or saved to a new trace file.

The same syntax used to perform a basic capture using Dumpcap will work with Tshark as well, so we won't repeat that here. However, Tshark offers a very wide range of additional features, with a corresponding large number of command-line options that can, as in all Wireshark utilities, be viewed by typing `tshark -h` in the command prompt.

A number of Tshark options are to view statistics; an example of the command syntax and statistical results from a capture (after pressing *Ctrl + C* to end the capture) is illustrated in the following screenshot:



You will find an extensive number of details and examples on using statistics and other Tshark options at https://www.wireshark.org/docs/man-pages/tshark.html.

Editing trace files with Editcap

You can use Editcap to split a trace file that is too large to work with in Wireshark into multiple smaller files, extract a subset of a trace file based on a start and stop time, alter timestamps, remove duplicate packets, and a number of other useful functions.

Type `editcap -h` in the command prompt for a list of options. The syntax to extract a single packet or a range of packets by packet numbers is as follows:

```
editcap -r <infile> <outfile> <packet#> [- <packet#>]
```

You must specify `<infile>` and `<outfile>`. The `-r` specifies to keep, not delete, the specified packet or packet range, for example:

```
editcap -r MergedTraces.pcapng packetrange.pcapng 1-5000
```

You can split a source trace file into multiple sequential files, each containing the number of packets specified by the `-c` option:

```
editcap -c 5000 MergedTraces.pcapng SplitTrace.pcapng
```

You can eliminate duplicate packets in a file within a five-packet proximity:

```
editcap -d hasdupes.pcapng nodupes.pcapng
```

If you have two trace files that have a significant span of time between them, and you want to merge them into one file but closer together, you can investigate all of the packets within one IO Graph or a similar analysis function; you can first use the `-t` option on one of the files to adjust the timestamps in that file by a constant amount (in seconds). For example, to subtract 5 hours from a trace file's timestamps, use the following command:

```
editcap -t -18000 packetrange.pcapng adj_packetrange.pcapng
```

Comparing the two traces in Wireshark reveals the following details:

- **Packet #500 before adjustment**: `2014-09-04 15:27:38.696897`
- **Packet #500 after adjustment**: `2014-09-04 10:27:38.696897`

You can get more information on and examples of Editcap options at https://www.wireshark.org/docs/man-pages/editcap.html.

Merging trace files with Mergecap

You can use Mergecap to merge two or more trace files into one file. The basic syntax is as follows:

```
mergecap –w <outfile.pcapng>  infile1.pcapng   infile2.pcapng  …
```

For example:

```
mergecap –w merged.pacap   source1.pcapng   source2.pcapng
source3.pcapng
```

One useful option you sometimes may want to use in Mergecap (and several of the other command-line utilities) is `–s <snaplen>`. This will truncate the packets at the specified length past the start of each frame, resulting in a smaller file; a typical value for `<snaplen>` is 128 bytes:

```
mergecap –w merged_trimmed.pcapng  -s 128  source1.pcapng
source2.pcapng
```

Mergecap batch file

If the capture files you want to merge have a variety of naming formats, you can create a `MergeTraces.bat` file containing the following Windows batch commands:

```
@echo off
cls
echo MergeTraces.bat
echo.
echo Merges multiple packet trace files with a .pcapng extension
into one .pcapng file
echo.
echo Usage: Copy MergeTraces.bat into the directory with the .pkt
files and execute
echo The utility will generate a 'MergedTraces.pcap' file
echo and a 'MergedFileList.txt' file which lists the .pcapng files
processed.
echo.
echo.
echo IMPORTANT!! You must type 'CMD /V:ON' from this window which
enables
echo 'Delayed environment variable expansion' in order to properly
execute
echo this batch utility.
echo.
echo You must also add the path to Wireshark's mergecap.exe to your
path statement.
echo.
echo If you've not done this, Type Ctrl-C to exit; Otherwise
pause
echo.
echo Deleting old MergedFileList.txt...
if exist "MergedFileList.txt" del MergedFileList.txt
for %%f in (*.pcap-ng) do echo "%%f" >> MergedFileList.txt
echo Deleting old MergedTraces.pcapng...
if exist "MergedTraces.pcapng" del MergedTraces.pcapng
echo Preparing to merge:
echo.
type MergedFileList.txt
echo.
echo Merging.........
set FILELIST=
for %%f in (*.pcap-ng) do set FILELIST=!FILELIST! %%f
:: DEBUG
```

```
:: echo %FILELIST%
mergecap -w MergedTraces.pcapng %FILELIST%
echo.
if exist MergedTraces.pcapng @echo Done!
if NOT exist MergedTraces.pcapng @echo Error!! -- Check your
settings.
echo.
```

Copy the batch file into a directory containing just the packet trace files you want to merge and execute it. The batch file will merge all the `.pcapng` files into one file called `MergedTraces.pcapng`. This is much easier than trying to specify a long list of unique source files in a command line, especially if the filenames contain date-time stamps. If you need to work with the `.pcap` files, change all instances of `.pcapng` to `.pcap` in the batch commands; you can also alter the output filename as desired.

Note

You can also merge trace files by clicking-and-dragging the files into the Wireshark desktop. The files will be merged in chronological order based on their timestamps after selecting **Merge** from the Wireshark **File** menu. This works reasonably well as long as the total file size doesn't exceed 1GB.

You can get more info and examples of Mergecap options at https://www.wireshark.org/docs/man-pages/mergecap.html.

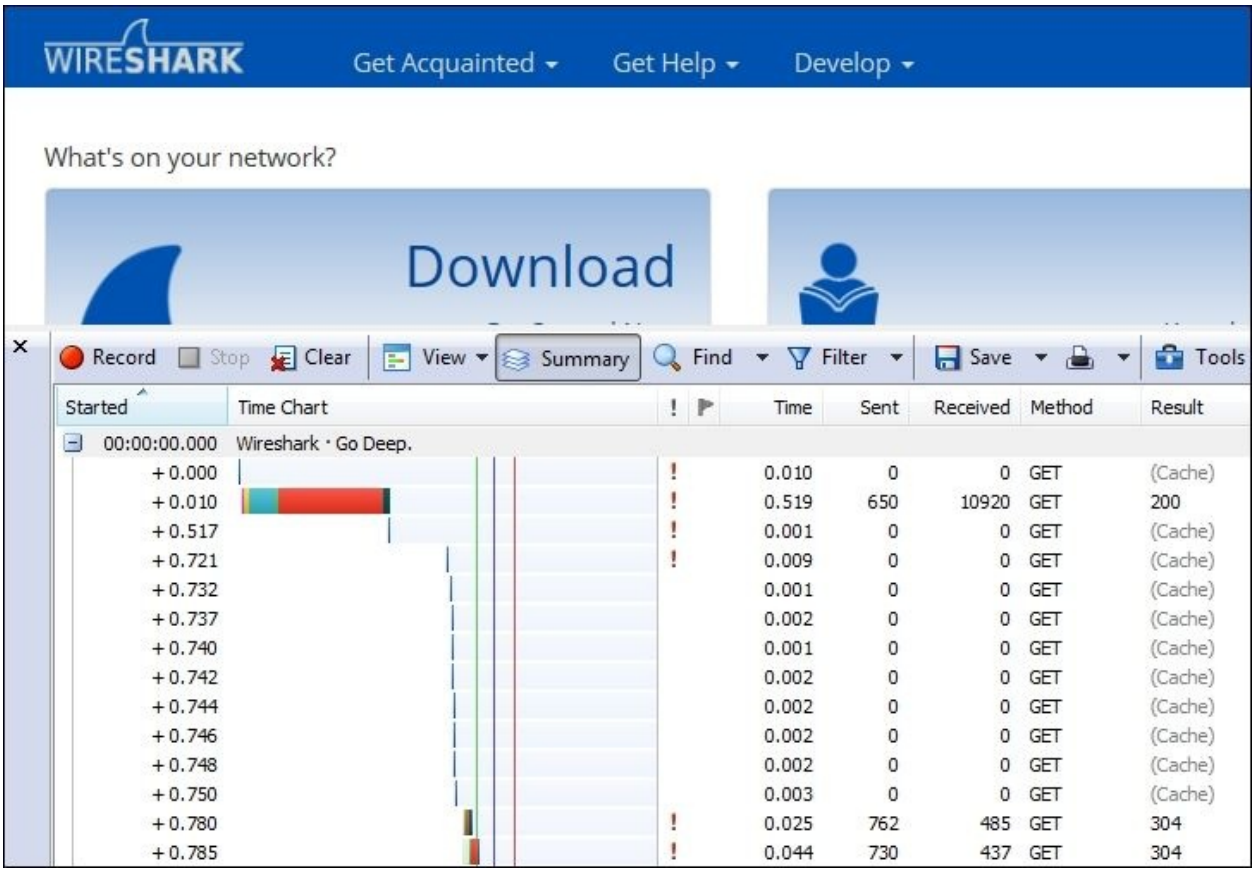
Other helpful tools

Wireshark is an extremely versatile and useful tool. However, there are some things it doesn't do easily or at all, so we'll discuss a few other tools you may want to include in your analysis toolset.

HttpWatch

HttpWatch is a packet-based performance analysis utility that integrates with Internet Explorer and Firefox browsers to view a graphical depiction and statistical values from HTTP interactions between the browser and websites. This kind of utility makes it easy to discover and measure from the user's perspective when significant delays are occurring and the source of those delays.

The following screenshot shows the HttpWatch visual and numerical analysis by loading the www.wireshark.org home page:

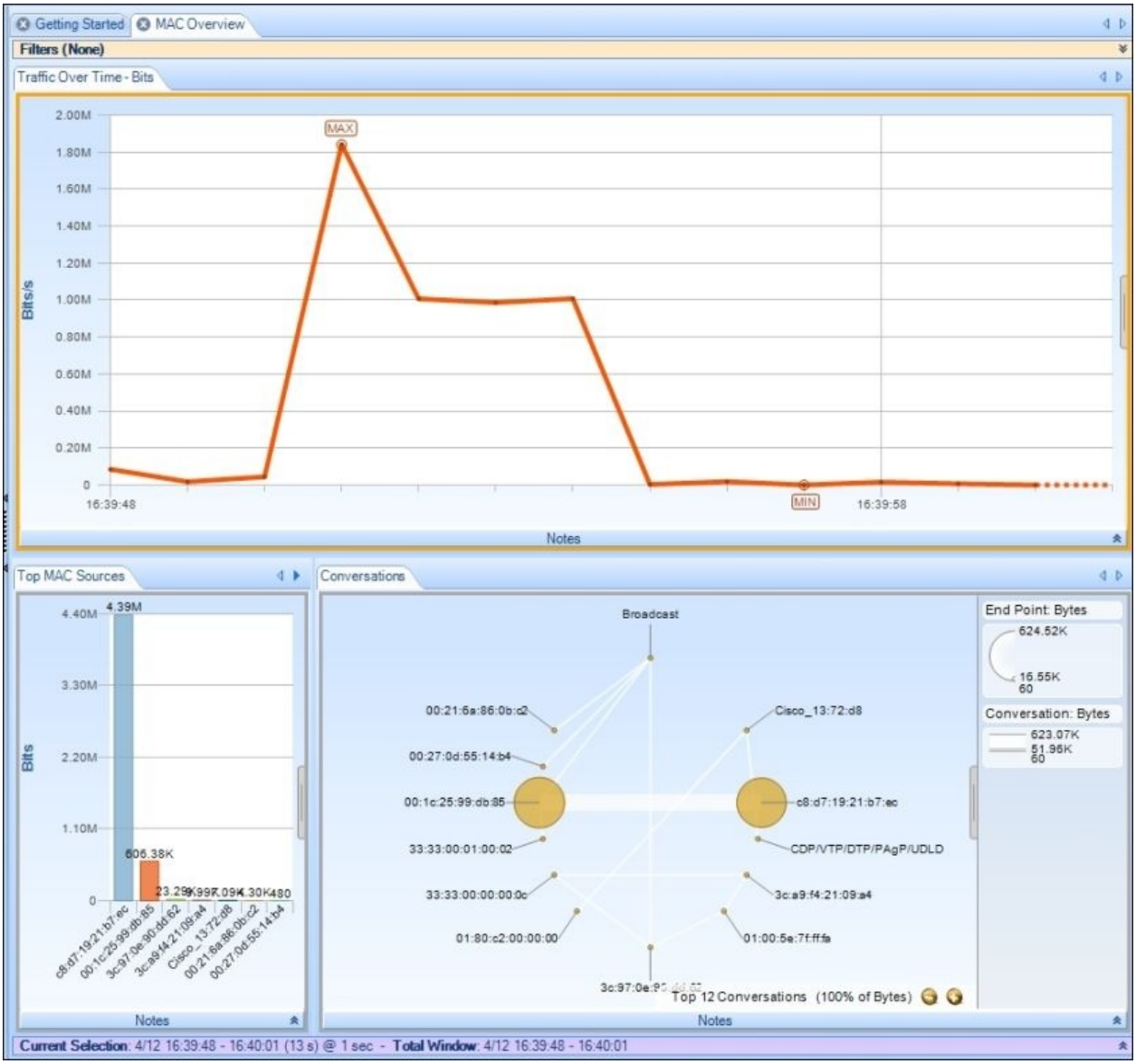


You can get more information about HttpWatch from http://www.httpwatch.com/. Also, a similar performance analysis utility is Fiddler, which can be found at http://www.telerik.com/fiddler.

SteelCentral Packet Analyzer Personal Edition

SteelCentral Packet Analyzer (previously known as Cascade Pilot) is available in Standard and Personal Edition versions. Unlike Wireshark, this utility is able to open and analyze multigigabyte trace files; you can quickly isolate a conversation of interest, right-click on it, and save that conversation in a separate packet trace file or launch Wireshark directly and pass that conversation to it from the same menu.

In addition, the utility offers a variety of network analysis screens called **Views** that provide graphical displays and reports on a wide range of performance perspectives. The following screenshot illustrates a set of **MAC Overview** Views:



You can get more information on the SteelCentral Packet Analyzer products at http://www.riverbed.com/products/performance-management-control/network-performance-management/packet-analysis.html.

AirPcap adapters

If you are using Wireshark to analyze wireless networks, you will need a wireless adapter that provides the ability to see all of the available channels and provides a Radiotap Header, which offers additional information for each frame such as radio channel and signal/noise strengths.

The prevalent wireless adaptor for use with Wireshark or SteelCentral Packet Analyzer on Windows platforms is the **Riverbed AirPcap adapter**, which is available from the Riverbed website. The AirPcap adapter plugs into a USB port and includes drivers to integrate with Wireshark and provide the Radiotap Header information. There are several product models that offer increasing coverage of the various WLAN bands; AirPcap Nx offers the widest coverage. The following image depicts two of the available adapters:

You can get more information on the Riverbed AirPcap adapters at http://www.riverbed.com/products/performance-management-control/network-performance-management/wireless-packet-capture.html.

Summary

The topics covered in this chapter included several of Wireshark's command-line utilities to capture packets and edit and merge packet trace files, as well as several useful tools to compliment your analysis toolset.

This is the final chapter of this book on Wireshark. I hope you enjoyed reading it, and mostly, I hope you use it as a foundation to become a Wireshark expert!

Part 2. Module 2

Network Analysis Using Wireshark Cookbook

Over 80 recipes to analyze and troubleshoot network problems using Wireshark

Chapter 1. Introducing Wireshark

In this chapter you will learn:

- Locating Wireshark
- Starting the capture of data
- Configuring the start window
- Using time values and summaries
- Configuring coloring rules and navigation techniques
- Saving, printing, and exporting data
- Configuring the user interface in the **Preferences** menu
- Configuring protocols preferences

Introduction

In this chapter, we will cover the basic tasks related to Wireshark. In the *Preface* of this book, we discussed network troubleshooting and the various tools that can help us in the process. After reaching the conclusion that we need to use the Wireshark protocol analyzer, it's time to locate it for testing in the network, to configure it with basic configurations, and to adapt it to be user friendly.

While setting Wireshark for basic data capture is considered to be very simple and intuitive, there are many options that we can use in special cases; for example, when we capture data continuously over a connection and we want to split the capture file into small files, when we want to see names of the devices participating in the connection and not only IP addresses, and so on. In this chapter we will learn how to configure Wireshark for these special cases.

Another important issue is where to locate Wireshark to capture data. Will it be before a firewall or after it? On which side of the router should we connect it? On the LAN side or on the WAN side? What should we expect to receive in each one of them? All these issues and more will be covered in the *Locating Wireshark* recipe in this chapter, along with recommendations on how to do it.

Another important issue that will be covered in this chapter is how to configure time values, that is, how you would like Wireshark to present the arrival time of captured packets. This is significantly important when we capture data of time-sensitive applications, when it is important to see the timing of packets inside a TCP connection or a UDP flow.

The next recipe will be on file manipulations, that is, how to save the captured data, whether we want to save the whole of it or part of it, save only filtered data, export that data into various formats, merge files (for example, when you want to merge captured files on two different router interfaces), and so on.

One more issue that will be discussed in this chapter is how to configure coloring rules. That is, how to configure Wireshark to present different packets and protocols in different colors. While Wireshark by default has its coloring scheme, we might want to configure it for special cases, for example, to give a special color to a specific protocol that we monitor or to a specific error or event

that we expect. The *Configuring coloring rules and navigation techniques* recipe discusses these issues.

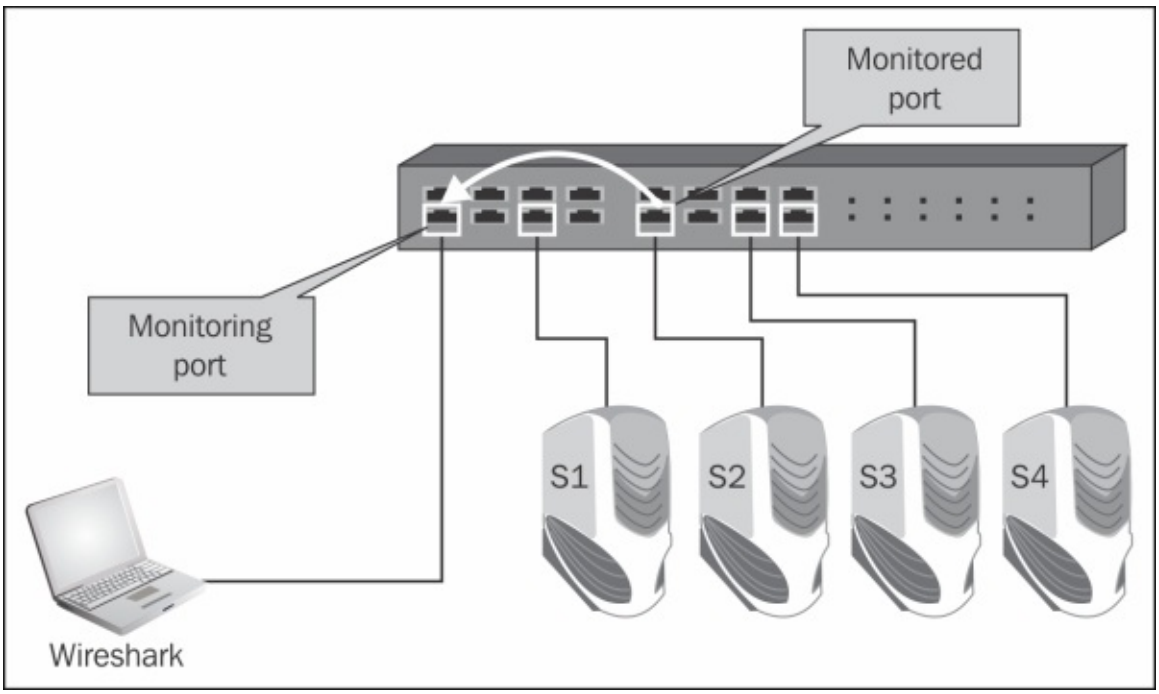
The last two recipes of the chapter will cover the configuration of the Wireshark preferences. These recipes discuss how to configure the user interface, that is, to configure the Wireshark windows, the columns and what to see in each one of them, text formats, and so on, along with specific protocol configurations; for example, which TCP ports should be resolved by default as a proxy service, whether or not to validate a protocol checksum, whether or not to calculate TCP timestamps, how to decode fields in the protocol header, and so on.

Locating Wireshark

After understanding the problem and deciding to use Wireshark, the first step would be to decide where to locate it. For this purpose, we need to have a precise network diagram (at least the part of the network that is relevant to our test).

The principle is to locate the device that you want to monitor, connect your laptop to the same switch that it is connected to, and configure a port mirror or monitor to the monitored device. This operation enables you to see all traffic coming in and out of the monitored device.

You can monitor a LAN port, WAN port, server or router port, or any other device connected to the network.



In the preceding diagram, the Wireshark software (installed on the PC on the left) and the port mirror, also called port monitor (configured on the switch in the direction as in the diagram), will monitor all the traffic coming in and out of server S2. Of course, we can also install Wireshark directly on the server itself, and by doing so, we will be able to watch the traffic directly on the server.

Some LAN switch vendors also enable other features such as:

- **Monitoring a whole VLAN**: We can monitor a server's VLAN, Telephony VLAN, and so on. In this case you will see all the traffic on a specific VLAN.
- **Monitoring several ports to a single analyzer**: We can monitor traffic on servers **S1** and **S2** together.
- **Filtering**: Filtering means choosing and accordingly configuring whether to monitor incoming traffic, outgoing traffic, or both.

Getting ready

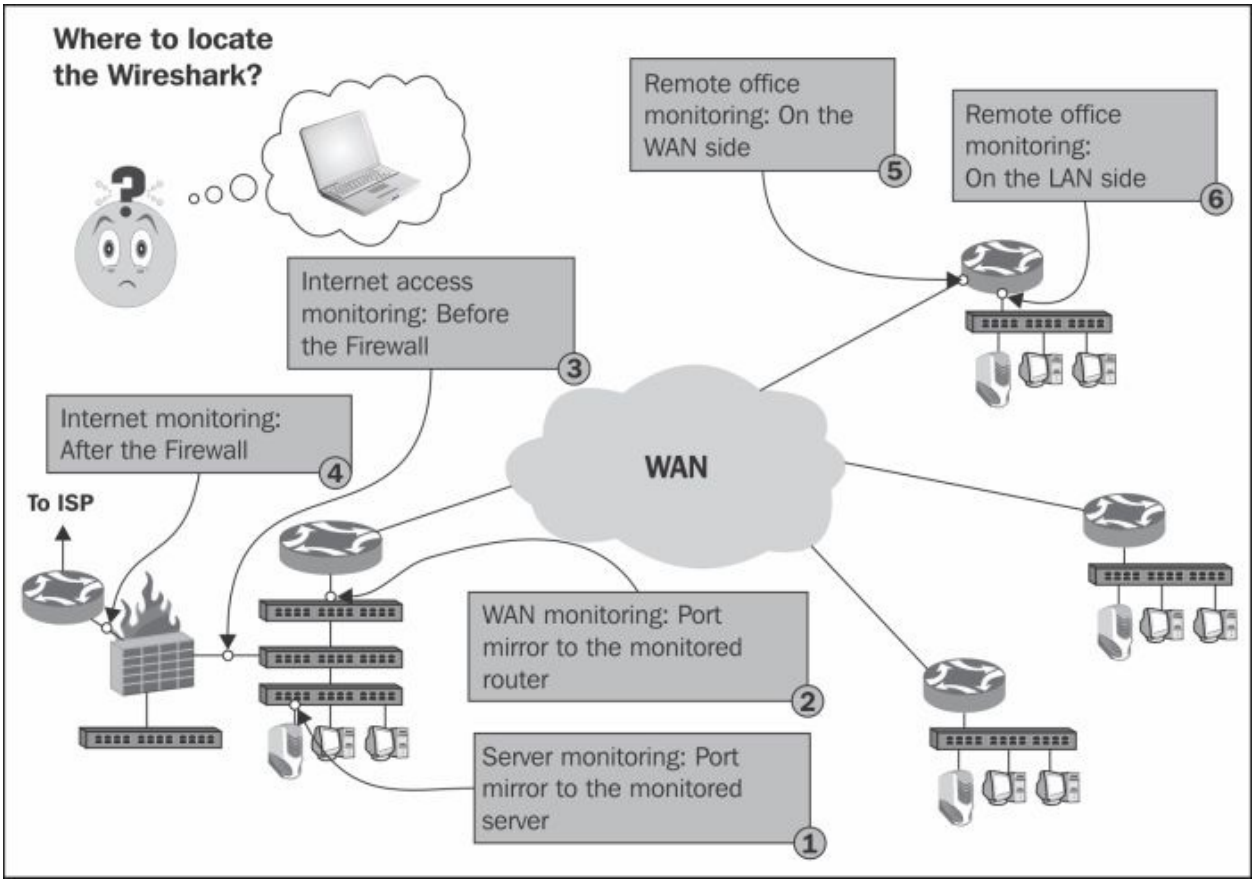
To start working with Wireshark, go to the the Wireshark website, and download the latest version of the tool.

An updated version of Wireshark can be found on the website at http://www.wireshark.org/, under the **Download** heading. Download the latest Wireshark stable release that is available at http://www.wireshark.org/download.html.

Each Wireshark Windows package comes with the latest stable release of WinPcap, which is required for live packet capture. The WinPcap driver is a Windows version of the UNIX Libpcap library for traffic capture.

How to do it...

Let's take a look at the typical network architecture and network devices, how they work, how to configure them when required, and where to locate Wireshark.



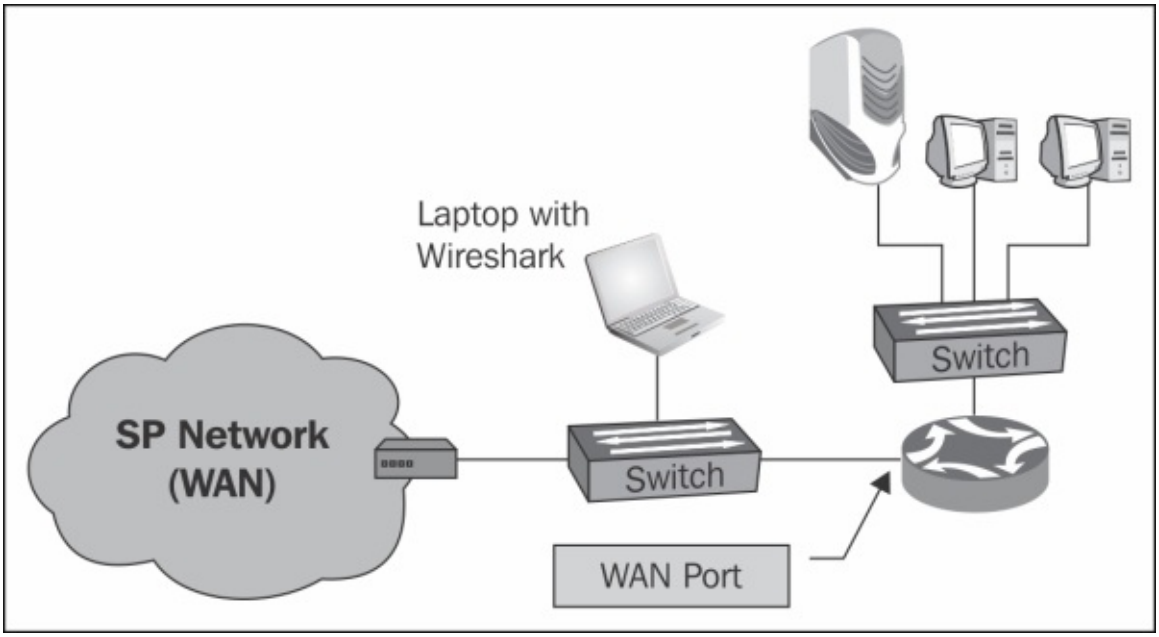
Let's have a look at the simple and common network architecture in the preceding diagram.

Monitoring a server

This will be one of the most common requirements that we will have. It can be done by either configuring the port monitor to the server (numbered as **1** in the preceding diagram), or installing Wireshark on the server itself.

Monitoring a router

In order to monitor a router, we can monitor a LAN port (numbered as **2** and **6** in the preceding diagram), or a WAN port (numbered as **5** in the preceding diagram). To monitor a LAN port is easy—simply configure the port monitor to the port you wish to monitor. In order to monitor a WAN port, you can connect a switch between the router port and the **Service Provider** (**SP**) network, and configure the port monitor on this switch, as in the following illustration.



Connecting a switch between the router and the service provider is an operation that breaks the connection; however, when you prepare for it, it should take less than a minute.

When monitoring a router, don't forget—not all packets coming in to a router will be forwarded. Some packets can be lost, dropped on the router buffers, or routed back on the same port that they came in from.

Two additional devices that you can use are TAPs and Hubs.

- **TAPs**: Instead of connecting a switch on the link you wish to monitor, you can connect a device called **Test Access Point** (**TAP**), which is a simple three-port device that, in this case, will play the same role as that of the switch. The advantage of a TAP over a switch is its simplicity and price. TAPs also forward errors that can be monitored on Wireshark, unlike a

LAN switch that drops them. Switches, on the other hand, are much more expensive, take a few minutes to configure, but provide you with additional monitoring capabilities, for example, **Simple Network Management Protocol** (**SNMP**). When you troubleshoot a network, it is better to have an available managed LAN switch, even a simple one.

- **Hubs**: You can simply connect a hub in parallel to the link you want to monitor, and since a hub is a half-duplex device, every packet sent between the router and the SP device will be watched on your Wireshark. The biggest con of this method is that the hub itself slows the traffic, and it therefore influences the test. In many cases you also want to monitor 1 Gbps ports, and since there is no hub available for this, you will have to reduce the speed to 100 Mbps, which again will influence the traffic. Therefore, hubs are not commonly used.

Monitoring a firewall

When monitoring a firewall, it differs depending on whether you monitor the internal port (numbered **3** in the diagram) or the external port (numbered **4** in the diagram). On the internal port you will see all the internal addresses and all traffic initiated by the users working in the internal network, while on the external port you will see the external addresses that we go out with (translated by NAT from the internal addresses); you will not see requests from the internal network that were blocked by the firewall. If someone is attacking the firewall from the Internet, you will see it (hopefully) only on the external port.

How it works...

To understand how the port monitor works, it is first important to understand the way that a LAN switch works. A LAN switch forwards packets in the following way:

1. The LAN switch continuously learns about the MAC addresses of the devices connected to it.
2. Now, if a packet is sent to a destination MAC, it will be forwarded only to the physical port that the switch knows this MAC address is coming from.
3. If a broadcast is sent, it will be forwarded to all the ports of the switch.
4. If a multicast is sent and **Cisco Group Management Protocol** (**CGMP**) or **Internet Group Management Protocol** (**IGMP**) is disabled, it will be forwarded to all the ports of the switch (CGMP and IGMP are protocols that enable multicast packets to be forwarded only to devices on a specific multicast group).
5. If a packet is sent to a MAC address that the switch does not know about (which is a very rare case), it will be forwarded to all the ports of the switch.

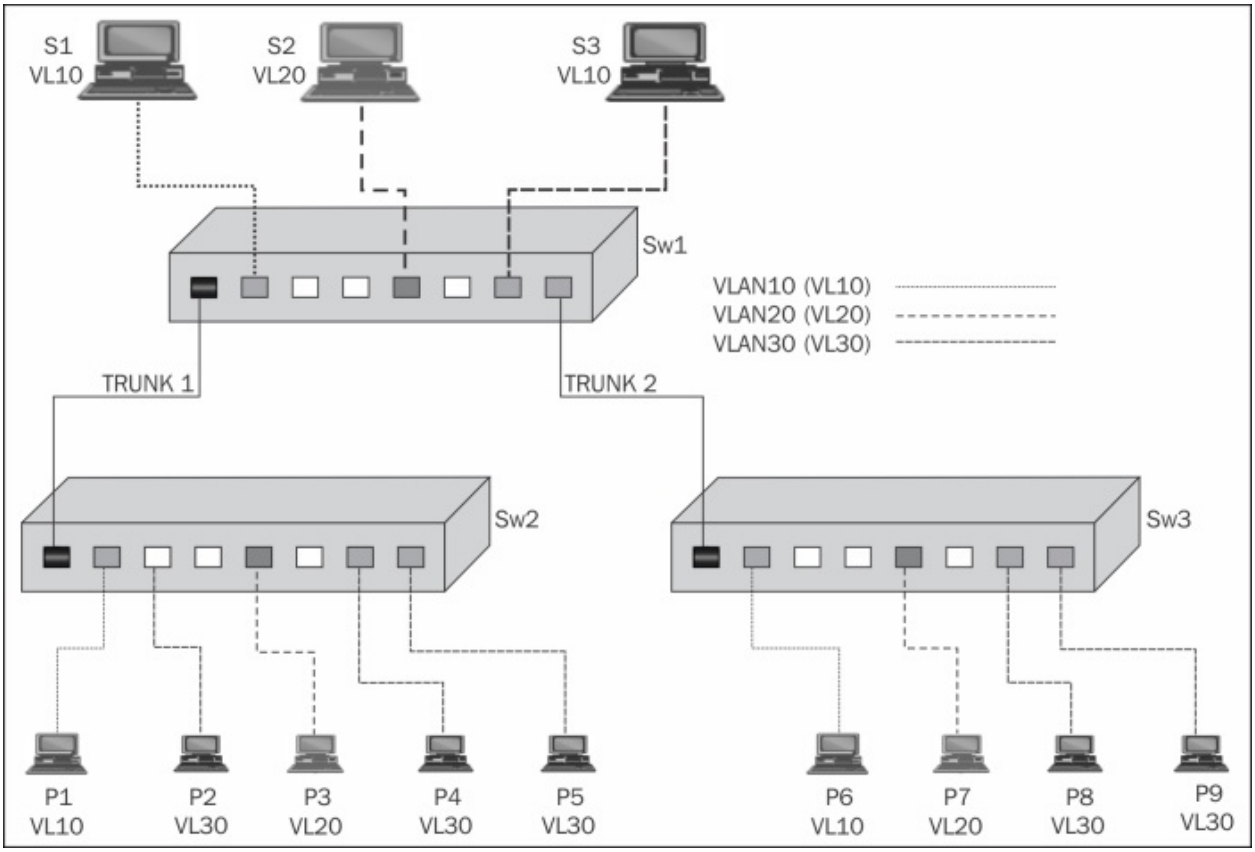
Therefore, when you configure a port monitor to a specific port, you will see all the traffic coming in and out of it. If you connect your laptop to the network, without configuring anything, you will see only the traffic coming in and out of your laptop, along with broadcasts and multicasts from the network.

There's more...

When capturing data, there are some tricky scenarios that you should be aware of.

One such scenario is monitoring a VLAN. When monitoring a VLAN, you should be aware of several important issues. The first issue is that even when you monitor a VLAN, the packet must physically be transferred through the switch you are connected to, in order to see it. If, for example, you monitor VLAN-10 that is configured across the network, and you are connected to your floor switch, you will not see the traffic that goes from other switches to the servers on the central switch.

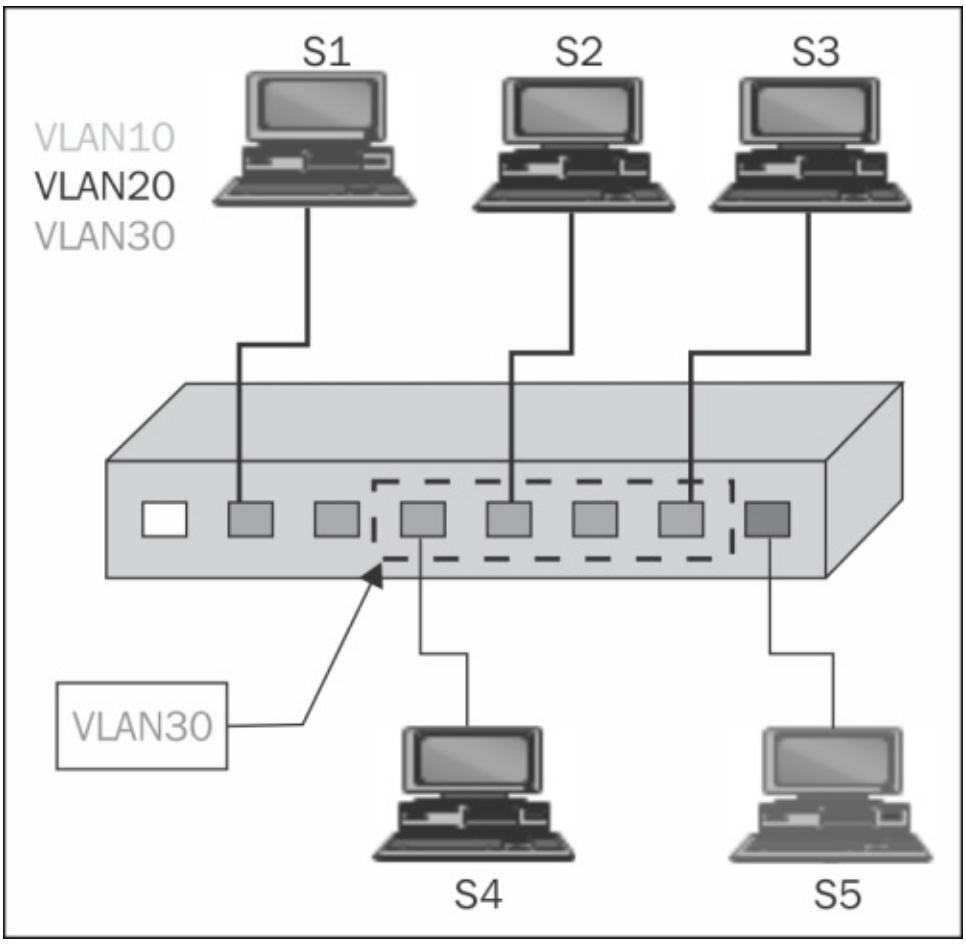
This is because when building a network, the users are usually connected to floor switches in single or multiple locations in the floor, that are connected to the building central switch (or two redundant switches). For monitoring all traffic on a VLAN, you have to connect to a switch on which all traffic of the VLAN goes through, and this is usually the central switch.



In the preceding diagram, if you connect Wireshark to Switch SW2, and configure a monitor to VLAN30, you will see all the packets coming in and out of **P2**, **P4**, and **P5**, inside or outside the switch. You will not see packets transferred between devices on **SW3** and **SW1**, or packets between **SW1** and **SW3**.

Another issue when monitoring a VLAN is that you might see duplicate packets. This is because when you monitor a VLAN, and packets are going in and out of the VLAN, you will see the same packet when it is comes in, and then when it goes out of the VLAN.

You can see the reason in the following illustration. When, for example, **S4** sends a packet to **S2**, and you configure the port mirror to VLAN30, you will see the packet once when sent from **S4** passing through the switch and entering the VLAN30, and then when leaving VLAN30 and coming to **S2**.


S1
S2
S3
VLAN10
VLAN20
VLAN30
VLAN30
S4
S5

See also

For information on how to configure the port mirror, refer to the vendor's instructions. It can be called **port monitor**, **port mirror**, or **SPAN** (**Switched Port Analyzer** from Cisco).

There are also advanced features such as remote monitoring (monitoring a port that is not directly connected to your switch), advanced filtering (such as filtering specific MAC addresses), and so on. There are also advanced switches that have capture and analysis capabilities on the switch itself. It is also possible to monitor virtual ports (for example, LAG or Ether channel groups). For all cases, refer to the vendor's specifications.

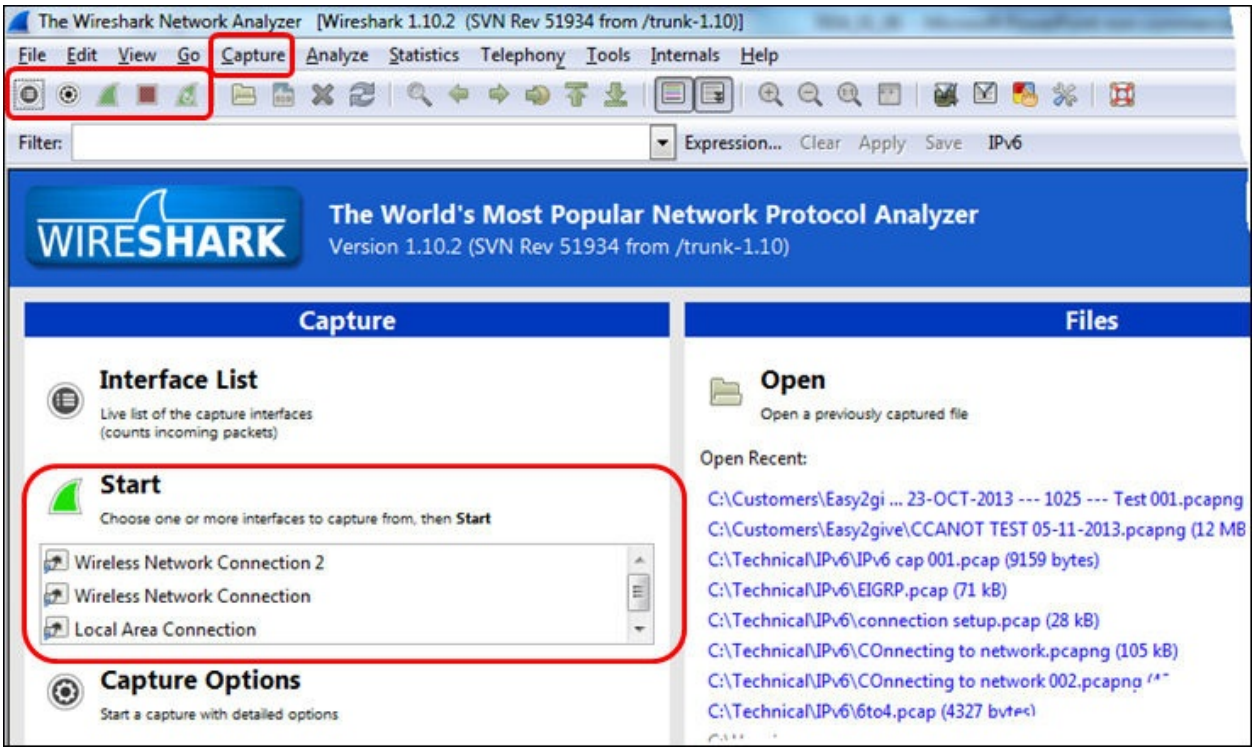
Starting the capture of data

In this recipe, we will learn how to start capturing data, and what we will get in various capture scenarios, after we have located Wireshark in the network.

Getting ready

After you install Wireshark on your computer, the only thing to do will be to start the analyzer from the desktop, program files, or the quick start bar.

When you do so, the following window will be opened (Version 1.10.2):



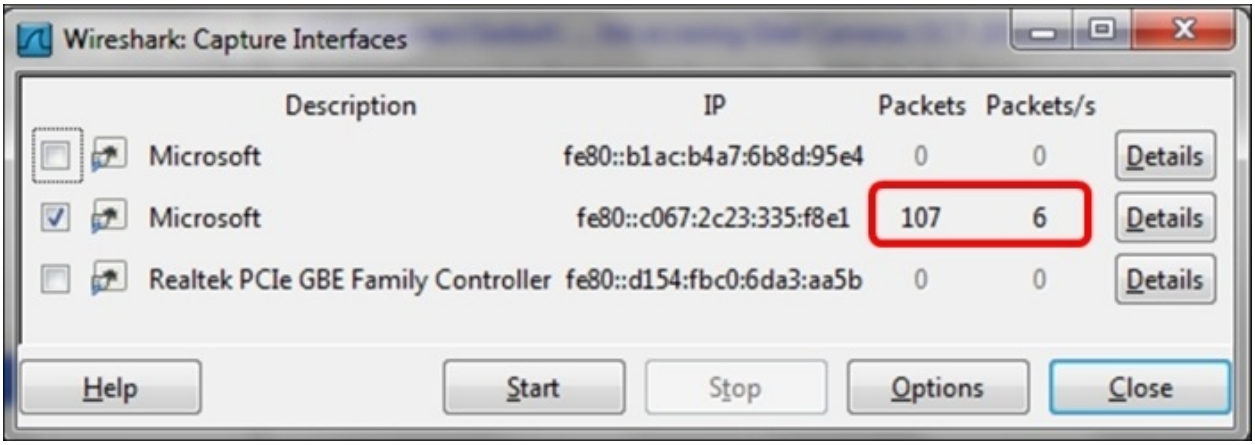
How to do it...

You can start the capture from the upper bar **Capture** menu, or from the quick-launch bar with the capture symbol, or from the center-left capture window on the Wireshark main screen. There are options that you can choose from.

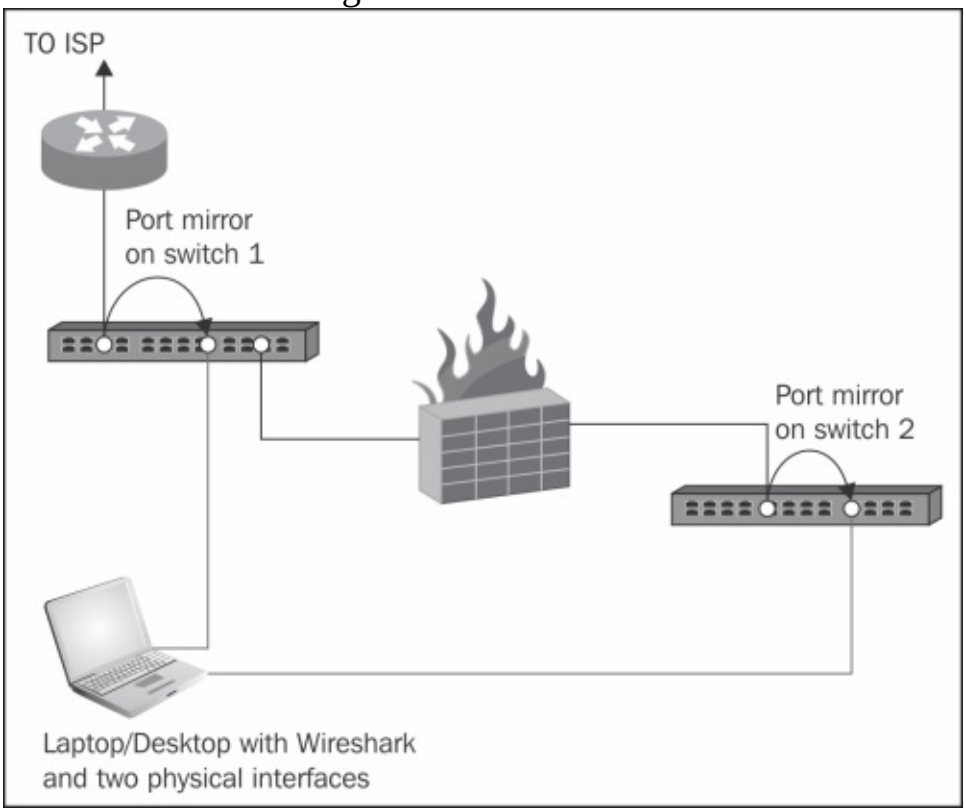
How to choose the interface to start the capture

If you simply click on the green icon, third to the right, in Wireshark and start the capture, Wireshark will start the capture on the default interface as configured in the software (explained later in the chapter in the recipe *Configuring the user interface in the Preferences menu*). In order to choose the interface you want to capture on, click on the **List the available capture interfaces** symbol, and the **Wireshark Capture Interfaces** window will open.

The best way to see which interface is active is simply to look at the right of the window of the interface on which you see the traffic running. There you will see the number of total **Packets** seen by Wireshark, and the number of **Packets/sec** in each interface.

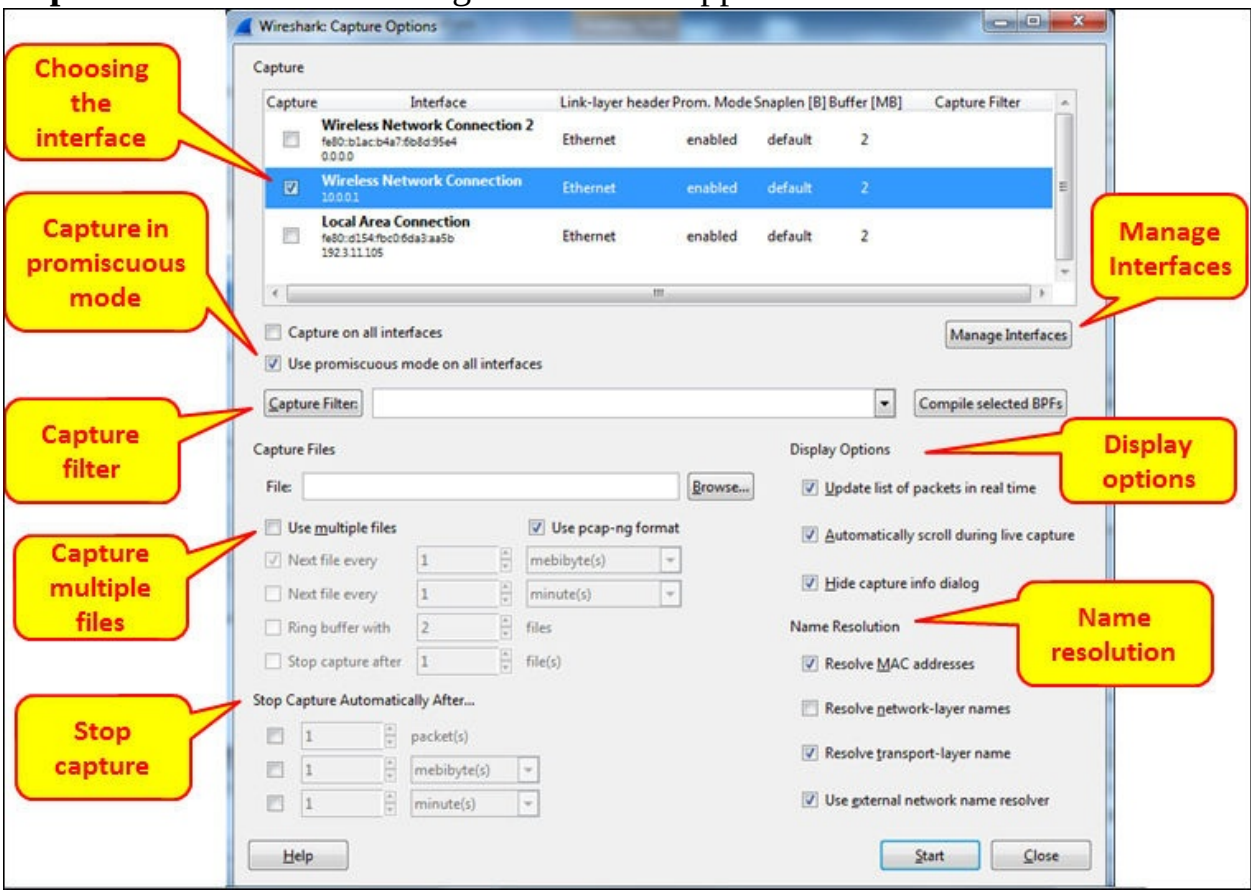


In Wireshark Version 1.10.2 and above, you can choose one or more interfaces for the capture. This can be helpful in many cases; for example, when you have multiple physical NICs, you can monitor the port on two different servers, two ports of a router, or other multiple ports at the same time. A typical configuration is seen in the following screenshot:



How to configure the interface you capture data from

To configure the interface you capture data from, choose **Options** from the **Capture** menu. The following window will appear:



In the preceding window you can configure the following parameters:

1. On the upper side of the window, choose the interface you want to capture the data from.
2. On the left side of the window, you have the checkbox **Use promiscuous mode on all interfaces**. When checked, Wireshark will capture all the packets that the computer receives. Unchecking it will capture only packets intended for the computer.
3. In some cases, when this checkbox is checked, Wireshark will not capture data in the wireless interface; so if you start capturing data on the wireless

interface and see nothing, uncheck it.

4. On the mid-left area of the window, you have the **Capture Files** field. You can write a file name here, and Wireshark will save the captured file under this name, with extensions 0001, 0002, and so on under the path you specify. This feature is extremely important when capturing a large amount of data; for example, when capturing data over a heavily-loaded interface, or over a long period of time. You can tell the software to open a new file after a specific interval of time, file size, or number of packets.
5. On the bottom left of the window, you have the area marked as **Stop Capture Automatically** in the preceding screenshot. In this area, you can tell the software to stop capturing data after a specific interval of time, file size, or number of packets.
6. On the mid-right area of the window, you can change the **Display** option and select the checkboxes **Update list of packets in real time**, **Automatically scroll during live capture**, and **Hide capture info dialog**, which close the annoying capture window (a pop up that appears the moment you start capture). In most of the cases you don't have to change anything here.
7. On the bottom right of the window, you configure the resolving options for MAC addresses, IP DNS names, and TCP/UDP port numbers. The last checkbox, **Use external network name resolver**, uses the system's configured name resolver (in most of the cases, DNS), to resolve network names.

How it works...

Here the answer is very simple. When Wireshark is connected to a wired or wireless network, there is a software driver that is located between the physical or wireless interface and the capture engine. In Windows we have the **WinPcap** driver, in Unix platforms the **Libpcap** driver, and for wireless interfaces we have the **AirPcap** driver.

There's more...

In cases where the capture time is important, and you wish to capture data on one interface or more, and be time-synchronized with the server you are monitoring, you can use **Network Time Protocol** (**NTP**) to synchronize your Wireshark and the monitored servers with a central time source.

This is important in cases when you want to go through the Wireshark capture file in parallel to a server logfile, and look for events that are shown on both. For example, if you see retransmissions in the capture file at the same time as a server or application error on the monitored server, you will know that the retransmissions are because of server errors and not because of the network.

The Wireshark software takes its time from the OS clock (Windows, Linux, and so on) For configuring the OS to work with a time server, go to the relevant manuals of the operating system that you work with.

In Microsoft Windows7, configure it as follows:

1. Go the **Control Panel**.
2. Choose **Clock, Language, and Region**.
3. Under **Date and Time**, Choose **Set the time and date** and change to the **Internet time** tab.
4. Click on the **Change Settings** button.
5. Change the server name or the IP address.

Note

In Microsoft Windows7 and later versions, there is a default setting for the time server. As long as all devices are tuned to it, you can use it as any other time server.

NTP is a network protocol used for time synchronization. When you configure your network devices (routers, switches, FWs, and so on) and servers to the same time source, they will be time synchronized to this source. The accuracy of the synchronization depends on the accuracy of the time server that is measured in levels or stratums. The higher the level, the more accurate it will be. Level 1 is the highest. Usually you will have levels 2 to 4.

NTP was first standardized in RFC 1059 (NTPv1), and then in RFC 1119 (NTPv2); the common versions in the last years are NTPv3 (RFC1305) and NTPv4 (RFC 5905).

You can get a list of NTP servers on various web sites, among them http://support.ntp.org/bin/view/Servers/StratumOneTimeServers and

http://wpollock.com/AUnix2/NTPstratum1PublicServers.htm.

See also

You can get more information about Pcap drivers at:

- For WinPcap visit: http://www.winpcap.org
- For Libpcap visit: http://www.tcpdump.org

Configuring the start window

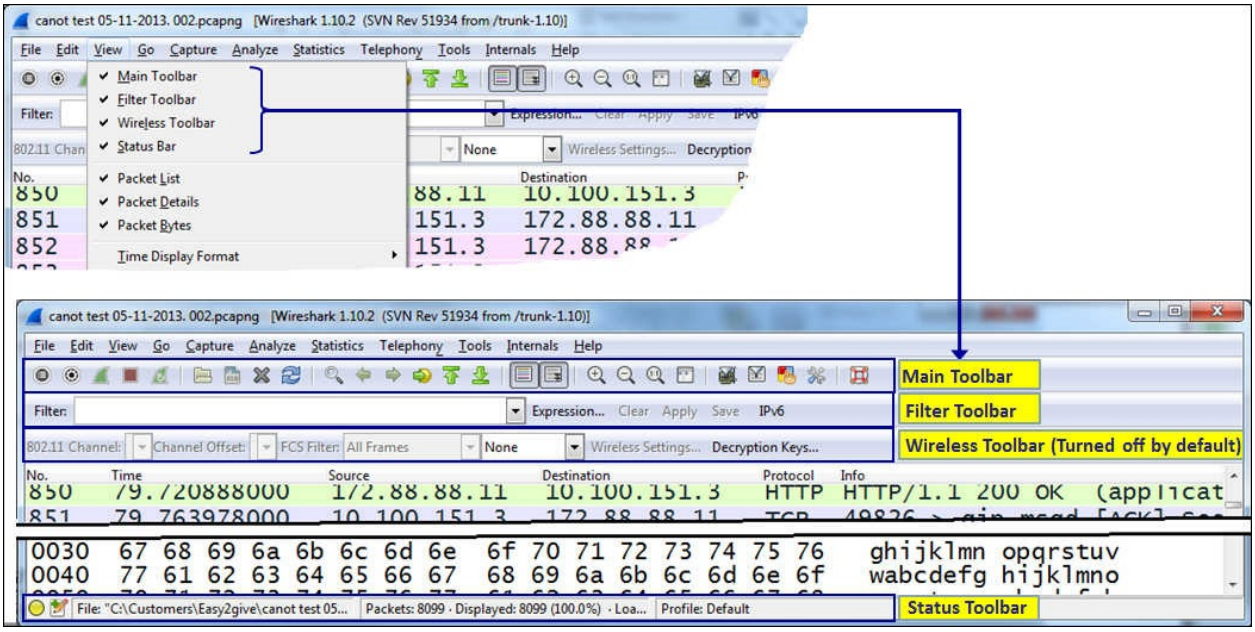
In this recipe we will see some basic configurations for the start window. We will talk about configuring the main window, file formats, and viewing options.

Getting ready

Start Wireshark, and you will get the start window. There are several parameters you can change here in order to adapt the capture window to meet your requirements:

- Toolbars configuration
- Main window configuration
- Time format configuration
- Name resolution
- Colorize packet list
- Auto scroll in live capture
- Zoom
- Columns configuration
- Coloring rules

First, let's have a look at the toolbars that are used by the software:



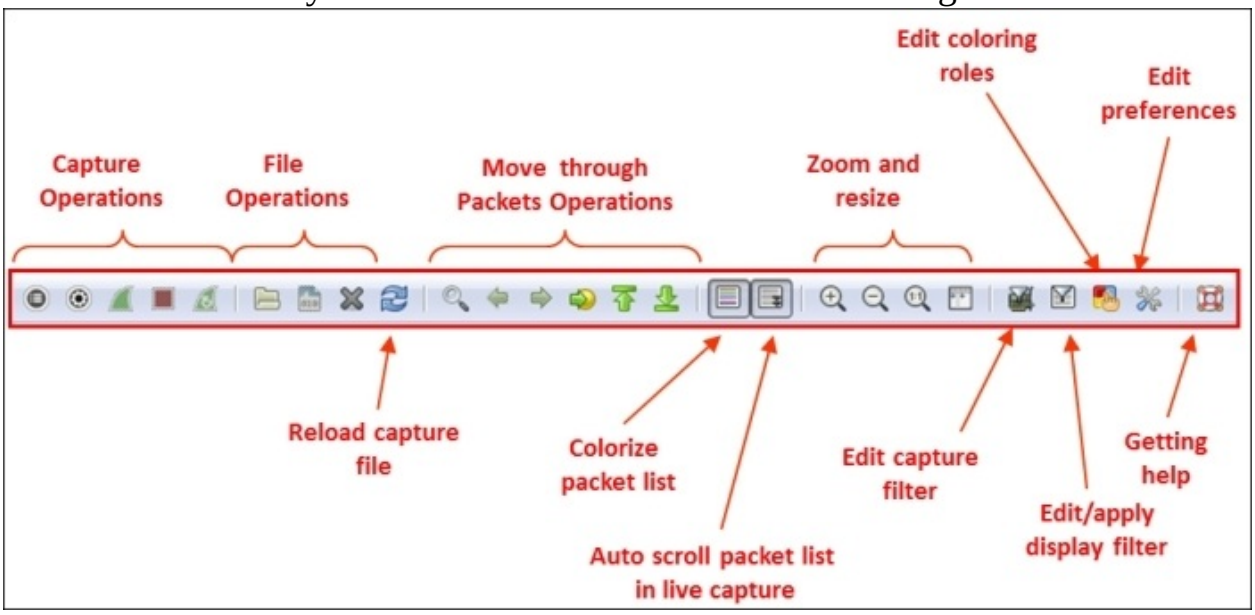
For operations with the other toolbars as follows, which are covered in the coming subsections in this recipe:

- Main Toolbar

- Display Filter Toolbar
- Status Bar

Main Toolbar

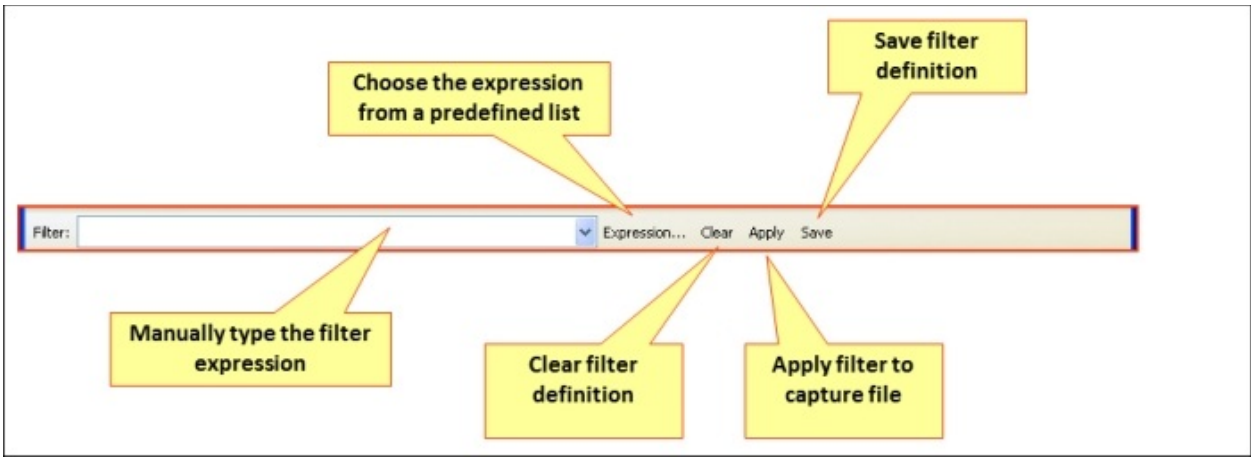
In the main toolbar you have the icons shown in the following screenshot:



The five leftmost symbols are for capture operations, then you have symbols for file operations, zoom and "go to packet" operations, colorize and auto-scroll, zoom and resize, filters, preferences, and help.

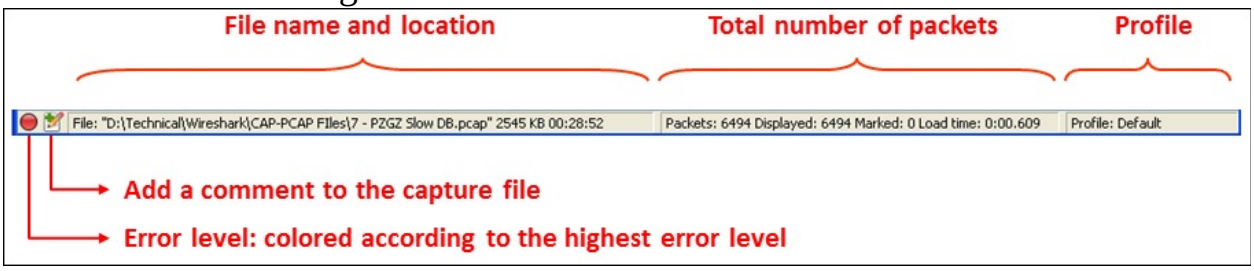
Display Filter Toolbar

In the filter toolbar, you have the following fields:



Status Bar

In the status bar on the lower side of the Wireshark window, you can see the data shown in the following screenshot:



In the preceding screenshot you can see the following:

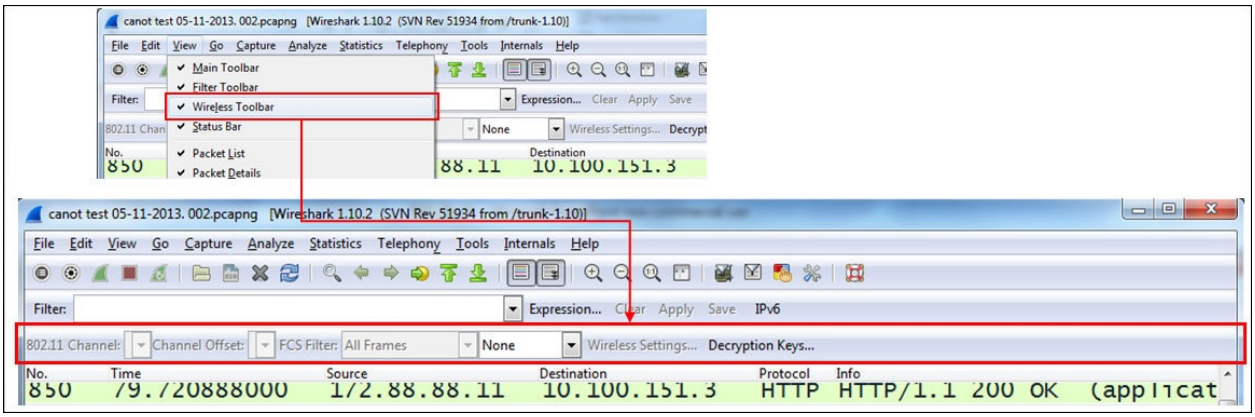
- Errors in the expert system
- The option to add a comment to the file
- The name of the captured file (during capture, it will show you a temporary name assigned by the software)
- Total number of captured packets, displayed packets (those which are actually displayed on the screen), and marked packets (those that you have marked).

How to do it...

In this part we will go step by step and configure the main menu.

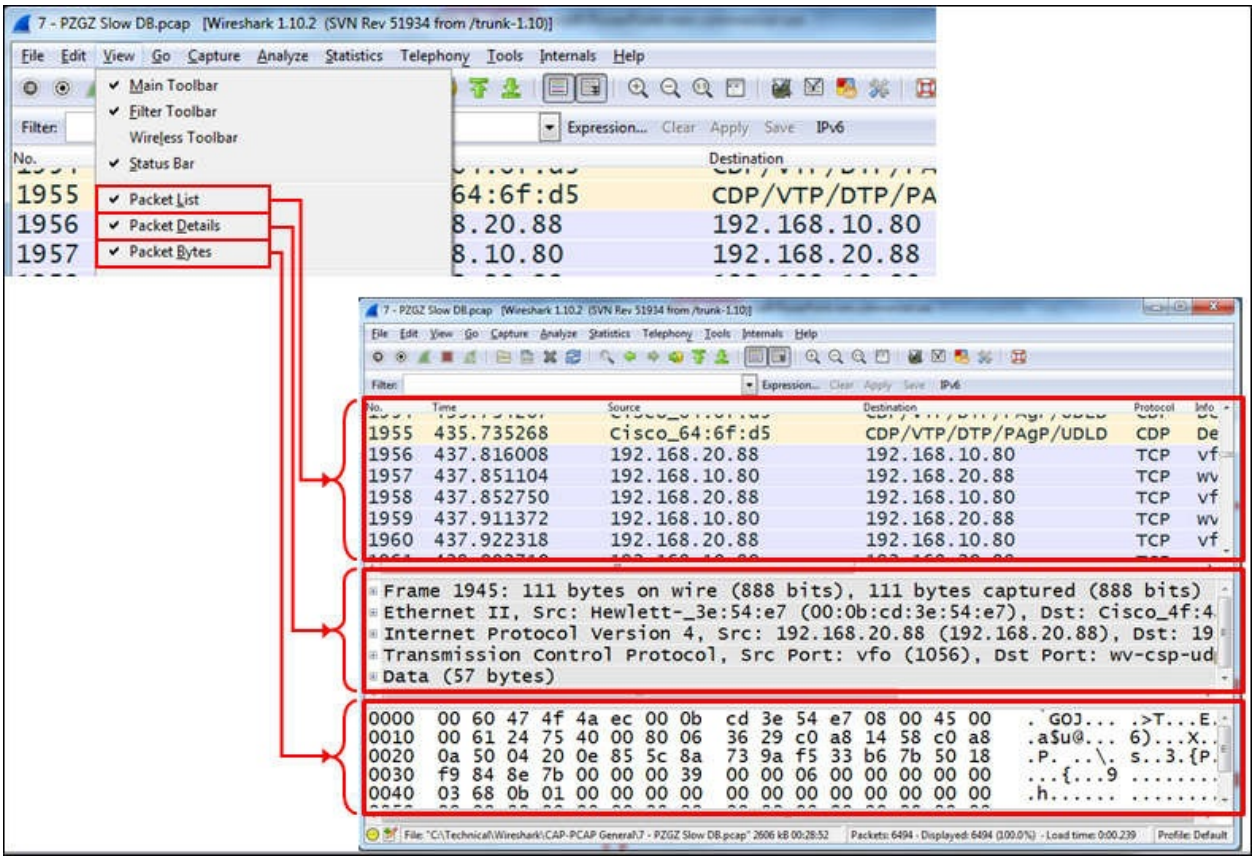
Configuring toolbars

Usually for regular packet capture, you don't have to change anything. This is different when you want to capture wireless data over the network (not only from your laptop); you will have to enable the wireless toolbar, and this will be done by clicking on it under the view menu, as shown in the following screenshot:



Configuring the main window

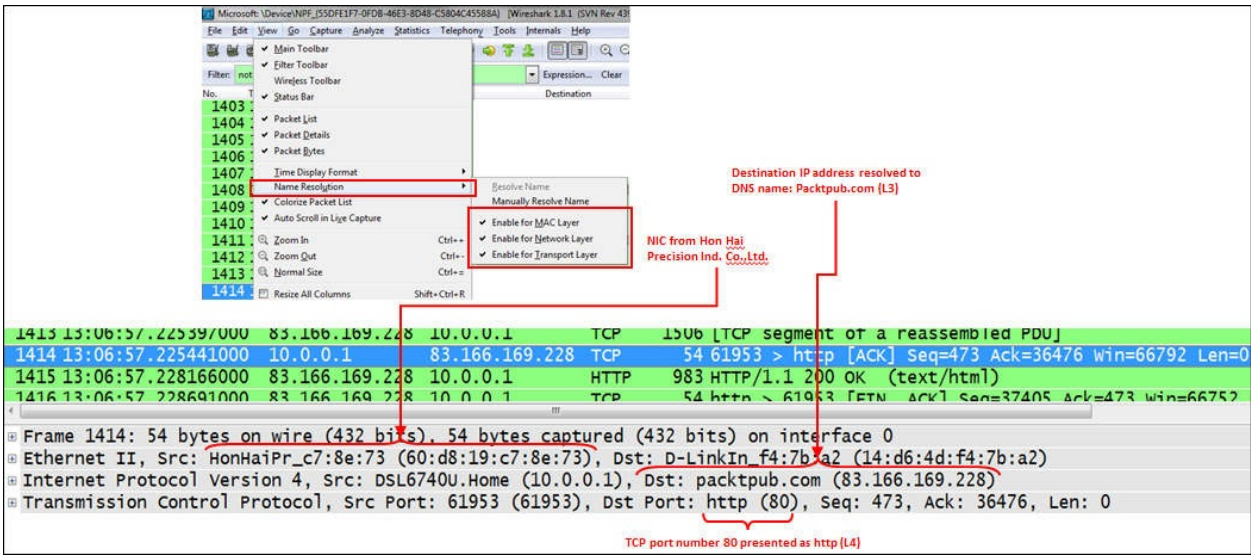
To configure the main menu for capturing, you can configure Wireshark to show the following windows:



In most of the cases you will not need to change anything here. In some cases, you can cancel the packet bytes when you don't need to see them, and you will get more "space" for the packet list and details.

Name Resolution

Name Resolution is the translation of layer 2 (MAC addresses), layer 3 (IP addresses), and layer 4 (Port numbers) into meaningful information.



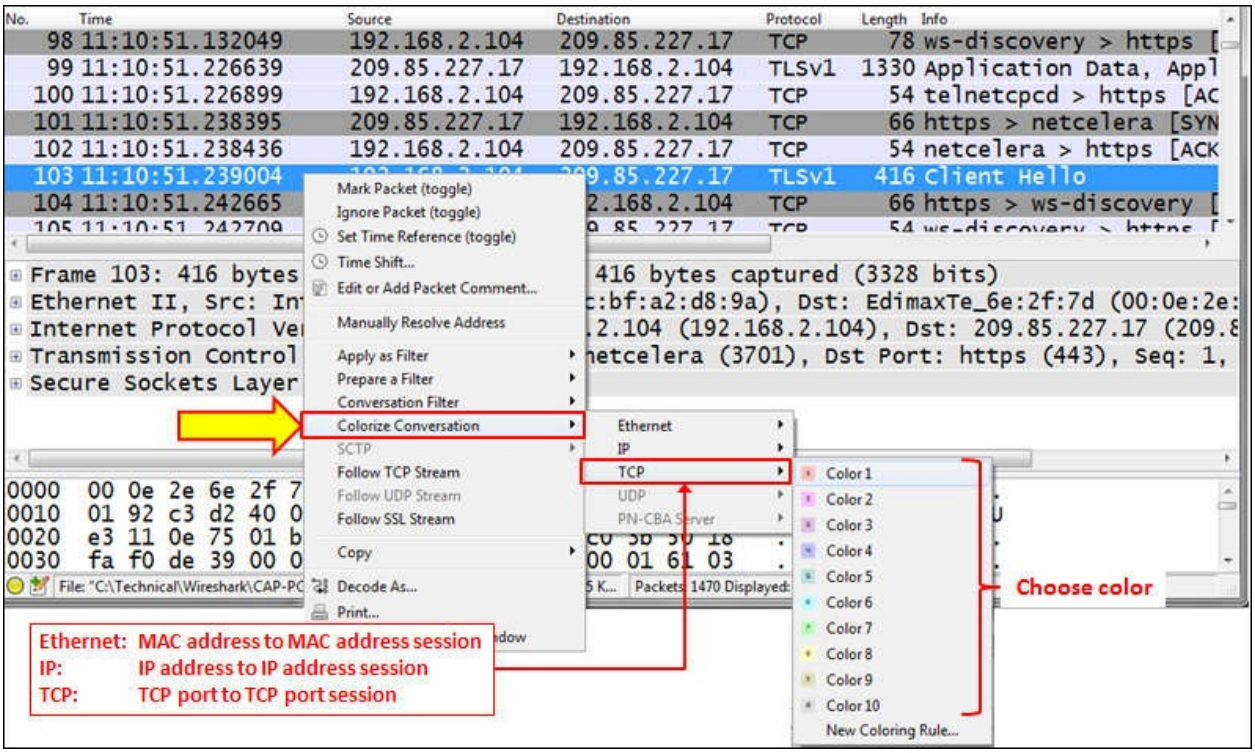
In the preceding screenshot, we see the MAC address **60:d8:19:c7:8e:73** (from **Hon Hai Precision Ind.**, used by Lenovo), the website (that is, Packtpub.com), and the HTTP port number (that is 80).

Colorizing the packet list

Usually you start a capture in order to establish a baseline profile of what normal traffic looks like on your network. During the capture, you look at the captured data and you might find a TCP connection, IP or Ethernet connectivity that are suspects, and you want to see them in another color.

To do so, right-click on the packet that belongs to the conversation you want to color, choose Ethernet, IP, or TCP/UDP (the appearance of TCP or UDP will depend on the packet), and choose the color for the conversation.

In the example you see that we want to color a **Transport Layer Security** (**TLS**) conversation.



For canceling the coloring rule:

1. Go to the **View** menu.
2. In the lower part of the menu, choose **Reset Coloring 1-10** or simply click on *Ctrl* + Space bar.

Auto scrolling in live capture

To configure Wireshark to auto-scroll the packets as it captures them, do the following:

1. Go to the **View** menu.
2. Mark the **Auto Scroll in Live Capture** item.
3. Zoom

For zooming in and out:

1. Go to the **View** menu.
2. Click on **Zoom In** or press *Ctrl* + + to zoom in.

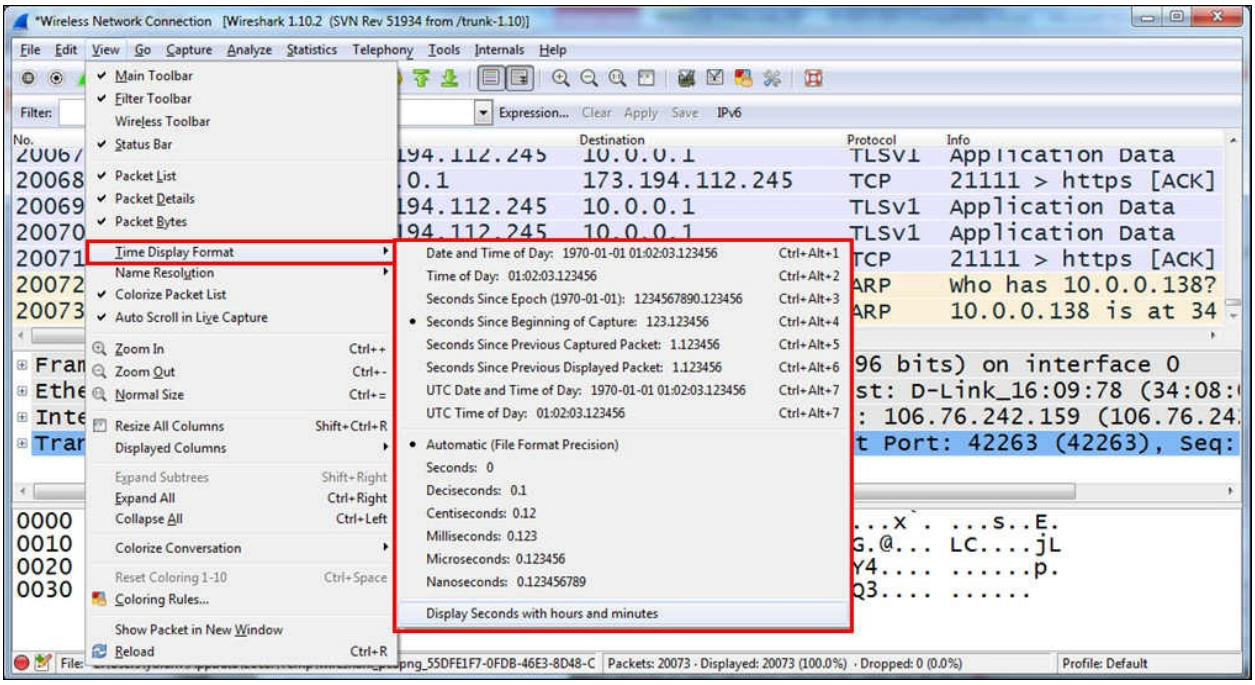
3. Click on **Zoom Out** or press *Ctrl* + - to zoom out.

Using time values and summaries

Time format configuration is about how the time column (second from the left on default configuration) will be presented. In some scenarios, there is a significant importance given to this; for example, in TCP connections that you want to see time intervals between packets, when you capture data from several sources and you want to see the exact time of every packet, and so on.

Getting ready

To configure the time format, go to the **View** menu, and under **Time Display Format** you will get the following window:



How to do it...

You can chose from the following options:

- **Date and Time of Day (the first two options)**: This will be good to configure when you troubleshoot a network with time-dependent events, for example, when you know about an event that happens at specific times, and you want to look at what happens on the network at the same time.
- **Seconds Since Epoch**: Time in seconds since January 1, 1970. Epoch is an arbitrary date chosen as a reference time for a system, and January 1, 1970 was chosen for Unix and Unix-like systems.
- **Seconds Since Beginning of Capture**: The default configuration.
- **Seconds Since Previous Captured Packet**: This is also a common feature that enables you to see time differences between packets. This can be useful when monitoring time-sensitive traffic (when time differences between packets is important), such as TCP connections, live video streaming, VoIP calls, and so on.
- **Seconds Since Previous Displayed Packet**: This is a useful feature that can be used when you configure a display filter, and only a selected part of the captured data is presented (for example, a TCP stream). In this case, you will see the time difference between packets that can be important in some applications.
- **UTC Date and Time of Day**: Provides us with relative UTC time.

The lower part of the submenu provides the format of the time display. Change it only if a more accurate measurement is required.

You can also use *Ctrl* + *Alt* + any numbered digit key on the keyboard for the various options.

How it works...

This is quite simple. Wireshark works on the system clock and presents the time as it is in the system. By default you see the time since the beginning of capture.

Configuring coloring rules and navigation techniques

Coloring rules define how Wireshark will color protocols and events in the captured data. Working with the coloring rules will help you a lot with network troubleshooting, since you are able to see different protocols in different colors, and you can also configure different colors for different events.

Coloring rules enable you to configure new coloring rules according to various filters. It will help you to configure different coloring schemes for different scenarios and save them in different profiles. In this way you can configure coloring rules for resolving TCP issues, rules for resolving Sip and Telephony problems, and so on.

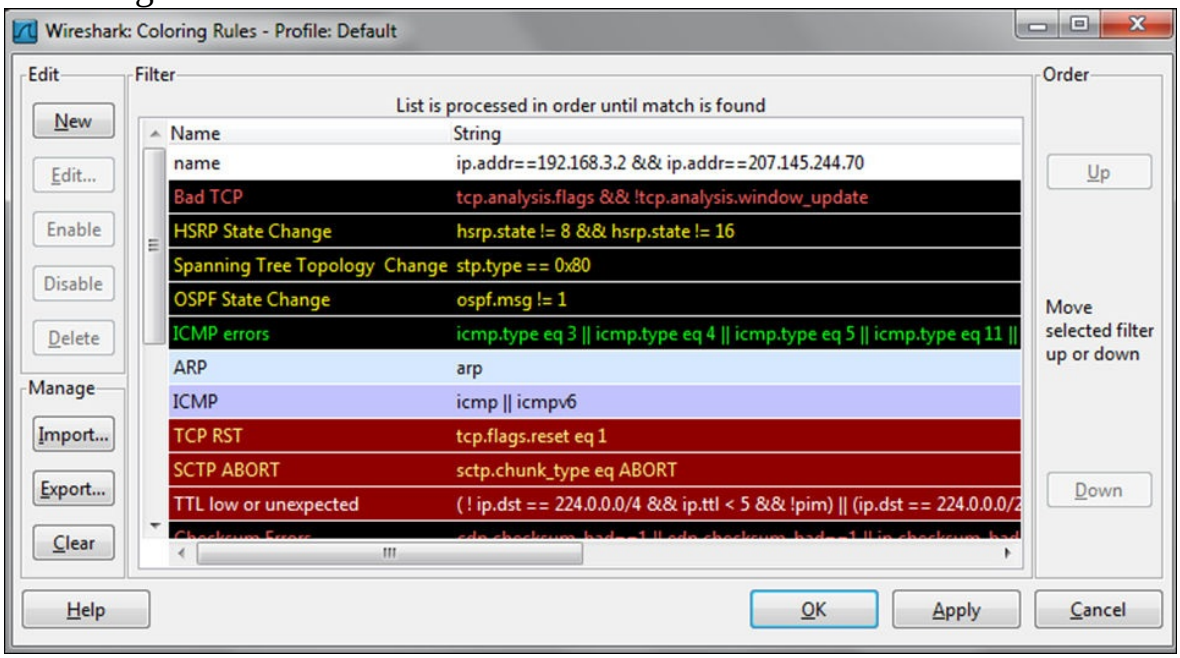
Tip

You can configure Wireshark Profiles in order to save Wireshark configuration; for example, predefined colors, filters, and so on. To do so, navigate to **Configuration Profiles** from the **Edit** menu.

Getting ready

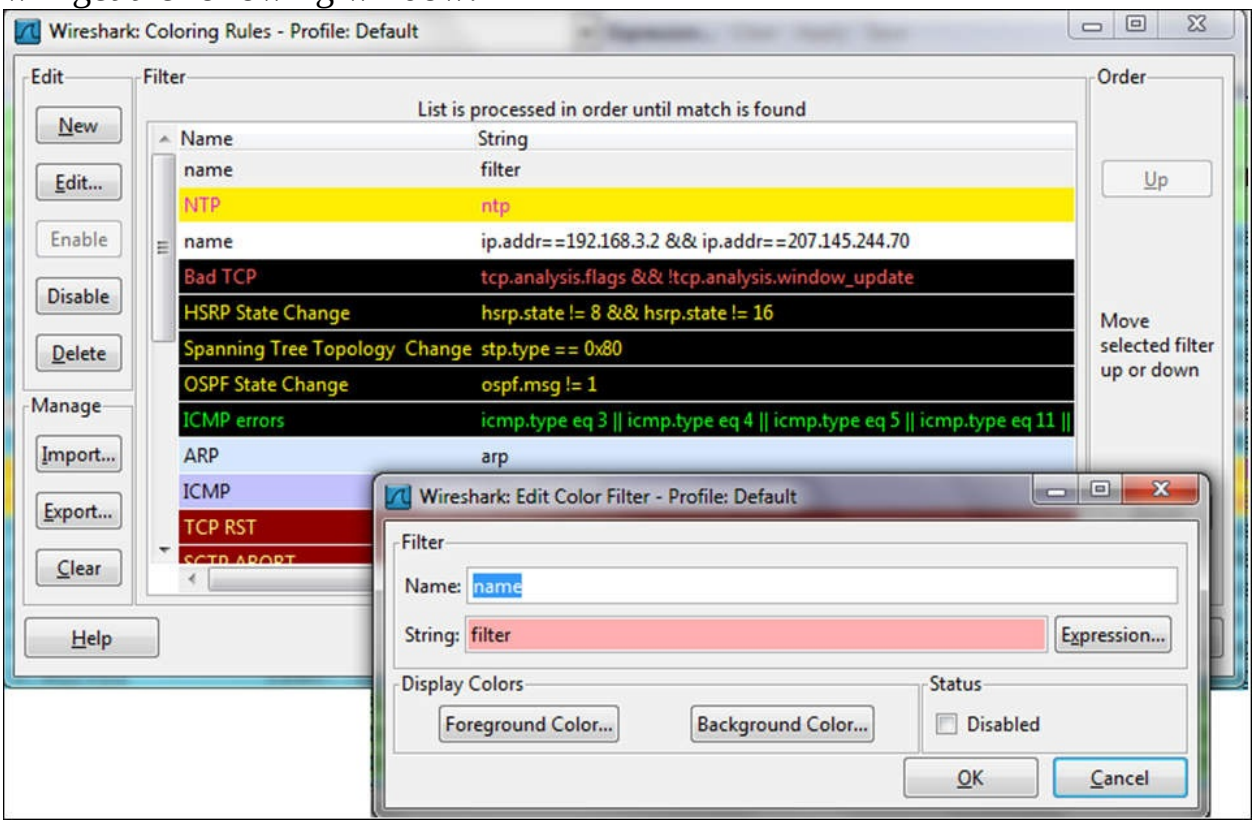
To start with the coloring rules, proceed as follows:

1. Go to the **View** menu.
2. On the lower part of the menu, choose **Coloring Rules**. You will get the following window:



How to do it...

We will now move on to the coloring rules: Click on the **New** button, and you will get the following window:



In order to configure a new coloring rule, follow these steps:

1. In the **Name** field, fill in the name of the rule. For example, fill in `NTP` for the Network Time Protocol.
2. In the **String** field, fill in the filter string, that is, what you want the rule to show (we will talk about display filters in Chapter 3, *Using Display Filters*). You can click on the expression button and get a list of preconfigured filters.
3. Click on the **Foreground Color** button and choose the foreground color for the rule. This will be the foreground color of the packet in the packet list.
4. Click on the **Background Color** button and choose the background color for the rule. This will be the background color of the packet in the packet

list.

5. Click on the **Edit** button if you want to edit an existing rule. You can also either click on the **Import** button to import an existing coloring scheme, or click on the **Export** rule for exporting the current scheme.

Tip

There is an importance to the order of the coloring rules. Make sure the order that the coloring rules are in is the order of implementation. For example, application layer protocols should come before TCP or UDP, so that Wireshark colors them in their color and not the regular TCP or UDP color.

How it works...

Like many operations in Wireshark, you can configure various operations on the data that is filtered. The coloring rules mechanism simply applies a coloring rule to a predefined filter.

See also

You can find various types of coloring schemes at http://wiki.wireshark.org/ColoringRules, along with many other examples, in a simple Internet search.

Saving, printing, and exporting data

In this recipe we will talk about file operations such as save, export, print, and others.

Getting ready

Start Wireshark or open a saved file.

How to do it...

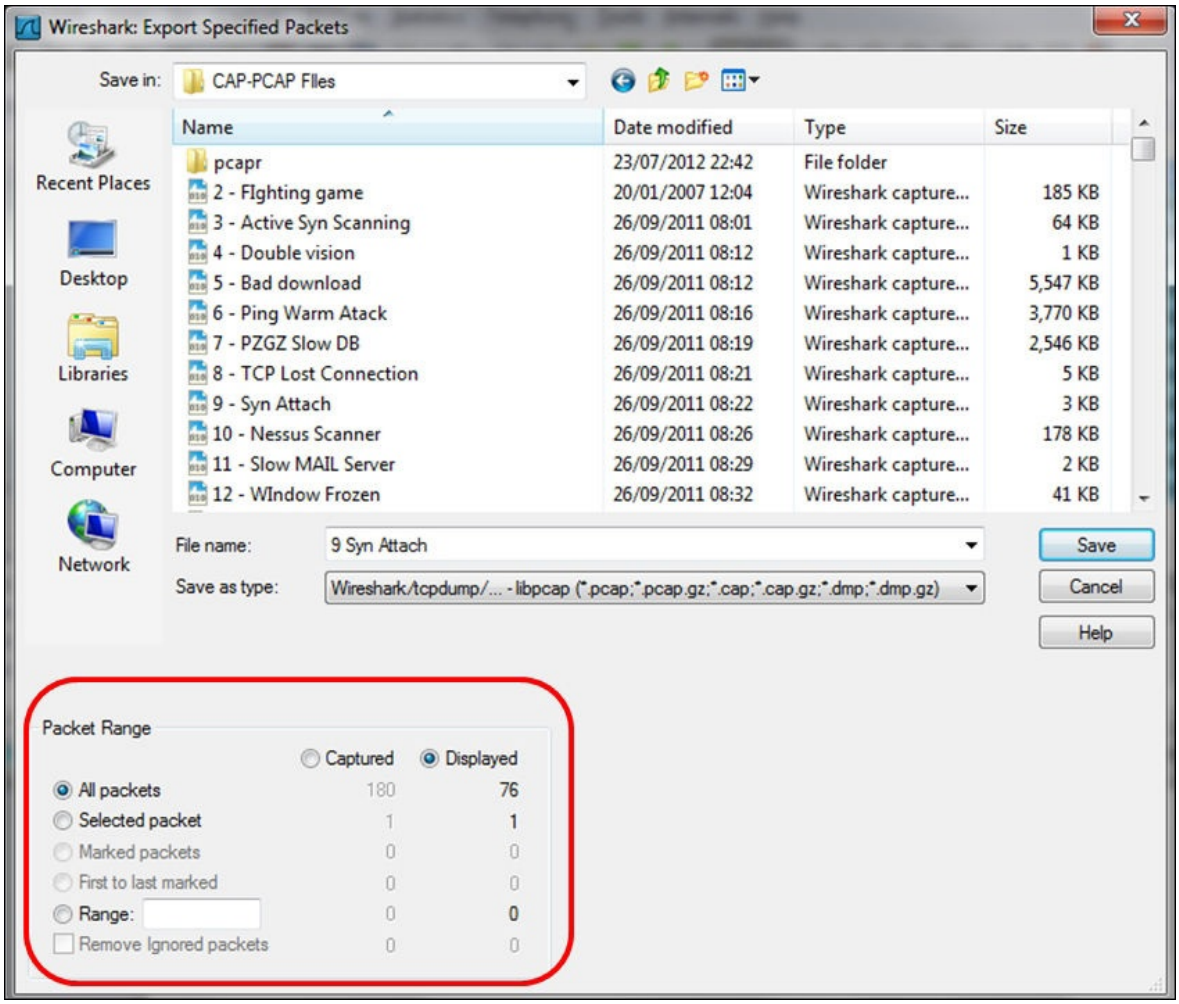
We can save a whole file, and export specific data in various formats and file types. In the following paragraphs we will see how to do it.

To save a whole file with captured data, perform the following steps:

1. In the **File** menu, click on **Save** (or press *Ctrl* + *S*) for saving the file with its own name.
2. In the **File** menu, click on **Save as** (or press *Shift* + *Ctrl* + *S*) for saving the file with a new name.

For saving a part of a file, for example, only the displayed data:

1. Navigate to **Export Specified Packets** under the **File** menu. You will get the following window:



- At the bottom-left side of the window, you will see that you can choose which part of the data you want to save.
- For saving all the captured data, select **All packets** and **Captured**.
- For saving only the displayed data, choose **All packets** and **Displayed**.
- For saving only selected packets from the file (a selected packet is simply a packet that you clicked on), choose **Selected packet**.
- For saving marked packets (that is, packets that were marked by right-clicking on it in the packet list window, and choosing the **Marked packet** toggle from the menu), choose **Marked packet**.
- For saving packets between two marked packets select the **First to last marked** option.
- For saving a range of packets, select **Range** and specify the range of packets you want to save.

- In the packet list window, you can manually choose to ignore a packet. In the **Export** window you can choose to ignore these packets and not save them.

In all the options mentioned, you can choose the packets from the entire captured file, or from the packets displayed on the screen (packets displayed on the packet list after a displayed filter has been applied).

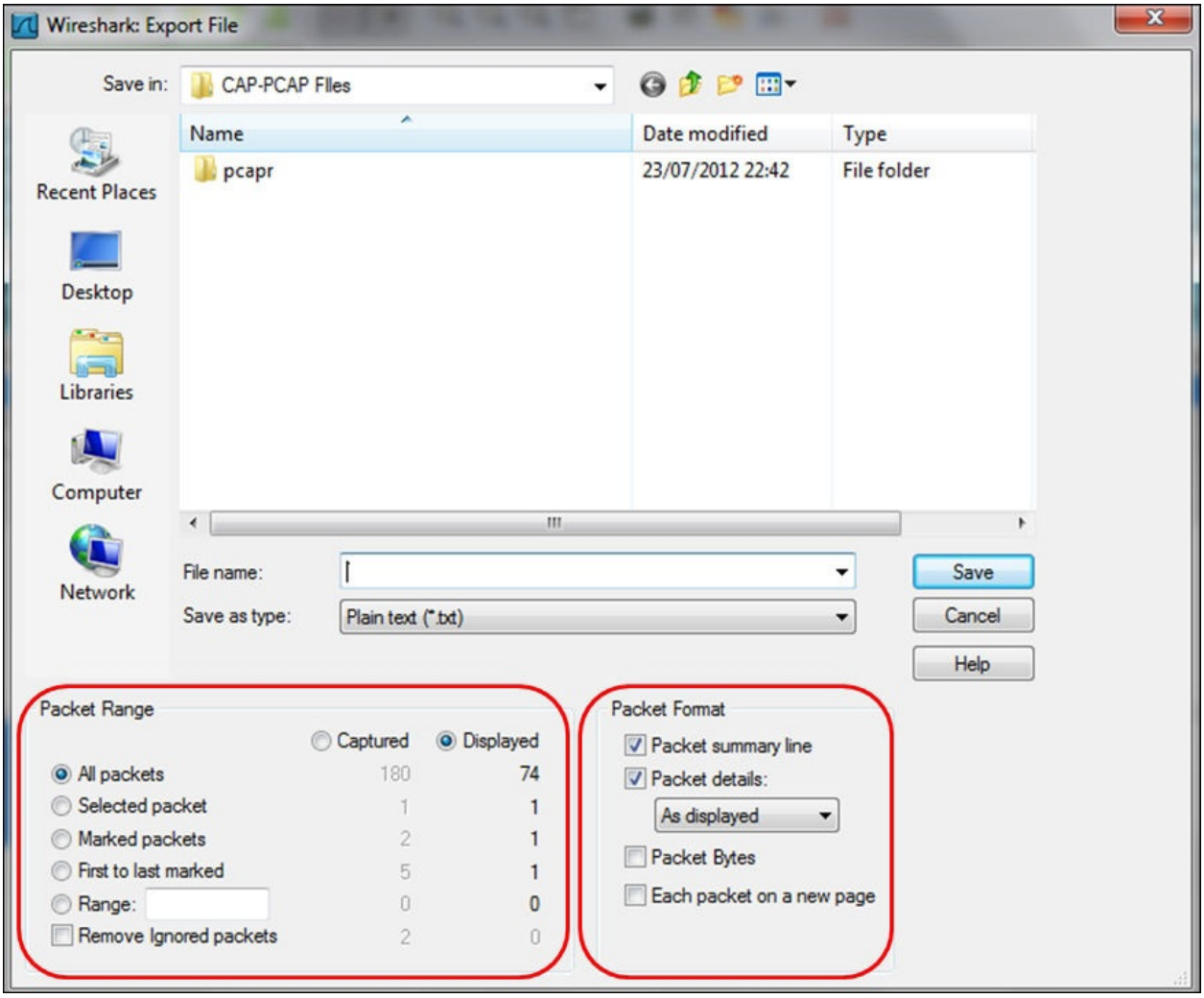
Saving data in various formats

You can save the data captured by Wireshark in various formats, for further analysis with other tools.

You can save the file in the following formats:

- **Plain text** (`*.txt`): export packet data into a plain text ASCII file.
- **PostScript** (`*.ps`): export packet data into PostScript format.
- **Comma Separated Values**: Packet Summary (`*.csv`): export packet summary into CSV file format, to use it with spreadsheet programs (such as Microsoft Excel).
- **C Arrays to Packet Bytes** (`*.c`): export packet bytes into C-Arrays so that it can be imported by C programs.
- **PSML or XML Packet Summary** (`*.psml`): export packet data into PSML, an XML-based format including only the packet summary. Further details about this format can be found at http://www.nbee.org/doku.php?id=netpdl:psml_specification.
- **PDML - XML Packet Details** (`*.pdml`): export packet data into PDM, an XML-based format including the packet details. Further details about this format can be found at http://www.nbee.org/doku.php?id=netpdl:pdml_specification.

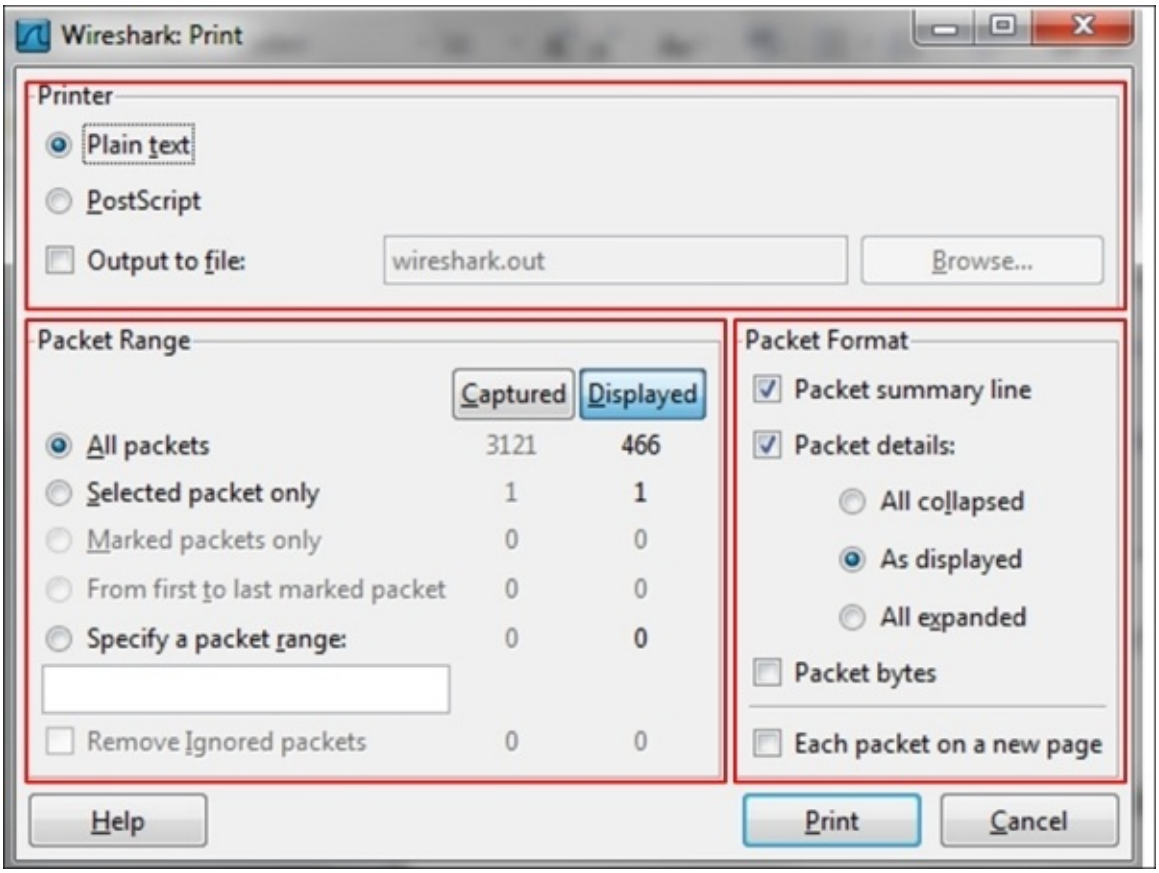
To save the file, select **Export Packet Dissections** from the **File** menu, and you will get the following window:



In the preceding screenshot, in the marked box on the left-hand side, you choose the packets you want to save. The process is the same as in the previous recipe. In the marked box on the right-hand side, you choose the format of the file to be saved.

How to print data

In order to print data, click on the **Print** button from the **File** menu, and you will get the following window:



In the Wireshark **Print** window, you have the following choices:

- In the upper window, you choose the file format to be printed
- In the lower-left window, you choose the packet to print (like in the Export window)
- In the lower-right window, you choose the format of the printed data, and the data panes to print from the Wireshark window:
    - The **Packet Summary** pane
    - The **Packet Details** pane
    - The **Packet Byte** pane

How it works...

The data can be printed in a text format, postscript (for postscript-aware printers), or to a file. After configuring this window and clicking on print, the regular printing window will appear and you can choose the printer.

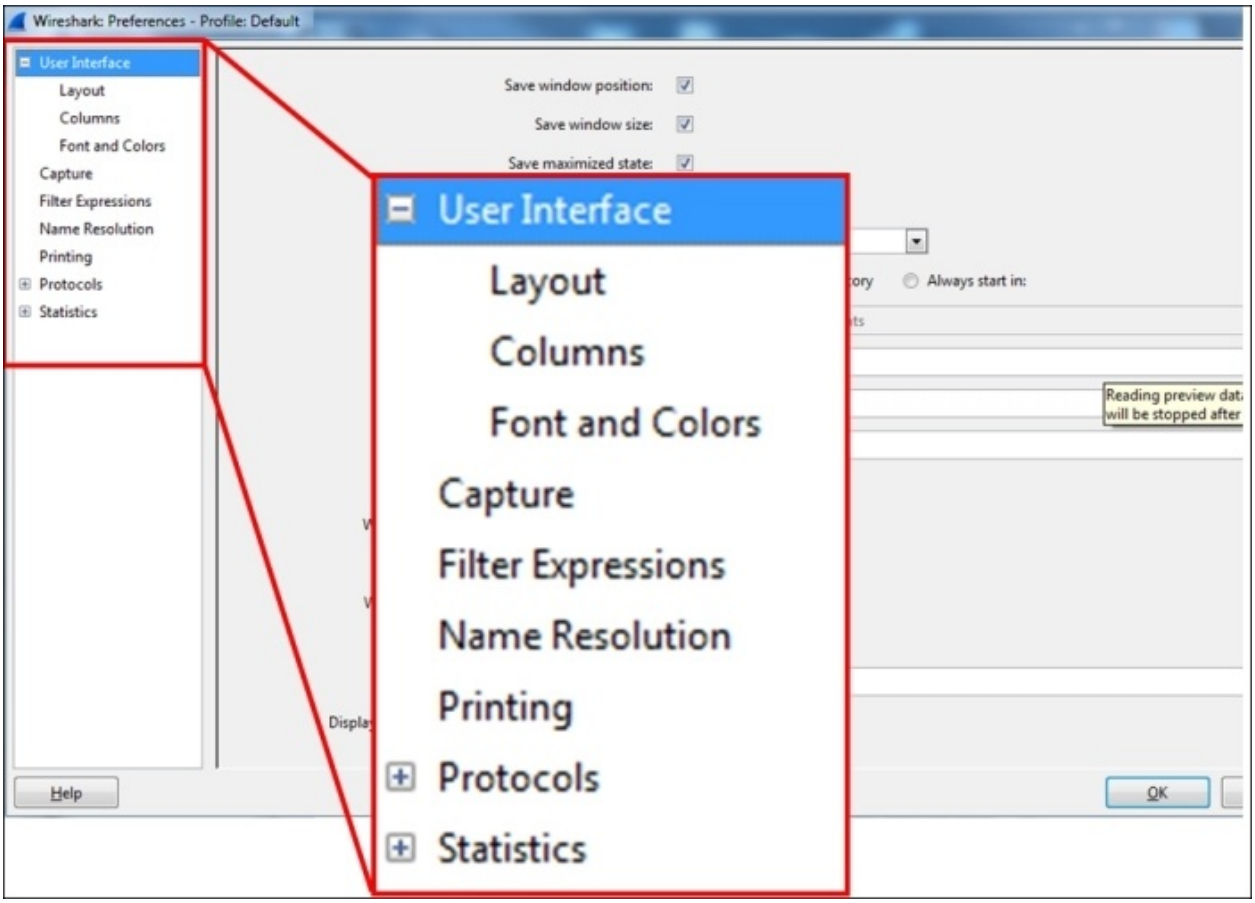
Configuring the user interface in the Preferences menu

There are a large number of parameters you can change in the **Preferences** window, including what data is presented, where files are saved by default, what is the default interface that Wireshark captures data from, and many more.

What we will refer to in this chapter are the common parameters that when changed will help us with various capture scenarios.

Getting ready

For configuring **User Interface**, we will choose the **Preferences** option from the **Edit** menu. You will get the following window:



We will look at the configuration of the following parameters:

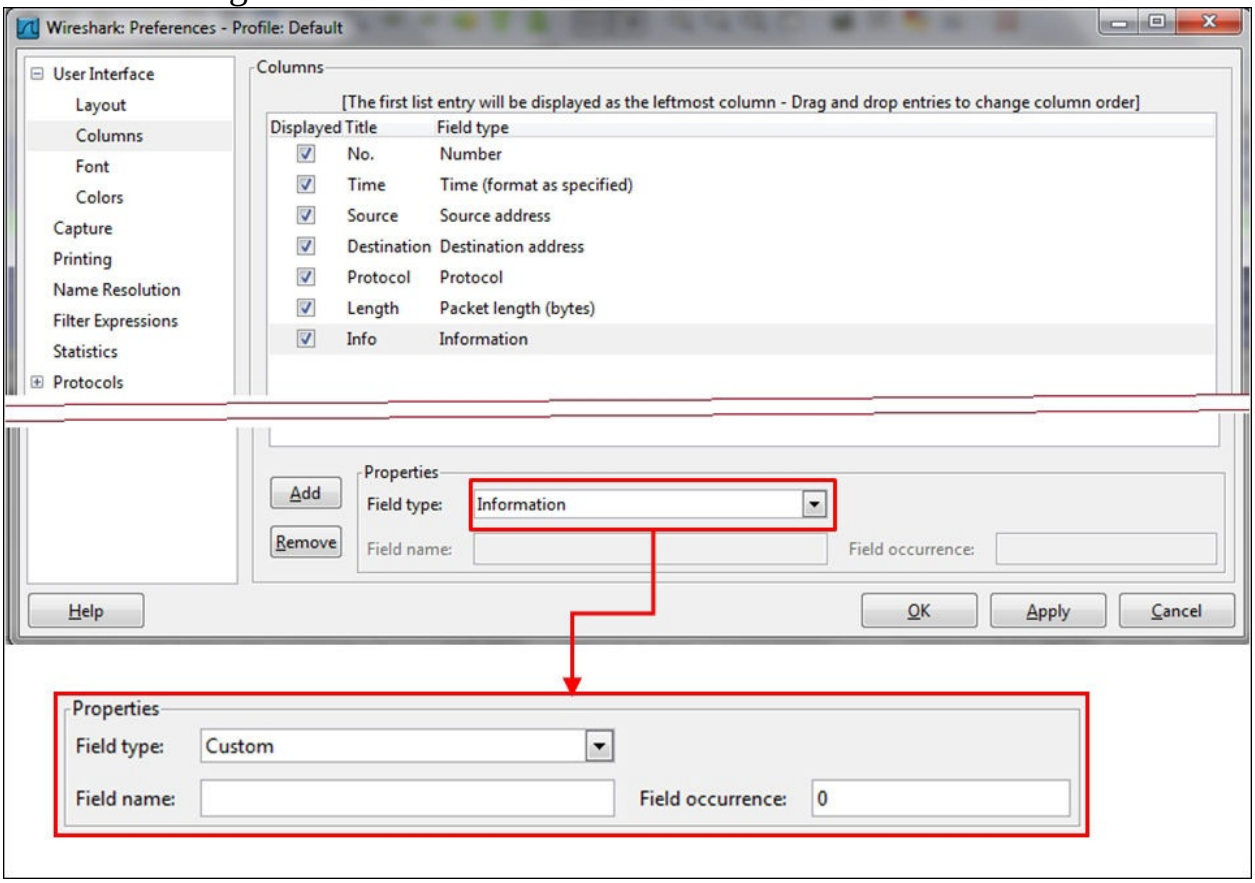
- **Columns**
- **Capture**
- **Name Resolution**

How to do it...

In this section we will see how to change parameters that will help in working with Wireshark.

Changing and adding columns

The default columns that we see in the packet pane are the number, time, source and destination addresses, protocol, length, and information columns, as shown in the following screenshot:



To add a new column to the packet pane:

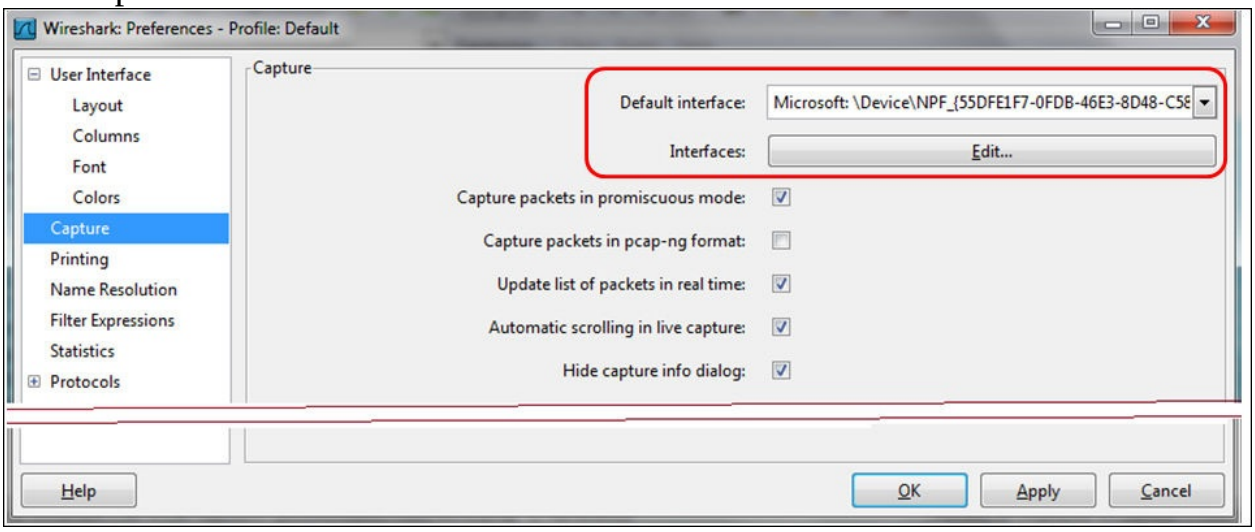
1. You can choose one of the predefined parameters to be added as a new column from the **Field type**. Among these parameters are time delta, IP DSCP value, port numbers, and others.
2. A very important feature comes up when you fill in **Custom** in the field

type. In this case, you can fill in any filter string for **Field name**. You can, for example, add the following:

1. Add the string `tcp.window_size` to view the TCP window size (that influences performance).
2. Add the string `ip.ttl` to view the IP **TTL (Time-To-Live)** parameter of every packet.
3. Add `rtp.marker` to view every instance of a marker set in an RTP packet.
4. As we will see in the later chapters, this feature will assist us a lot for fast resolutions of network problems.

Changing the capture configuration

There are some parameters that can be configured before capturing data. In the **Preferences** window choose the **Capture** menu, and the following window will come up:

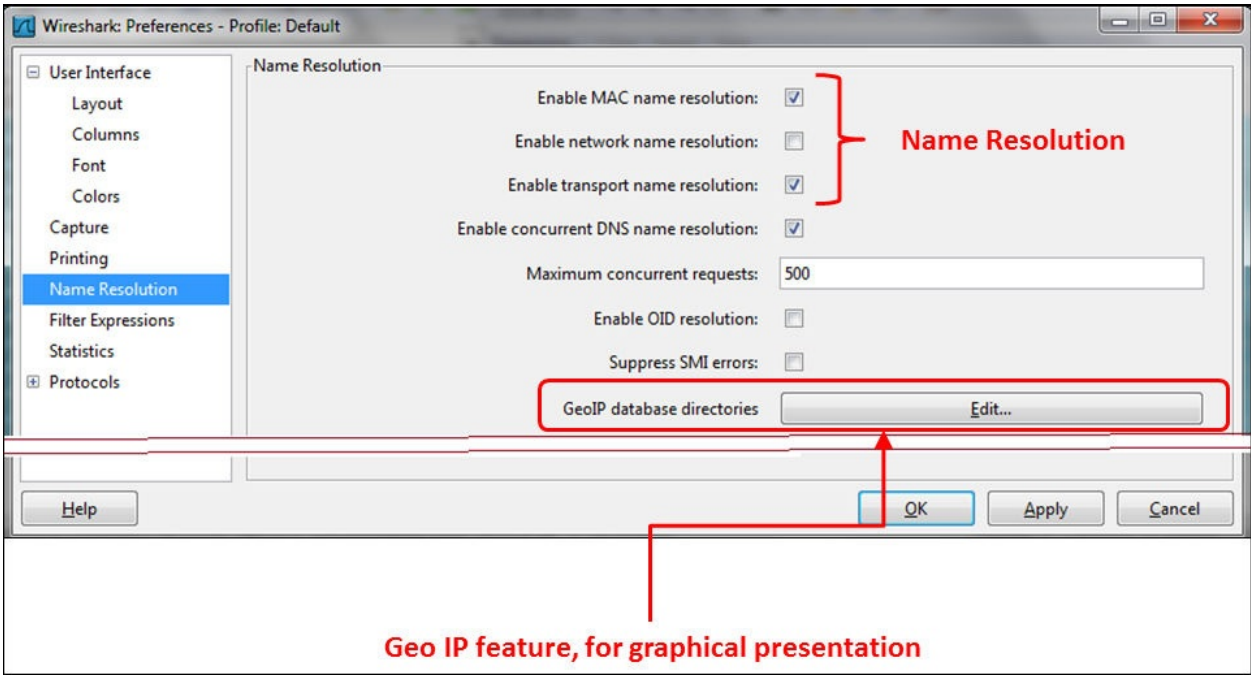


For changing the default interface that the capture will start from, just click on the **Edit** button, and mark the interface you would like to be the default. Of course you can change it every time you start a new capture, this is only the default.

Configuring the name resolution

Wireshark supports **Name Resolution** in three layers:

- **Layer 2**: by resolving the first part of the MAC addresses to the vendor name. For example, **14:da:e9** will be presented as AsusTeckC (ASUSTeK Computer Inc.).
- **Layer 3**: by resolving IP addresses to the DNS names. For example, **157.166.226.46** will be resolved to www.edition.cnn.com.
- **Layer 4**: by resolving TCP/UDP port numbers to port names. For example, port 80 will be resolved as HTTP, and port 53 as DNS.



Tip

In TCP and UDP, there is a meaning only to the destination port that the client initially opens the session to. The source port that the connection is opened from is a random number (higher than 1024), and therefore there is no meaning to its translation to a port name.

The Wireshark default is to resolve layer-2 MAC addresses and layer-4 TCP/UDP port numbers. Resolving IP addresses can slow down Wireshark due to a large amount of DNS queries that it uses; therefore, use it carefully.

How it works...

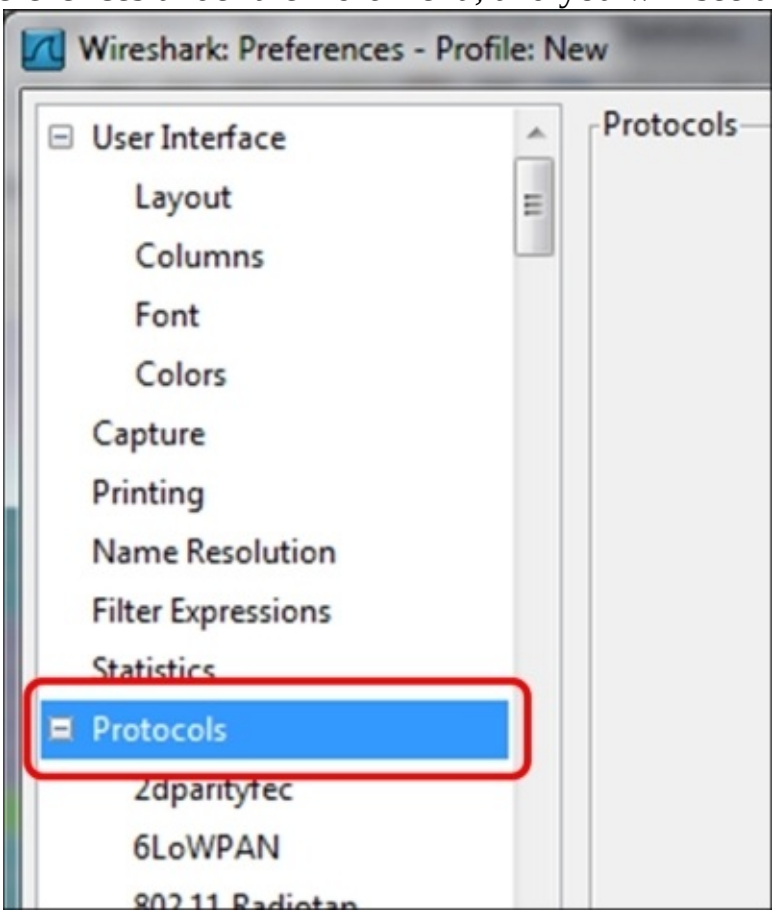
Very simple. This is the configuration menu for the Wireshark. Here you can configure parameters as described in this recipe, along with some other parameters. You can refer to Wireshark manuals at www.wireshark.org for further information.

Configuring protocol preferences

Configuring protocol preferences provides us with capabilities to change the way that Wireshark captures and presents common protocols. In this recipe we will learn how to configure the most common protocols.

Getting ready

1. Go to **Preferences** under the **Edit** menu, and you will see the following window:



- Click on the + sign on the left side of the protocols, and a protocol list will be opened. Under the protocol list you will find the common and lesser-common protocols. In this part we will talk about the common configurations, and we'll get into protocol details in the protocols chapters that is, Chapter 7, *Ethernet, LAN Switching, and Wireless LAN*, to Chapter 14, *Understanding Network Security*.

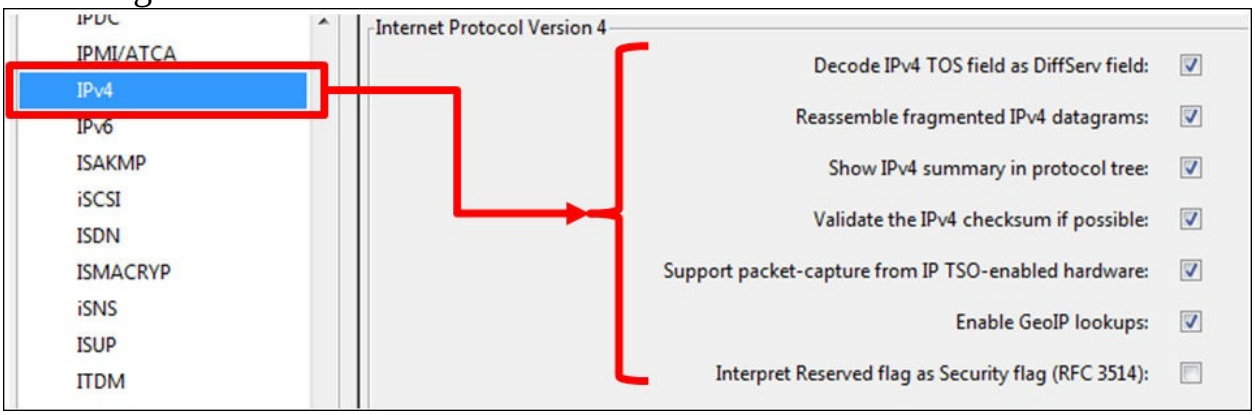
How to do it...

In this recipe, we will talk about the following basic protocols (basic means that they are used everywhere, not that they are simple):

- IPv4 and IPv6
- TCP and UDP

Configuring of IPv4 and IPv6 Preferences

When you choose to configure the IPv4 or IPv6 parameters, you will get the following window:



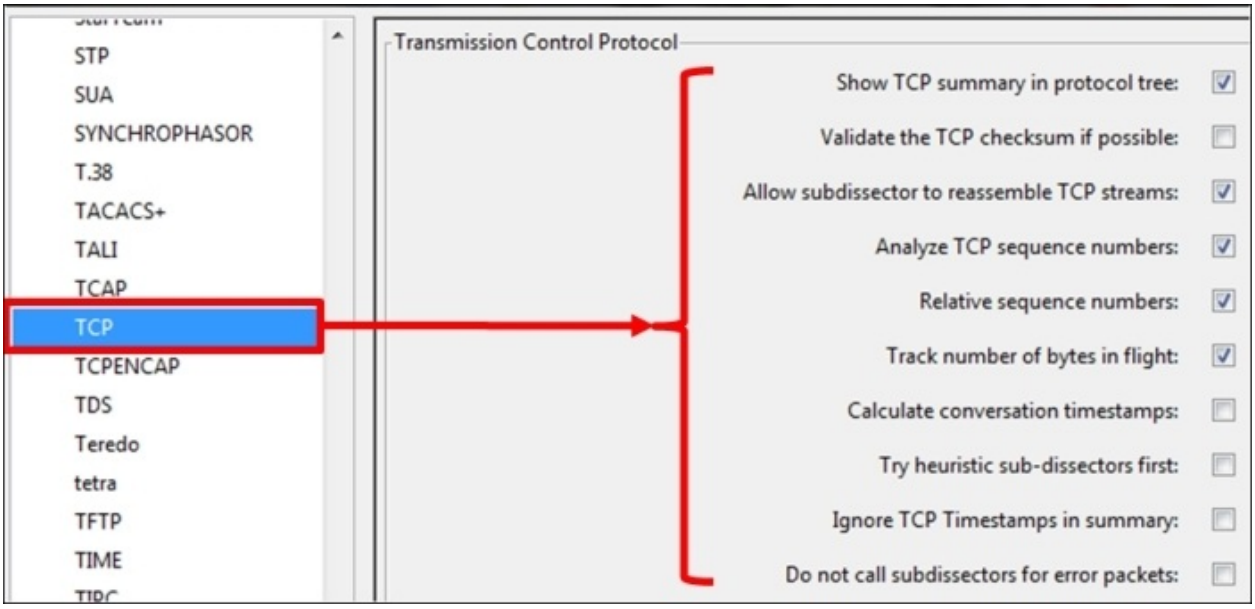
The parameters that you may change are:

- **Decode IPv4 ToS field as DiffServ Field**: the original IP protocol came out with a field called **Type Of Service** (**ToS**), for enabling the IP quality of service through the network. In the early 90s the **Differentiated Services** (**DiffServ**) standard changed the way that an IP device looked on this field. Unchecking this checkbox will show this field as in the original IP standard.
- **Enable GeoIP lookups**: GeoIP is a database that enables Wireshark to present IP addresses as geographical locations. Enabling this feature in IPv4 and IPv6 will enable this presentation. This feature involves name resolutions and can therefore slow down packet capture in real time.

Configuring TCP and UDP

In UDP, there is not much to change. A very simple protocol, with a very simple

configuration. In TCP on the other hand, there are some parameters that can be changed.



Most of the changes you can do in the TCP preferences are in the way that Wireshark dissects the captured data.

- **Validate the TCP checksum if possible**: in some NICs, you may see many "checksum errors". This is due to the fact that TCP Checksum offloading is often being implemented on some NICs. The problem here might be that the NIC actually adds the checksum AFTER Wireshark captures the packet, so if you see many TCP checksum errors, the first thing to do will be to disable this checkbox and verify that this is not the problem.
- **Analyze TCP Sequence numbers**: this checkbox must be checked for Wireshark to provide TCP analysis, which is one of its main and most important features.
- **Relative Sequence Numbers**: when TCP opens a connection, it starts from a random sequence number. When this checkbox is checked, the Wireshark will normalize it to "0", so what you will see are not the real numbers, but numbers starting from "0" and increasing. In most of the cases the relative numbers are much easier to handle.
- **Calculate conversations timestamps**: When checking this checkbox, the TCP dissector will show you the time since the beginning of the connection

in every packet. This can be helpful in cases of very fast connection when times are critical.

How it works...

Using the **Protocols** feature from the **Preferences** menu adds more analysis capabilities to the Wireshark software. Just be careful here to not add too many capabilities that will slow down the packet capture and analysis.

There's more...

You can get more information on GeoIP at http://wiki.wireshark.org/HowToUseGeoIP.

Chapter 2. Using Capture Filters

In this chapter, we will cover the following topics:

- Configuring capture filters
- Configuring Ethernet filters
- Configuring hosts and networks filters
- Configuring TCP/UDP and port filters
- Configuring compound filters
- Configuring byte-offset and payload matching filters

Introduction

In the first chapter we talked about how to install Wireshark, how to configure it for basic operations, and where to locate it in the network. In this chapter and the next one we will talk about capture filters (Chapter 2, *Using Capture Filters*) and display filters (Chapter 3, *Using Display Filters*).

It is important to distinguish between these two types of filters:

- Capture filters are configured before we start to capture data, so only data that is approved with the filters will be captured. All other data will be lost. These filters are described in this chapter.
- Display filters are filters that filter data after it has been captured. In this case, all data is captured, and you configure what data you wish to display. These filters are described in Chapter 3, *Using Display Filters*.

Tip

Capture filters are based on the `tcpdump` syntax presented in the `libpcap`/`WinPcap` library, while the display filters syntax was presented some years later. Therefore, keep in mind that the display and capture filters have different syntaxes!

In some cases, you need to configure Wireshark to capture only a part of the data that it sees over the interface, for example, cases such as:

- When there is a large amount of data running over the monitored link, and you want to capture only the data you care about
- When you want to capture data only going in and out of a specific server on a VLAN that you monitor
- When you want to capture data only of a specific application or applications (for example, you suspect that there is a DNS problem in the network, and you want to analyze only DNS queries and responses that go to and from the Internet)

There are many other cases in which you want to capture only specific data and not all that runs on your network. When using the capture filters, only predefined data will be captured, and all other packets will be ignored, so you will get only the desired data.

Tip

Be careful when using capture filters. In many cases in networking, there are dependencies between different applications and servers that you are not always aware of; so, when you use Wireshark with capture filters for troubleshooting a network, make sure that you don't filter out some of the connections that will cause some problems to disappear. A common and simple example of this is to filter only traffic on TCP port 80 for monitoring suspected slow HTTP responses, while the problem could be due to a slow or non-responsive DNS server that you will not see.

In this chapter we will describe how to configure simple, structured, byte offset, and payload matching capture filters.

Configuring capture filters

We recommend that before configuring a capture filter, you will carefully design what you want to capture, and prepare your filter for this. Don't forget—what doesn't pass the filter, will be lost.

There are some Wireshark predefined filters that you can use, or you can configure it yourself as described in the next recipe.

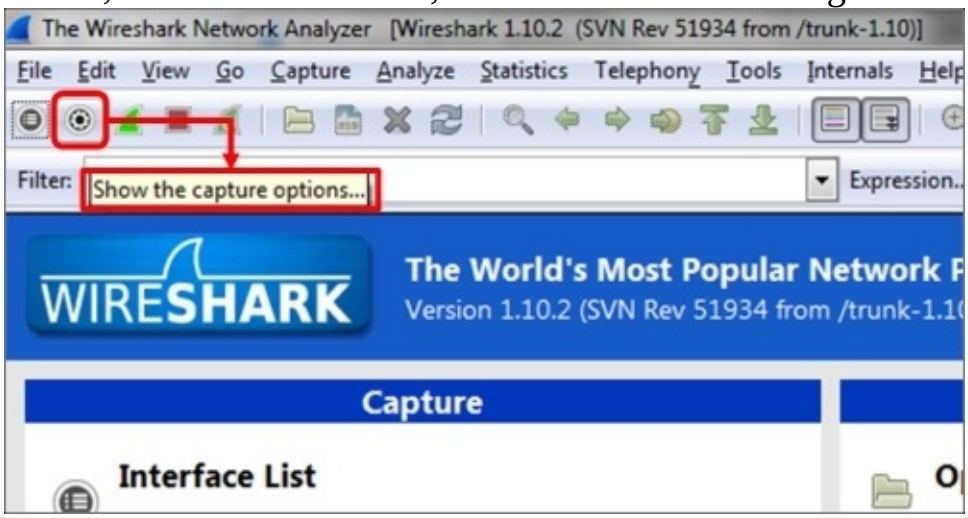
Getting ready

For configuring capture filters, open Wireshark, and follow the steps in the recipe.

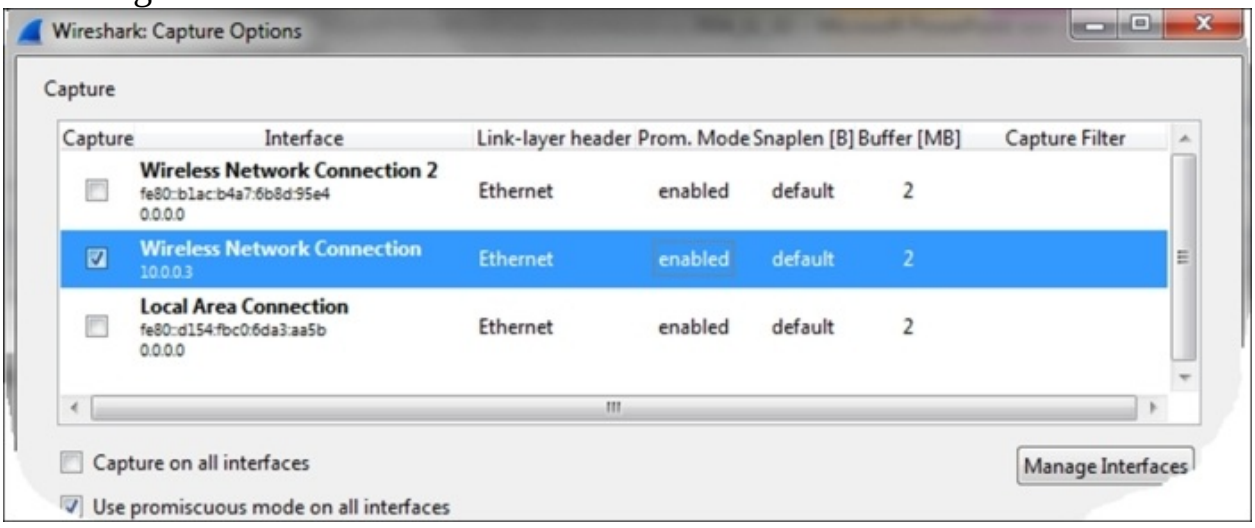
How to do it...

For configuring capture filters before starting with the capture, go through the following steps:

1. For configuring a capture filter, click on the **Show the capture options…** button, second from the left, as shown in the following screenshot:

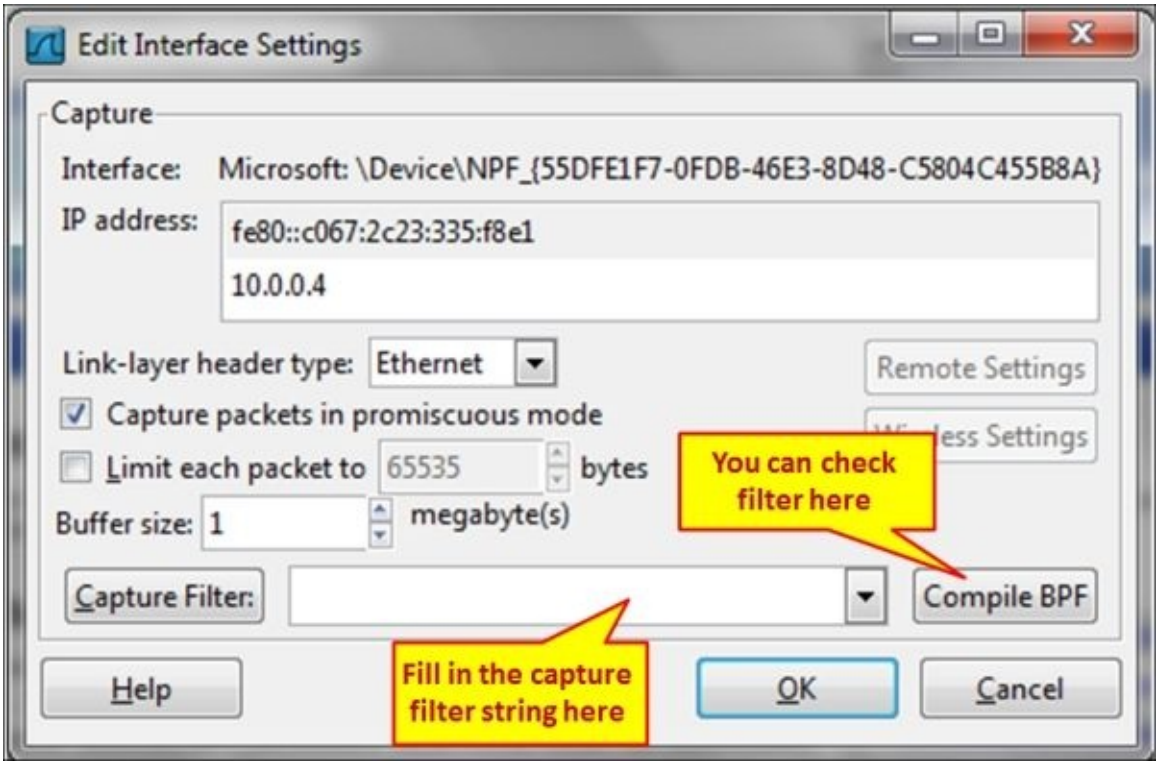


- The **Wireshark: Capture Options** window will open as you see in the following screenshot:

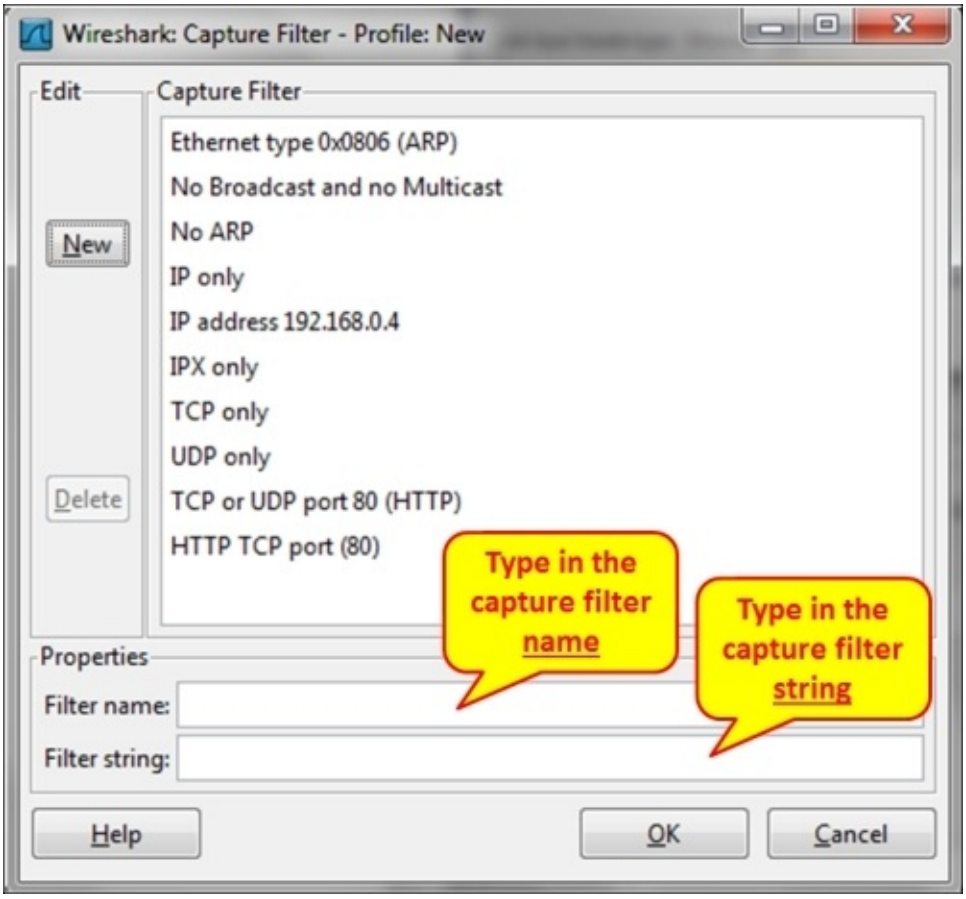


- Double-click on the interface on which you want to configure the capture filter (you can verify which interface is the active one, as described in Chapter 1, *Introducing Wireshark*).
- The **Edit Interface Settings** window will open up, as in the following

screenshot:



- Now, we can configure the capture filters by simply writing the filter string in the **Wireshark: Capture Filter** window, or click on the **Capture Filter:** button; the following window will open:


Wireshark: Capture Filter - Profile: New
Edit
Capture Filter
New
Ethernet type 0x0806 (ARP)
No Broadcast and no Multicast
No ARP
IP only
IP address 192.168.0.4
IPX only
TCP only
UDP only
Delete
TCP or UDP port 80 (HTTP)
HTTP TCP port (80)
Type in the capture filter name
Type in the capture filter string
Properties
Filter name:
Filter string:
Help
OK
Cancel

How it works...

The **Wireshark: Capture Filter** window enables you to configure filters according to **Berkeley Packet Filter** (**BPF**). After writing a filter string, you can click on the **Compile BPF** button, and the BPF compiler will check your syntax, and if it's wrong you will get an error message.

In addition to this, when you type a filter string in the capture filter text box, and the filter string is correct, it will become green, and if not, it will become red.

The BPF filter only checks if the syntax is right. It does not check if the condition is correct. For example, if you type the string `host` without any parameter, you will get an error and the string will become red, but if you type `host 192.168.1.1000`, it will pass and the window will become green.

Tip

BPF is a syntax coming from the paper *The BSD Packet Filter: A New Architecture for User-level Packet Capture* by *Steven McCanne* and *Van Jacobson* from the Lawrence Berkeley Laboratory at Berkeley University from December 1992. The document can be seen at: http://www.tcpdump.org/papers/bpf-usenix93.pdf.

Capture filters are made out of a string containing a filtering expression. This expression selects the packets which will be captured and which packets will be ignored. Filter expressions consist of one or more primitives. Primitives usually consist of an identifier (name or number) followed by one or more qualifiers. There are three different kinds of qualifiers:

- **Type**: These qualifiers say what kind of thing the identifier name or number refers to. Possible types are `host` for host name or address, `net` for network, `port` for TCP/UDP port, and so on.
- **Dir (direction)**: These qualifiers specify a particular transfer direction to and/or from ID. For example `src` indicates source, `dst` indicates destination, and so on.
- **Proto (protocol)**: These are the qualifiers that restrict the match to a particular protocol. For example, `ether` for Ethernet, `ip` for Internet Protocol, `arp` for Address Resolution Protocol, and so on.

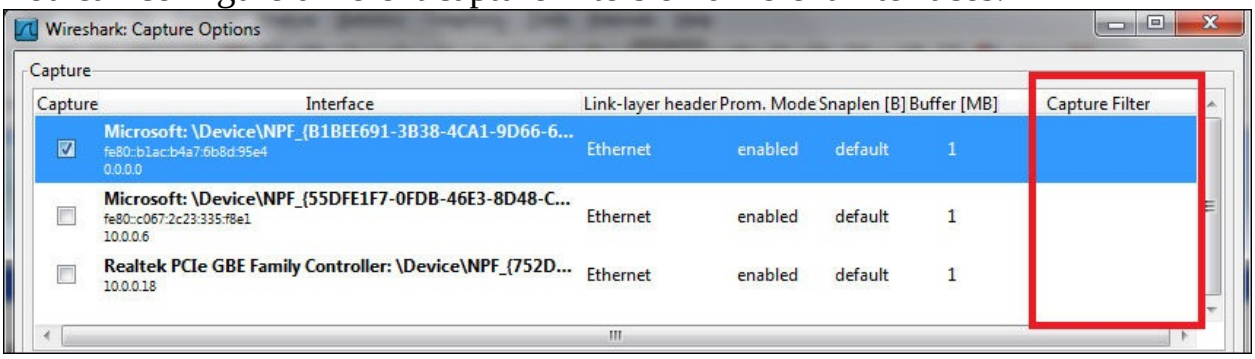
Identifiers are the actual condition that we test. Identifier can be the address 10.0.0.1, port number 53, or network address 192.168.1 (this is an identifier for network 192.168.1.0/24).

For example, in the filter `tcp dst port 135`, we have:

- `dst` is the dir qualifier
- `port` is the type qualifier
- `tcp` is the Proto qualifier

There's more...

You can configure different capture filters on different interfaces:



This can be used when you capture traffic on two interfaces of a device, and want to check for different packets on the two sides.

The capture filters are stored in a file named `cfilters` under the Wireshark directory. In this file you will find the predefined filters, along with the filters you have configured, and you will be able to copy the file to other computers. The location of this directory will change depending on how Wireshark is installed and on what platform.

See also

1. The Wireshark Capture Filters are based on the `tcpdump` program. You can find the reference at http://www.tcpdump.org/tcpdump_man.html.
2. You can also find helpful information on the Wireshark manual pages at http://wiki.wireshark.org/CaptureFilters.

Configuring Ethernet filters

When talking about Ethernet filters, we refer to Layer-2 filters that are MAC address-based filters. In this recipe we will refer to these filters and what we can do with them.

Getting ready

The basic Layer 2 filters are:

- `ether host <Ethernet host>`: To get the Ethernet address
- `ether dst <Ethernet host>`: To get the Ethernet destination address
- `ether src <Ethernet host>`: To get the Ethernet source address
- `ether broadcast`: To capture all Ethernet broadcast packets
- `ether multicast`: To capture all Ethernet multicast packets
- `ether proto <protocol>`: To filter only the protocol type indicated in the protocol identifier
- `vlan <vlan_id>`: To pass only packets from a specific VLAN that is indicated in the identifier field

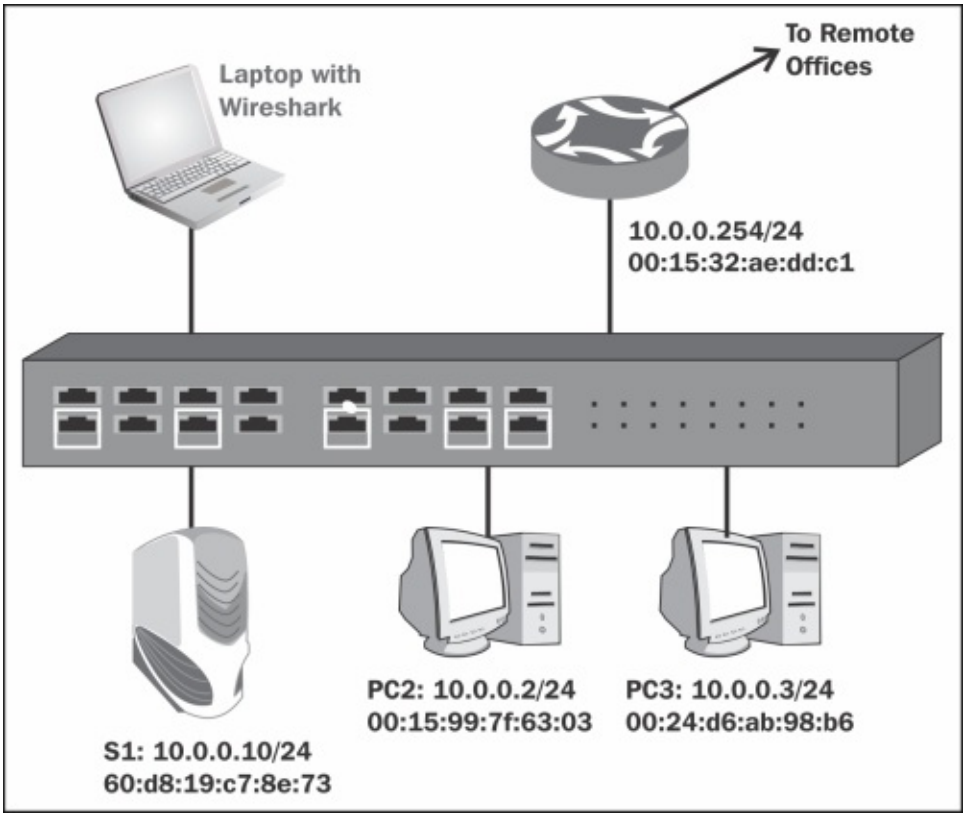
For negating a filter rule, simply type the word `not` or `!` in front of the primitive. For example:

`Not ether host <Ethernet host>` or `! Ether host <Ethernet host>` will capture packets that are not from/to the Ethernet address specified in the identifier field.

How to do it...

Let's look at the following diagram, in which we have a server, PCs, and a router, connected to a LAN switch. Wireshark is running on the laptop connected to the LAN switch, with port mirror to the entire switch (to VLAN1).

The `/24` notation in the drawing refers to a subnet mask of 24 bits, that is, 11111111.11111111.11111111.00000000 in binary or 255.255.255.0 in decimal.



Follow the instructions in the *Configuring capture filters* recipe, and configure filters as follows:

1. To capture packets only from/to a specific MAC address, for example of PC3 in the preceding image, configure `ether host 00:24:d6:ab:98:b6`.
2. To capture packets going to a destination MAC address, for example of PC3 in the preceding image, configure `ether dst 00:24:d6:ab:98:b6`.
3. To capture packets coming from a source MAC address, for example of

PC3 in the preceding image, configure `ether src 00:24:d6:ab:98:b6`.

4. To capture broadcast packets, configure `ether broadcast` or `ether dst ff:ff:ff:ff:ff:ff`.
5. To capture multicast packets, configure `ether multicast`.
6. To capture a specific Ether Type (number in Hexadecimal value), configure `ether proto 0800`.

How it works…

The way capture filters work with source host and destination host is simple—the capture engine simply compares the condition with the actual MAC addresses, and passes only what is relevant.

A broadcast address is an address in which the destination address is all 1's, that is, `ff:ff:ff:ff:ff:ff:ff:ff`, therefore when you configure a broadcast filter, only these addresses will pass the filter. Broadcast addresses can be:

- L3 IPv4 broadcast that is converted to L2 broadcast; for example, IP packet to 10.0.0.255 (class C subnet, as in the previous illustration), which will be converted to L2 broadcast in the destination MAC field.
- A network-related broadcast; for example, IPv4 ARP (Address Resolution Protocol) that sends a broadcast as a part of network operation.

Tip

Network-related broadcasts are broadcasts that are sent for the regular operation of the network. Among these are ARPs, routing updates, discovery protocols, and so on.

In a multicast filter, there are IPv4 and IPv6 multicasts:

- In IPv4, a multicast MAC address is transmitted when the MAC address starts with the string `01:00:5e`. Every packet with a MAC address that starts with this string will be considered a multicast.
- In IPv6, a multicast address is transmitted when the MAC address starts with the string `33:33`. Every packet with a MAC address that starts with this string will be considered a multicast.

Ethernet protocol refers to the ETHER-TYPE field in the Ethernet packet that indicates what will be the upper Layer protocol. Common values here are `0800` for IPv4, `86dd` for IPv6, and `0806` for ARP.

There's more...

- To configure filter for a specific VLAN, use `vlan <vlan number>`
- To configure filter on several VLANs, use `vlan <vlan number> and vlan <vlan number> and vlan <vlan number>...`

See also

There are around a hundred ETHER-TYPE codes, most of them not in use. You can refer to http://www.mit.edu/~map/Ethernet/Ethernet.txt for additional codes, or simply browse the Internet for Ethernet code.

Configuring host and network filters

When talking about host and network filters, we refer to Layer 3 filters that are IP address-based filters. In this recipe we will refer to these filters and what we can do with them.

Getting ready

The basic Layer 3 filters are:

- `ip` or `ip6`: To capture IP or IPv6 packets.
- `host <host>`: To get host name or address.
- `dst host <host>`: To get destination host name or address.
- `src host <host>`: To get source host name or address.

  Tip

  Host can be an IP address or a host name related with this number. You can type, for example, a filter host www.packtpub.com that will show you all packets to/from the IP address related to the Packt website.

- `gateway <Host name or address>`: It captures traffic to or from the hardware address but not to the IP address of the host. This filter captures traffic going through the specified router. This filter requires a host name that is used and can be found by the local system's name resolution process (for example, DNS).
- `net <net>`: All packets to or from the specified IPv4/IPv6 network.
- `dst net <net>`: All packets to the specified IPv4/IPv6 destination network.
- `src net <net>`: All packets to the specified IPv4/IPv6 destination network.
- `net <net> mask <netmask>`: All packets to/from the specific network and mask. This syntax is not valid for the IPv6 network.
- `dst net <net> mask <netmask>`: All packets to/from the specific network and mask. This syntax is not valid for the IPv6 network.
- `src net <net> mask <netmask>`: All packets to/from the specific network and mask. This syntax is not valid for the IPv6 network.
- `net <net>/<len>`: All packets to/from the `<net>` network with `<len>` length in bits.
- `dst net <net>/<len>`: All packets to/from the `<net>` network with `<len>` length in bits.
- `dst net <net>/<len>`: All packets to/from the `<net>` network with `<len>` length in bits.
- `broadcast`: All broadcast packets.
- `multicast`: All multicast packets.
- `ip proto <protocol code>`: It captures packets while the IP protocol field

equals to the `<protocol>` identifier. There can be various protocols, such as, TCP (Code 6), UDP (Code 17), ICMP (Code 1), and so on.

- `ip6 proto <protocol>`: It captures IPv6 packets with protocol as indicated in the type field. Note that this primitive does not follow the IPv6 extension headers chain.

  ### Tip

  In IPv6 header, there is a field in the header that can point to an optional extension header, which points to the next extension header, and so on. In the current version, Wireshark capture filter does not follow this structure.

- `icmp[icmptype]==<identifier>`: It captures ICMP packets, while the identifier is ICMP codes, such as `icmp-echo` and `icmp-request`.

How to do it...

Follow the instructions mentioned in the *Configure capture filters* recipe, and configure filters as follows:

1. For capturing packets to/from host `10.10.10.1`, configure `host 10.10.10.1`.
2. For capturing packets to/from host at www.cnn.com, configure `host www.cnn.com`.
3. For capturing packets to host `10.10.10.1`, configure `dst host 10.10.10.1`.
4. For capturing packets from host `10.10.10.1`, configure `src host 10.10.10.1`.
5. For capturing packets to/from network `192.168.1.0/24`, configure `net 192.168.1` or `net 192.168.1.0 mask 255.255.255.0` or `net 192.168.1.0/24`.
6. For capturing all data without broadcasts or without multicasts, configure `not broadcast` or `not multicast`.
7. For capturing packets to/from the IPv6 network 2001::/16, configure `net 2001::/16`.
8. For capturing packets to IPv6 host 2001::1, configure `host 2001::1`.
9. For capturing only ICMP packets, configure `ip proto 1`.
10. For filtering only ICMP Echo's pings, you can use ICMP messages or message codes. configure `icmp[icmptype]==icmp-echo` or `icmp[icmptype]==8`.

How it works…

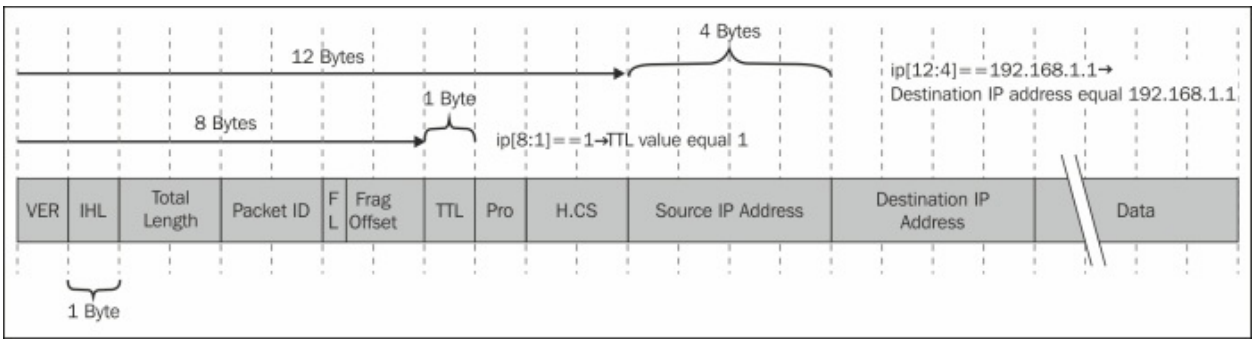
For host filtering, when you type a host name, Wireshark will translate the name to an IP address, and capture packets that refer to this address. For example, if you configure a filter host www.cnn.com, it will be translated by a name resolution service (mostly DNS) to an IP address, and will show you all packets going to and from this address. Note that in this case, if CNN website will forward you to other websites on other addresses, only packets to the first address will be captured.

There's more...

Some more useful filters:

- `ip multicast`: IP multicast packets
- `ip broadcast`: IP broadcast packets
- `ip[2:2] == <number>`: IP packet size
- `ip[8] == <number>`: TTL (Time To Live) value
- `ip[9] == <number>`: Protocol value
- `(ip[12:4] = ip[16:4])`: IP source equal to IP destination address
- `ip[2:2]==<number>`: Total length or IP packet
- `ip[9] == <number>`: Protocol identifier

These filters are further explained in the *Configuring byte offset and payload matching filters* recipe at the end of this chapter. The principle, as illustrated in the following diagram, is that the first number in the brackets defines how many bytes are from the beginning of the protocol header, and the second number indicates how many bytes to watch.



See also

For more filters, refer to the tcpdump manual pages at http://www.tcpdump.org/tcpdump_man.html.

Configuring TCP/UDP and port filters

In this recipe we will present Layer 4 TCP/UDP port filters and how we can use them with capture filters.

Getting ready

The basic Layer 4 filters are:

- `port <port>`: When the packet is a Layer 4 protocol, such as TCP or UDP, this filter will capture packets to/from the port indicated in the identifier field
- `dst port <port>`: When the packet is a Layer 4 protocol, such as TCP or UDP, this filter will capture packets to the destination port indicated in the identifier field
- `src port <port>`: When the packet is a Layer 4 protocol, such as TCP or UDP, this filter will capture packets to the source port indicated in the identifier field

The port-range matching filters are:

- `tcp portrange <p1>-<p2>` or `udp portrange <p1>-<p2>`: TCP or UDP packets in the port range of p1 to p2
- `tcp src portrange <p1>-<p2>` or `udp src portrange <p1>-<p2>`: TCP or UDP packets in the source port range of p1 to p2
- `tcp dst portrange <p1>-<p2>` or `udp src portrange <p1>-<p2>`: TCP or UDP packets in the destination port range of p1 to p2

How to do it...

Follow the instructions in the *Configuring capture filters* recipe, and configure filters as follows:

1. To capture packets to port 80 (HTTP), configure `dst port 80` or `dst port http`.
2. To capture packets to or from port 5060 (SIP), configure `port 5060`.
3. To capture packets to or from port 5060 (SIP), configure `port 5060`.
4. To capture the start (SYN flag) and end (FIN flag) packets of all TCP connections, configure `tcp[tcpflags] & (tcp-syn|tcp-fin) != 0`.

   Tip

   In `tcp[tcpflags] & (tcp-syn|tcp-fin) != 0`, it is important to note that this is a bitwise and operation, not a logical and operation. For example, 010 or 101 equals 111, and not 000.

5. To capture all TCP packets with RST (Reset) flag set to 1, configure `tcp[tcpflags] & (tcp-rst) != 0`.
6. Length filters are configured in the following way:
   - `less <length>`: It captures only packets with length less than or equal to length identifier. This is equivalent to `len <= <length>`.
   - `greater <length>`: It captures only packets with length greater than or equal to length identifier. This is equivalent to `<len >= length>`.

For example,

- `tcp portrange 2000-2500`
- `udp portrange 5000-6000`

Port range filters can be used for protocols that work in a range of ports rather than specific ones.

How it works…

Layer 4 protocols, mostly TCP and UDP, are the protocols that connect between end applications. The end node on one side (for example, a web client) sends a message to the other side (for example, a web server), requesting to connect to it. The codes of the processes that send the request and the processes that receive the request are called port numbers. Further discussion on this issue is provided in Chapter 9, *UDP/TCP Analysis*.

Both in TCP and UDP, the port numbers indicate the application codes. The difference between them is that TCP is a connection-oriented, reliable protocol, while UDP is a connectionless unreliable protocol. There is an additional Layer 4 protocol called **Stream Control Transport Protocol** (**SCTP**) that you can refer to as an advanced version of TCP, which also uses port numbers.

TCP flags are sent in packets in order to establish, maintain, and close connections. A signal is set when a specific bit in the packet is set to `1`. The most common flags that are in use are:

- **SYN**: A signal sent in order to open a connection
- **FIN**: A signal sent in order to close a connection
- **ACK**: A signal sent to acknowledge received data
- **RST**: A signal sent for immediate close of a connection
- **PSH**: A signal sent for pushing data for processing by the end process (application)

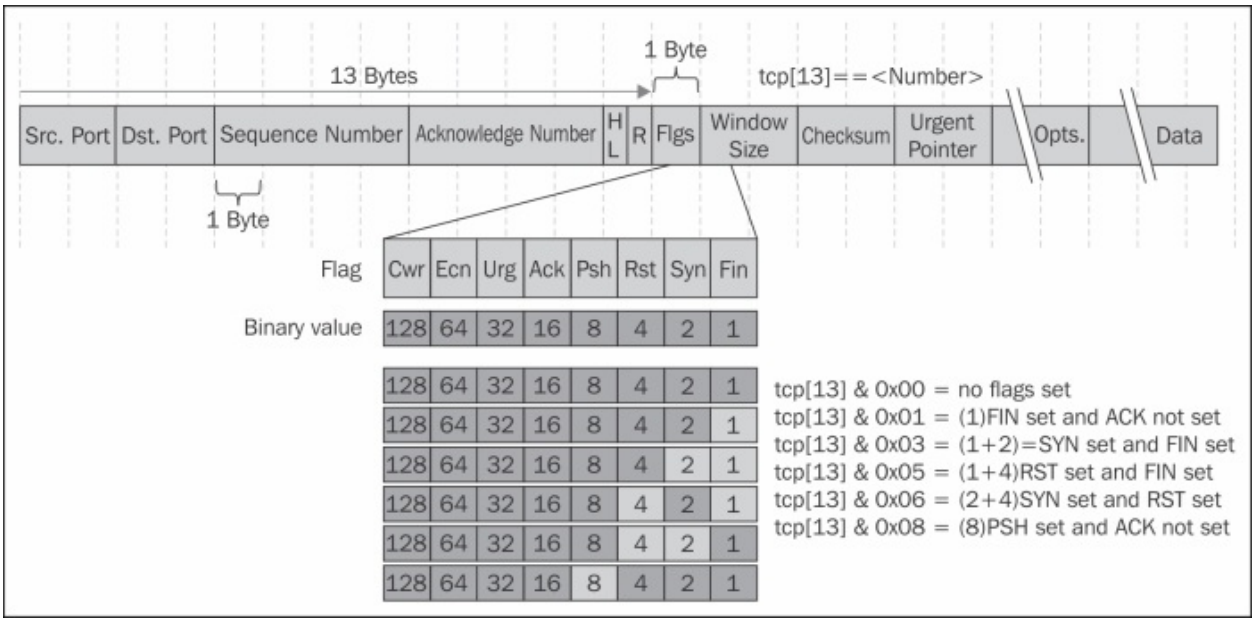
Using capture filters you can filter packets to/from specific applications, along with filtering packets with specific flags turned on.

There's more...

Some problematic scenarios (mostly attacks…) are:

- `tcp[13] & 0x00 = 0`: No flags set (null scan)
- `tcp[13] & 0x01 = 1`: **FIN** set and **ACK** not set
- `tcp[13] & 0x03 = 3`: **SYN** set and **FIN** set
- `tcp[13] & 0x05 = 5`: **RST** set and **FIN** set
- `tcp[13] & 0x06 = 6`: **SYN** set and **RST** set
- `tcp[13] & 0x08 = 8`: **PSH** set and **ACK** not set

In the following diagram you can see how it works. `tcp[13]` is the number of bytes from the beginning of the protocol header, when the values `0`,`1`,`3`,`5`,`6`, and `8` refer to the flag locations.



See also

A deeper description of UDP and TCP is provided in Chapter 9, *UDP/TCP Analysis*.

Configuring compound filters

Structure filters are simply made for writing filters out of several conditions. It uses simple conditions, such as `not`, `and`, and `or` for creating structured conditions.

Getting ready

Structured filters are written in the following format:

```
[not] primitive [and|or [not] primitive ...]
```

The following modifiers are commonly used in the Wireshark capture filters:

- `!` or `not`
- `&&` or `and`
- `||` or `or`

How to do it...

To configure structured filters, you simply write the conditions according to what we learned in the previous recipes, with conditions to meet your requirements.

Some common filters are:

1. For capturing only unicast packets, configure `not broadcast and not multicast`.
2. For capturing HTTP packets to www.youtube.com, configure `host www.youtube.com and port 80`.
3. A capture filter for telnet that captures traffic to and from a particular host, configures `tcp port 23 and host 192.180.1.1`.
4. For capturing all telnet traffic not from `192.168.1.1`, configure `tcp port 23 and not src host 192.168.1.1`.

How it works…

Some examples for structured filters:

For capturing data to tcp port 23 (Telnet) from source port range of 5000-6000, configure `tcp dst port 23 and tcp src portrange 5000-6000`.

There's more...

Some interesting examples are as follows:

- `host www.mywebsite.com and not (port 80 or port 23)`
- `host 192.168.0.50 and not tcp port 80`
- `host 10.0.0.1 and not host 10.0.0.2`

See also

For more examples, you can take a look at:

- http://www.packetlevel.ch/html/tcpdumpf.html
- http://www.packetlevel.ch/html/txt/tcpdump.filters

Configuring byte offset and payload matching filters

Byte offset and payload matching filters come to provide us with a flexible tool for configuring self-defined filters (filters for fields that are not defined in the Wireshark dissector and filters for proprietary protocols). By understanding the protocols that we work with and understanding their packet structure, we can configure filters that will watch a specific string in the captured packets, and filter packets according to it. In this recipe we will learn how to configure these types of filters, and we will also see some common and useful examples of the subject.

Getting ready

To configure byte offset and payload matching filters, start Wireshark and follow the instructions in the *Configuring capture filters* recipe in the beginning of this chapter.

How to do it...

1. String matching filters comes to check a specific string in the packet header. It comes in the following format:

   ```
   proto [Offset: bytes]
   ```

With this filter we can create filters for strings over IP, TCP, and UDP.

- For IP string-matching filters you can create the following filter:

```
ip [Offset:Bytes]
```

- For matching application data, that is, to look into the application data that is carried by TCP or UDP, the most common uses of it are: `tcp[Offset:Bytes]` Or `udp[Offset:Bytes]`.

How it works…

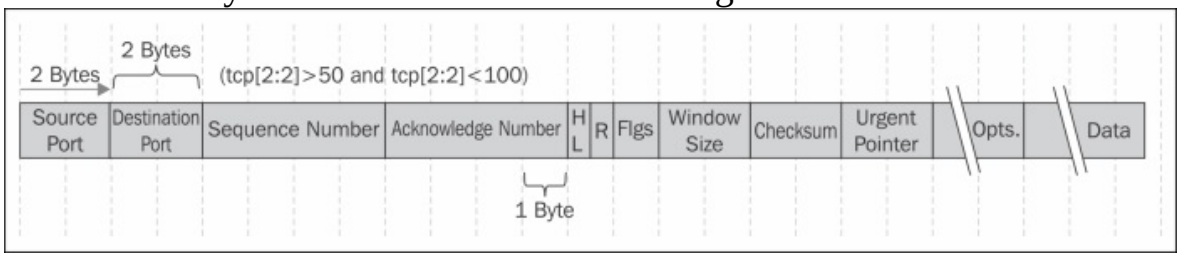
The general structure of offset filter is:

```
proto [Offset in bytes from the start of the header : Number of
bytes to check]
```

Common examples for string matching filters are:

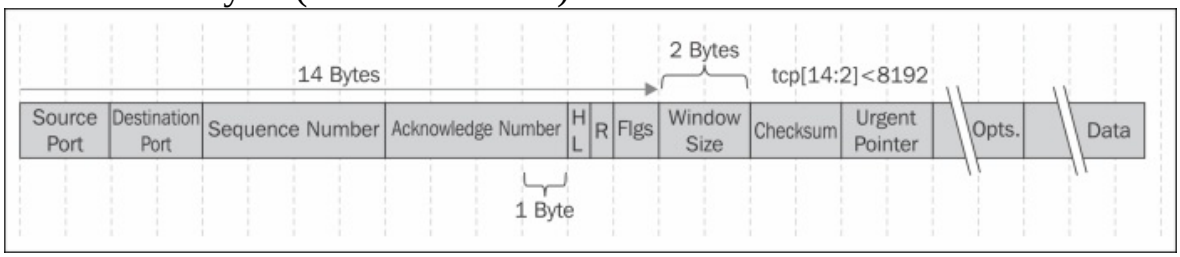
1. For filtering destination TCP ports between 50 and 100, configure `(tcp[2:2] > 50 and tcp[2:2] < 100)`.

   Here we count two bytes from the beginning of the TCP header, and check the next two bytes to be lower than 100 and higher than 50.



2. For checking TCP window size smaller then 8192, configure `tcp[14:2] < 8192`.

   Here we count two bytes from the beginning of the TCP header, and check the next two bytes (the window size) to be less than 8192.



   There's a nice string-matching capture filter generator in http://www.wireshark.org/tools/string-cf.html

There's more...

You can also see additional filters in the `tcpdump` man pages:

1. To print all IPv4 HTTP packets to and from port 80, (that is to print only packets that contain data, not, for example, SYN, FIN or ACK-only packets), configure the following filter: `tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)`.
2. To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host, configure `tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net <localnet>`.
3. To print IP broadcast or multicast packets that were not sent via Ethernet broadcast or multicast, configure `ether[0] & 1 = 0 and ip[16] >= 224`.
4. To print all ICMP packets that are not echo requests/replies (that is, not ping packets), configure `icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply`.

See also

- There is a string calculator at http://www.wireshark.org/tools/string-cf.html.

  It doesn't always provide working results, but it might be a good place to start from.
- Another interesting blog can be found on http://www.packetlevel.ch/html/txt/byte_offsets.txt.

Chapter 3. Using Display Filters

In this chapter you will learn the following:

- Configuring display filters
- Configuring Ethernet, ARP, host, and network filters
- Configuring TCP/UDP filters
- Configuring specific protocol filters
- Configuring substring operator filters
- Configuring macros

Introduction

In this chapter we will learn how to work with display filters. Display filters are filters that we apply after capturing data (filtered by capture filters or not), and when we wish to display only part of the data.

Display filters can be implemented in order to locate various types of data:

- Parameters such as the IP address, TCP or UDP port numbers, URLs, and server names
- Conditions such as "packet length shorter than..." and the TCP port range
- Phenomena such as TCP retransmissions, duplicate and other types of ACKs, various protocol error codes, and flag existence
- Various applications parameters such as Short Message Service (SMS) source and destination numbers and Server Message Block (SMB) server names

Any data that is sent over the network can be filtered, and when filtered, you can create statistics and graphs according to it.

As we will describe in the recipes in this chapter, there are various ways to configure display filters: from predefined menus, from the packet pane, or by writing the syntax directly.

Tip

While using display filters, don't forget that all the data was already captured and the display filters only decide what to display. Therefore, after filtering data, the capture file still contains the original data that was captured. You may later save the complete data or only the displayed data.

Configuring display filters

In order to configure display filters, you can choose one of the several options:

- Choosing from the filters menus
- Writing the syntax directly into the display filter window (while working with Wireshark; after a while this will become your favorite)
- Choosing a parameter in the packet pane and defining it as a filter
- Using `tshark` or `wireshark` with command line ; this will be discussed in Appendix

This chapter discusses the first three options.

Getting ready

In general, a display filter string takes the form of a series of primitive expressions connected by conjunctions (and, or, or something else) and optionally preceded by `not`: `[not] Expression [and|or] [not] Expression...`

While `Expression` can be any filter expression, such as `ip.src==192.168.1.1` for the source address, `tcp.flags.syn==1` for TCP SYN flag presence, and `tcp.analysis.retransmission` for TCP retransmissions, `and|or` are conjunctions that can be used in any combinations of expression, including brackets, multiple brackets, and any lengths of strings.

There are several conditions to these. They can be one of the following:

C-like Syntax	Shortcut	Description	Example
==	**eq**	Equal	`ip.addr == 192.168.1.1` or `ip.addr eq 192.168.1.1`
!=	**ne**	Not equal	`!ip.addr==192.168.1.1`, `ip.addr != 192.168.1.1`, or `ip.addr ne 192.168.1.1`
>	**gt**	Greater than	`frame.len > 64`
<	**lt**	Less than	`frame.len < 1500`
>=	**ge**	Greater than or equal to	`frame.len >= 64`
<=	**le**	Less than or equal to	`frame.len <= 1500`
	is present	A parameter is present	`http.response`
	contains	Contains a string	`http.host contains cisco`
	matches	A string matches the	`http.host matches www.cisco.com`

		condition	

You can insert a space character between parameters and operators or leave it without spaces.

Wireshark colorizes the display filter area in yellow whenever you use the `!=` operator for combined expressions such as `eth.addr`, `ip.addr`, `tcp.port`, and `udp.port`, but this will not work due to the following reason.

When you type a filter expression such as `ip.addr != 192.168.1.100`, you will see **The packet contains the field ip.addr with a value different from 192.168.1.100**. Because an IP datagram contains both a source and a destination address, the expression will evaluate to `true` whenever at least one of the two addresses differs from 192.168.1.100. For this reason you should write `!(ip.addr == 192.168.1.100)`; this will display **Show me all the packets for which it is not true that a field ip.addr have the value of 1.2.3.4**.

There are several operators. They can be as follows:

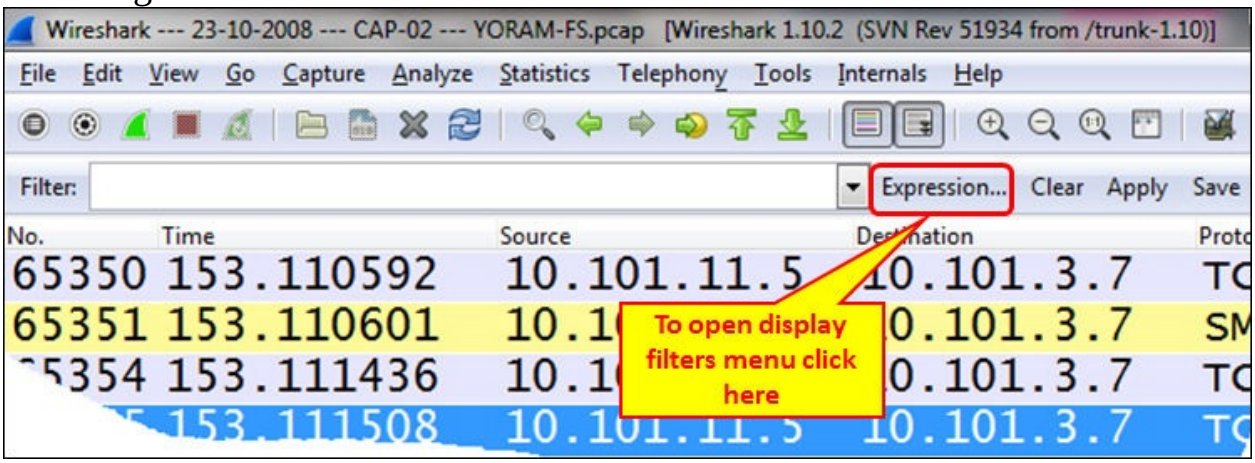
C-like Syntax	Shortcut	Description	Example
`&&`	`and`	Logical AND	`ip.src==10.0.0.1 and tcp.flags.syn==1` All SYN flags sent from IP address 10.0.0.1 practically and all connections opened (or tried to be opened) from 10.0.0.1.
`||`	`or`	Logical OR	`ip.addr==10.0.0.1 or ip.addr==10.0.02` All the packets going in or out the two IP addresses.
`!`	`not`	Logical NOT	`not arp and not icmp` All the packets that are neither ARP nor ICMP.

How to do it...

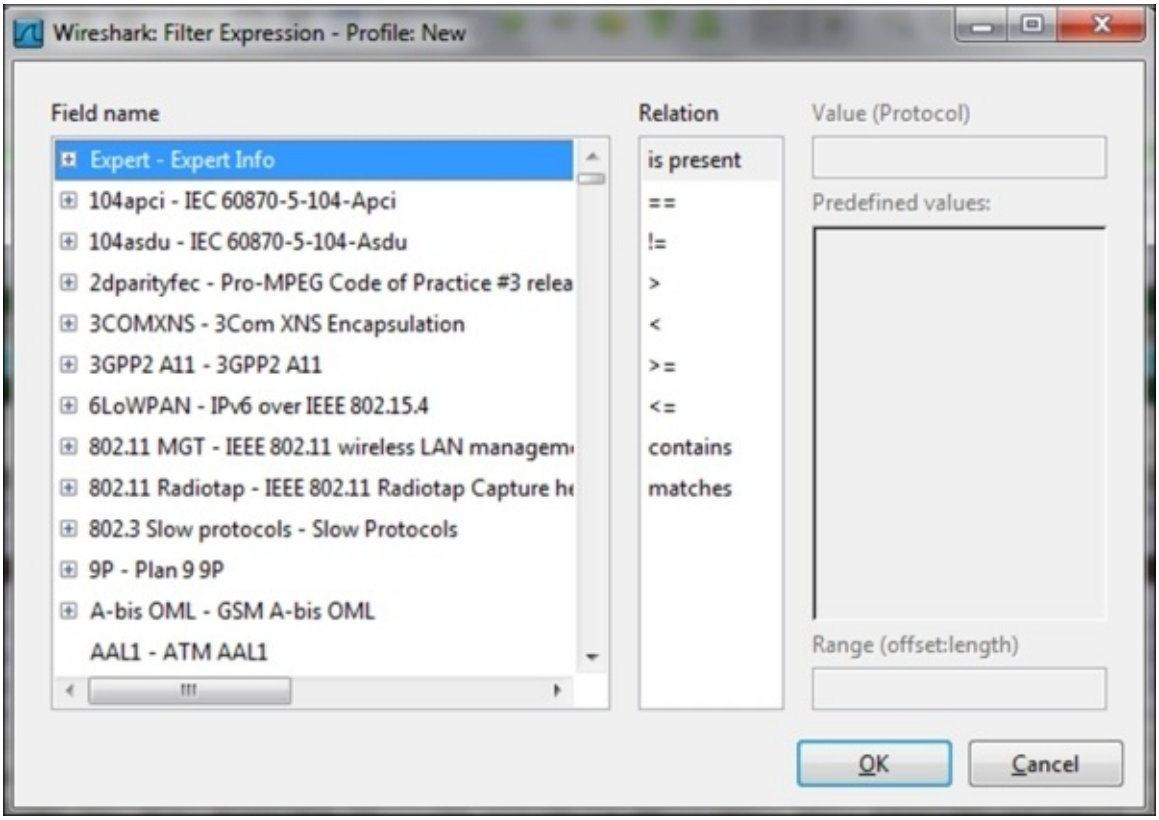
To configure display filters, you can choose any one of the methods mentioned earlier.

Choosing from the filters menu

For choosing from the filters menu, navigate to the display filter pane on the upper side of the window and click on the **Expression...** button as you see in the following screenshot:



When you click on the **Expression...** button, the following window will open:



There are five important panes in the filters menu:

- **Field name**: In this pane you configure the filter parameter. You can go to the protocol by typing its name, and get to the protocol parameter by clicking on the + sign to the left of the list.

  One example for this would be: type `ipv4` to get to the **IPv4** protocol, click on the + sign to expand the protocol parameters (or press *Enter* twice) and choose **ip.addr** to filter a specific IP address.

  Another example would be to type `tcp` to get to the **TCP** protocol, click on the + sign to the left of the protocol parameter and choose **tcp.port** for the source or destination port number.
- **Relation**: This is the pane from where you choose the operator. You can choose == for equal, `!=` for not equal, and so on.

  An example for this would be: type `sip` to get to the **SIP** protocol, choose

sip.Method, and choose == from the **Relation** pane. Type `invite` in the **Value (Protocol)** pane. This will filter all the SIP INVITE methods.

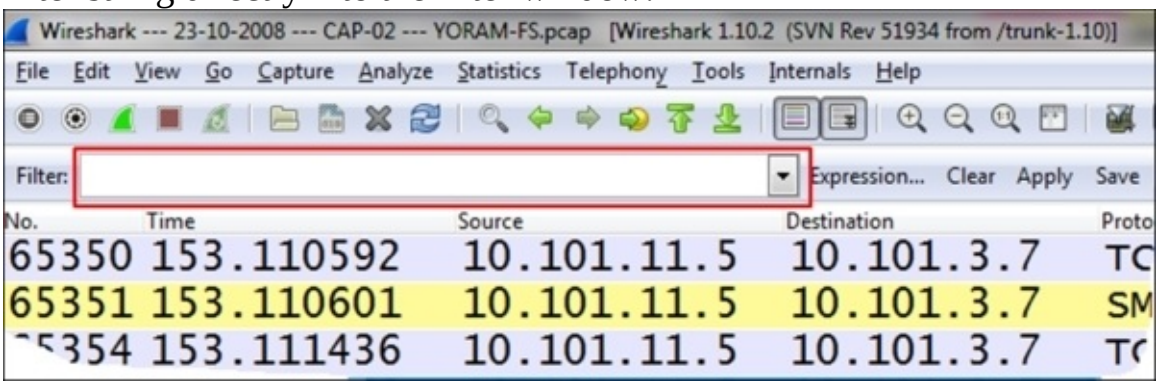
- **Value**: Here you enter the value of the field that you have chosen before.

  An example for this would be: type `tcp` to get to the **TCP** protocol, click on the + sign to go to the protocol parameter, choose **tcp.flags.syn** for the TCP SYN flag, and enter `1` in the **Value** field.
- **Predefined values**: When the value of the field you chose is not Boolean, there might be a list of options in this field.

  An example for this would be: under **TCP**, there is an option named **tcp.option_kind**. This option is related to **TCP** options (for more details, refer to Chapter 9, *UDP/TCP Analysis*). You will get a list of values that are possible.
- **Range (offset: length)**: This field provides you the length of the string in the `offset:length` format.

Writing the syntax directly into the display filter window

After you get used to the display filters syntax, you may find it easier to type the filter string directly into the filter window:



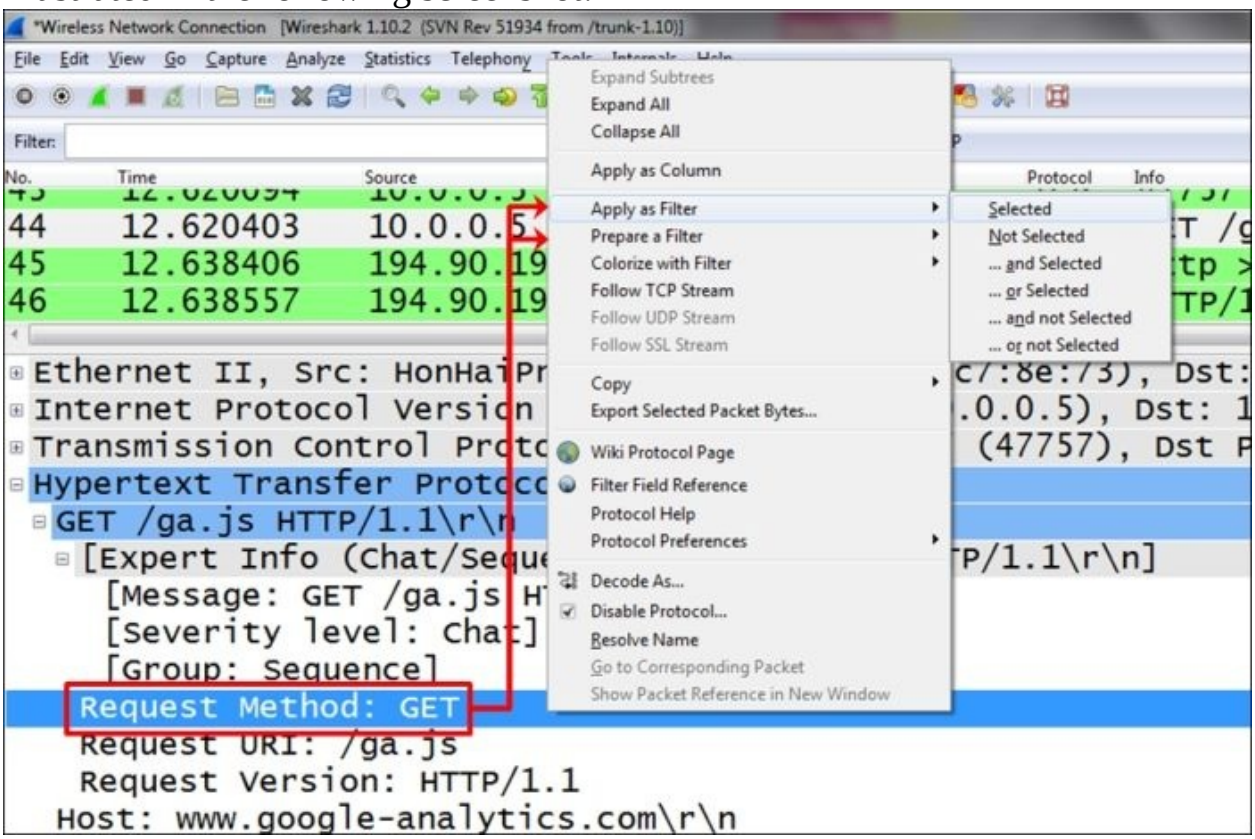
In this case, when you write a filter string into the window, the window will light up in one of the following three colors:

- **Green**: This is when the filter is correct and you can apply it.
- **Red**: This indicates a wrong string. Fix the string before you apply it.
- **Yellow**: Whenever you use the `!=` operator, the display filter area will turn

yellow. It doesn't mean your filter will not work, it is just a warning that it *may not* work.

Choosing a parameter in the packet pane and defining it as a filter

This is a very convenient option. You can choose any field from the packet detail pane in the captured file; right-click on it and you will get a few options, as illustrated in the following screenshot:



A couple of options are as follows:

- **Apply as Filter**: This will set a filter according to the field you choose and apply it to the captured data.
- **Prepare a Filter**: This will prepare a filter but not apply it. It will be applied when you click on the **Apply** button on the right-hand side of the filter window.

In both the options, you can choose to configure a filter:

- **Selected**: This will choose the selected field and parameter
- **Not Selected**: This will choose the the field and parameter that are not selected

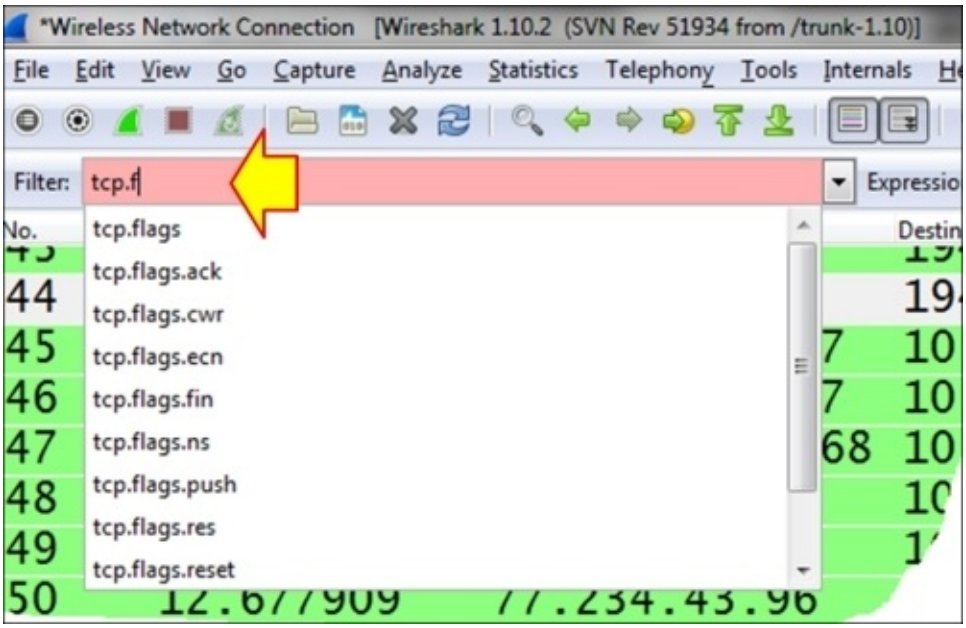
For example, right-clicking on the field `http.request.method` and choosing **Selected** will come with the filter string `http.request.method == GET`; while, choosing **Not Selected** will come with the string `!(http.request.method == "GET")`.

You can also choose the options **... and selected**, **... or selected**, **... and not selected**, or **... or not selected** for structured filters.

How it works...

The display filter is a proprietary Wireshark language. There are many places where display filters can be used that will be discussed in the later chapters. Additional filters will be introduced in the upcoming recipes of this chapter.

You can always use the autocomplete feature to complete filter strings. For example, if you type in `tcp.f`, as shown in the following screenshot, the autocomplete feature lists possible display filter values that can be created beginning with `tcp.f`, that is, TCP flags (SYN, FIN, RST, and so on).

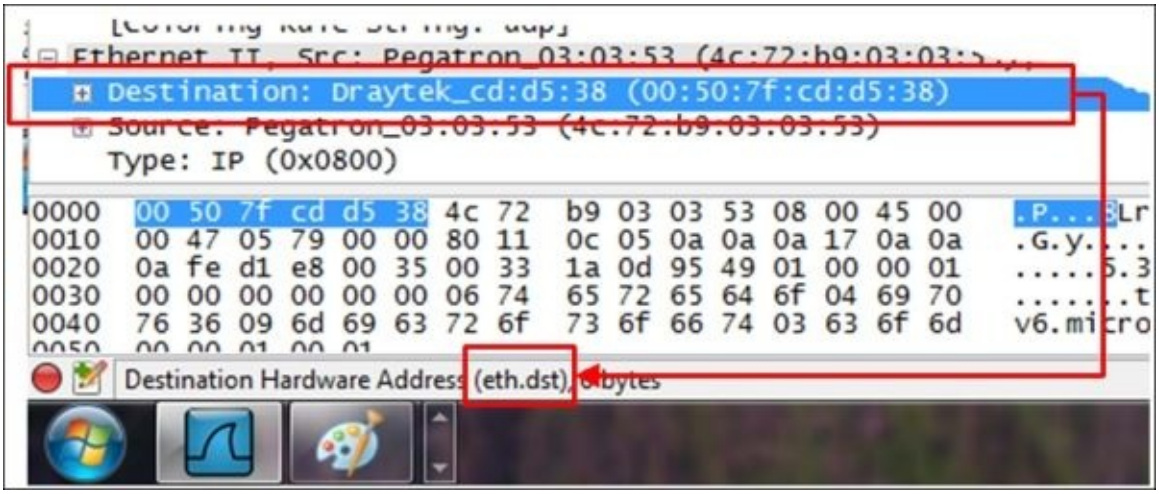


There's more...

Now we will cover some additional helpful features.

What is the parameter we filter?

Anytime you mark a specific field in the packet details pane, you will see the correlating filter string in the status bar at the bottom-left corner of the Wireshark window.

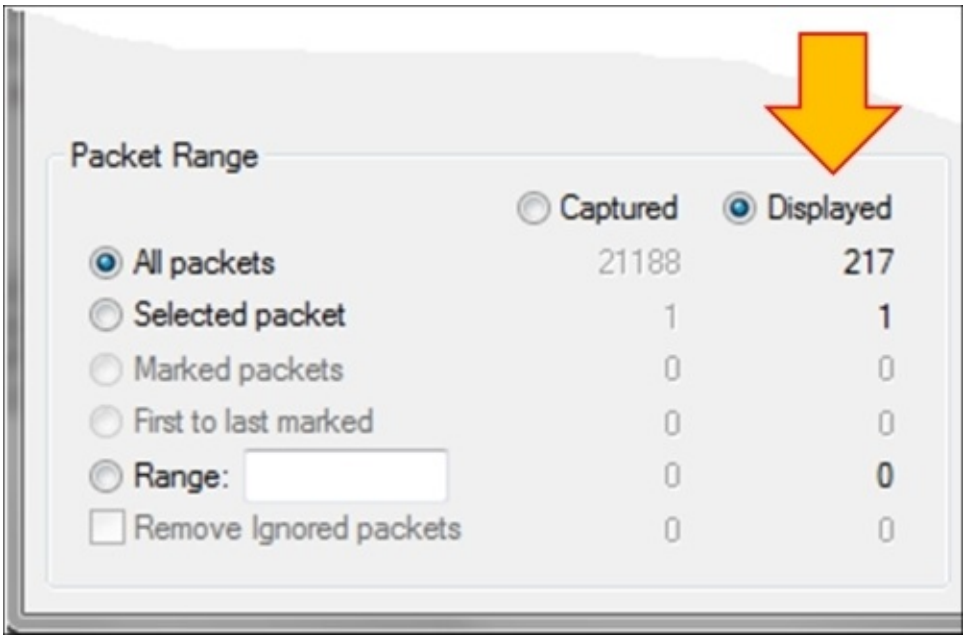


Adding a parameter column

You can also right-click on a parameter in the packet pane and choose **Apply as Column**. This will add a column with the specific parameter. For example, you can choose the parameter `tcp.window_size_value` and add it as a column to the packet list pane, so you will be able to watch the TCP window size online. This influences TCP performance, as we will learn in Chapter 9, *UDP/TCP Analysis*.

Saving the displayed data

To save the displayed data, you can navigate to **File** | **Export Specified Packets...** and choose which packets to save.


Packet Range
Captured
Displayed
All packets
21188
217
Selected packet
1
1
Marked packets
0
0
First to last marked
0
0
Range:
0
0
Remove Ignored packets
0
0

Configuring Ethernet, ARP, host, and network filters

In this recipe we will discuss how to configure filters of layers 2 and 3, that is, Ethernet- and IP-based filters respectively. We will also discuss **Address Resolution Protocol** (**ARP**) filters.

Getting ready

In layer 2 we will configure Ethernet-based filters, while in layer 3 we will configure IP-based filters. In Ethernet we have filters based on the Ethernet frame and the MAC address, while in IP we have filters based on the IP packet and address.

The common frame delta filters are as follows:

- `frame.time_delta`: This is used for the time delta between the current and previously captured frames; this will be used in statistical graphs displayed in Chapter 5, *Using Advanced Statistics Tools*
- `frame.time_delta_displayed`: This is used for the time delta between current and previously displayed frames; this will be used in statistical graphs displayed in Chapter 5, *Using Advanced Statistics Tools*

Since the time between frames can influence TCP performance significantly, we will use the `frame.time_delta` parameters in statistical graphs for monitoring TCP performance.

The common layer 2 (Ethernet) filters are as follows:

- `eth.addr == <MAC Address>`: This is used to display a specific MAC address
- `eth.src == <MAC Address>`: This is used to get the source MAC address
- `eth.dst == <MAC Address>`: This is used to get the destination MAC address
- `eth.type == <Protocol Type (Hexa)>`: This is used to get the Ethernet protocol types

The common ARP filters are as follows:

- `arp.opcode == <value>`: This is used for ARP requests/responses
- `arp.src.hw_mac == <MAC Address>`: This is used to capture the ARP address of the sender

The common layer 3 (IP) filters are as follows:

- `ip.addr == <IP Address>`: This is used to get the source or destination IP

address

- `ip.src == <IP Address>`: This is used to get the source IP address
- `ip.dst == <IP Address>`: This is used to get the destination IP address
- `ip.ttl == <value>`, `ip.ttl < value>`, or `ip.ttl > <value>`: This is used to get IP TTL (Time To Live) values
- `ip.len = <value>`, `ip.len > <value>`, or `ip.len < <value>`: This is used to get IP packet length values
- `ip.version == <4/6>`: This is used to get the IP protocol version (Version 4 or Version 6)

How to do it...

Here we will see some common examples for layer 2 and 3 filters.

Address format	Syntax	Example
Ethernet (MAC) address	`eth.addr == xx:xx:xx:xx:xx:xx` Here, `x` = 0 to f.	`eth.addr == 00:50:7f:cd:d5:38`
	`eth.addr == xx-xx-xx-xx-xx-xx` Here, `x` = 0 to f.	`eth.addr == 00-50-7f-cd-d5-38`
	`eth.addr == xxxx.xxxx.xxxx` Here `x` = 0 to f.	`eth.addr == 0050.7fcd.d538`
Broadcast MAC address	`Eth.addr == ffff.ffff.ffff`	
IPv4 host address	`ip.addr == x.x.x.x` Here, `x` = 0 to 255.	`Ip.addr == 192.168.1.1`
IPv4 network address	`ip.addr == x.x.x.x/y` Here `x` = 0 to 255, `y` = 0 to 32.	`ip.addr == 192.168.200.0/24` This covers all the addresses in the network 192.168.200.0 mask 255.255.255.0.

IPv6 host address	`ipv6.addr == x:x:x:x:x:x:x:x`
	
	`ipv6.addr == x::x:x:x:x`
	
	Here in the format of nnnn, `n` = 0 to f (Hex).
		`ipv6.addr == fe80::85ab:dc2e:ab12:e6c7`

IPv6 network address	`ipv6.addr == x::/y` Here `x` = 0 to f (Hex) and `y` = 0 to 128.	`ipv6.addr == fe80::/16` This covers all the addresses that start with the 16 bits `fe80`.

The table refers to `ip.addr` and `ipv6.addr` filter strings. The value for any field that has an IP address value can be written the same way.

Ethernet filters

These are classified into two categories:

- To display only packets sent from or to specific MAC addresses, use something like these: `eth.src == 10:0b:a9:33:64:18` and `eth.dst == 10:0b:a9:33:64:18`
- To display only broadcasts, use `Eth.dst == ffff.ffff.ffff`

ARP filters

These are classified into two categories:

- To display only ARP requests, use `arp.opcode == 1`
- To display only ARP responses, use `arp.opcode == 2`

IP and ICMP filters

- To display only packets from a specific IP address, use something like this: `ip.src == 10.1.1.254`
- To display only packets that are not from a specific address, use something like this: `!ip.src == 64.23.1.1`
- To display only packets between two hosts, use something like these: `ip.addr == 192.168.1.1` and `ip.addr == 200.1.1.1`
- To display only packets that are sent to multicast IP addresses, use something like this: `ip.dst == 224.0.0.0/4`
- To display only packets coming from the network 192.168.1.0/24 (mask 255.255.255.0), use `ip.src==192.168.1.0/24`
- To display only IPv6 packets to/from specific addresses, use something like the following:
  - `ipv6.addr == ::1`
  - `ipv6.addr == 2008:0:130F:0:0:09d0:666A:13ab`
  - `ipv6.addr == 2006:0:130f::9c2:876a:130b`
  - `ipv6.addr == ::`

Complex filters

- To check for packets sent from the network 10.0.0.0/24 to a website that

contains the word `packt`, use `ip.src == 10.0.0.0/24` and `http.host contains "packt"`

- To check for packets sent from the network 10.0.0.0/24 to websites that end with `.com`, use `ip.addr == 10.0.0.0/24` and `http.host matches "\.com$"`
- To check for all the broadcasts from the source IP address 10.0.0.0, use `ip.src == 10.0.0.0/24` and `eth.dst == ffff.ffff.ffff`
- To check for all the broadcasts that are not ARP requests, use `not arp` and `eth.dst == ffff.ffff.ffff`
- To check for all the packets that are not ICMP, use `!arp || !icmp`, and to check for all the packets that are not ARP, use `not arp or not icmp`

How it works...

Here are some explanations to the filters we saw in the *How to do it...* section of this recipe.

Ethernet broadcasts

In Ethernet, broadcasts are packets that are sent to addresses with all 1s in the destination field and this is why, to find all broadcasts in the network, we insert the filter `eth.dst == ffff.ffff.ffff`.

IPv4 multicasts

IPv4 multicasts are packets that are sent to an address in the address range 224.0.0.0 to 239.255.255.255 that is in binary of the address range 11100000.00000000.00000000.00000000 to 11101111.11111111.11111111.11111111.

If you look at the binary representation, a destination multicast address is an address that starts with three 1s and a 0, and therefore, a filter to IPv4 multicast destinations will be `ip.dst == 224.0.0.0/4`. That is, an address that starts with four 1s (224), and a subnet mask of 4 bits (`/4`) will indicate a network address ranger from 224 to 239 that will filter multicast addresses.

IPv6 multicasts

IPv6 multicasts are packets that are sent to an address that starts with ff (first two hex digits = `ff`), then one-digit flags, and scope. Therefore when we write the filter `ipv6.dst == ff00::/8`, it means to display all the packets in IPv6 that are sent to an address that starts with the string `ff`, that is, IPv6 multicasts.

See also

- For more information on Ethernet, refer to Chapter 7, *Ethernet, LAN Switching, and Wireless LAN*

Configuring TCP/UDP filters

TCP and UDP are the main protocols in layer 4 that provide connectivity between end applications. Whenever you start an application from one side to another, you start the session from a source port, usually a random number equal or higher than 1,024, and connect to a destination port, which is a well-known or registered port that waits for the session on the other side. These are the port numbers that identify the application that works over the session.

Other types of filters refer to other fields in the UDP and TCP headers. In UDP we have a very simple header with very basic data, while in TCP we have a more complex header that we can get much more information from.

In this recipe we will concentrate on the possibilities while configuring TCP and UDP display filters.

Getting ready

As done earlier, we should plan precisely what we want to display and prepare the filters accordingly.

For TCP or UDP port numbers use the following display filters:

- `tcp.port == <value>` or `udp.port == <value>`: This is used for specific TCP or UDP ports (source or destination)
- `tcp.dstport == <value>` or `udp.dstport == <value>`: This is used for specific TCP or UDP destination ports
- `tcp.srcport == <value>` or `udp.srcport == <value>`: This is used for specific TCP or UDP destination ports

In UDP, the header structure is very simple: source and destination ports, packet length, and checksum. Therefore, the only significant information here is the port number.

TCP on the other hand is more complex and uses connectivity and reliability mechanisms that can be monitored by Wireshark. Using `tcp.flags`, `tcp.analysis`, and other smart filters will help you resolve performance problems (retransmissions, duplicate ACKs, zero windows, and so on), protocol operation issues such as resets, half-opens, and so on.

Common display filters in this category are as follows:

- `tcp.analysis`: This is used for TCP analysis criteria such as retransmission, duplicate ACKs, or window issues. Examples for this filter are as follows (you can use the autocomplete feature to get the full list of available filters):
  - `tcp.analysis.retransmission`: This is used to display packets that were retransmitted.
  - `tcp.analysis.duplicate_ack`: This is used to display packets that were acknowledged several times.
  - `tcp.analysis.zero_window`: This is used to display packets when a device on the connection end sends a zero-window message (that tells the sender to stop sending data on this connection, until window size increases again).

Tip

The `tcp.analysis` filters do not analyze the TCP header; they provide a protocol analysis through the Wireshark expert system.

- `tcp.flags`: These are used to find out if a flag(s) is set or not. Examples of this filter are as follows:
  - `tcp.flags.syn == 1`: This is used to check if the SYN flag is set.
  - `tcp.flags.reset == 1`: This is used to check if the RST flag is set.
  - `tcp.flags.fin == 1`: This is used to check if the FIN flag is set.

Tip

For TCP flags, the `tcp.flags` filter will be used to find out whether a specific flag is set or not.

- `tcp.window_size_value < <value>`: This is used to look for small TCP window sizes that are in some cases indications for slow devices.

How to do it...

Some examples for filters in TCP/UDP filters:

- To filter all the packets to the HTTP server, use `tcp.dstport == 80`
- To filter all the packets from the network 10.0.0.0/24 to the HTTP server, use `ip.src==10.0.0.0/24` and `tcp.dstport == 80`
- For all the retransmissions in a specific TCP connection, use `tcp.stream eq 16 && tcp.analysis.retransmission`

To isolate a specific connection, place the mouse on a packet in the connection you want to watch, right-click on it, and choose **Follow TCP Stream**. A TCP stream is the data that is transferred between the two ends of the connection from the connection establishment to the connection tear down. The string `tcp.stream eq <value>` will appear in the display filter window. This is the stream you can work on now. In the preceding example, it came out as stream `16`, but it can be any stream number (starting the count from stream `1` in the capture file).

Retransmissions are TCP packets that are sent again. It can be due to several reasons, as explained in Chapter 9, *UDP/TCP Analysis*.

Tip

While monitoring phenomena such as retransmissions, duplicate ACKs, and others that influence performance, it is important to remember that these phenomena refer to a specific TCP connection.

Other examples of the types of TCP filters are as follows:

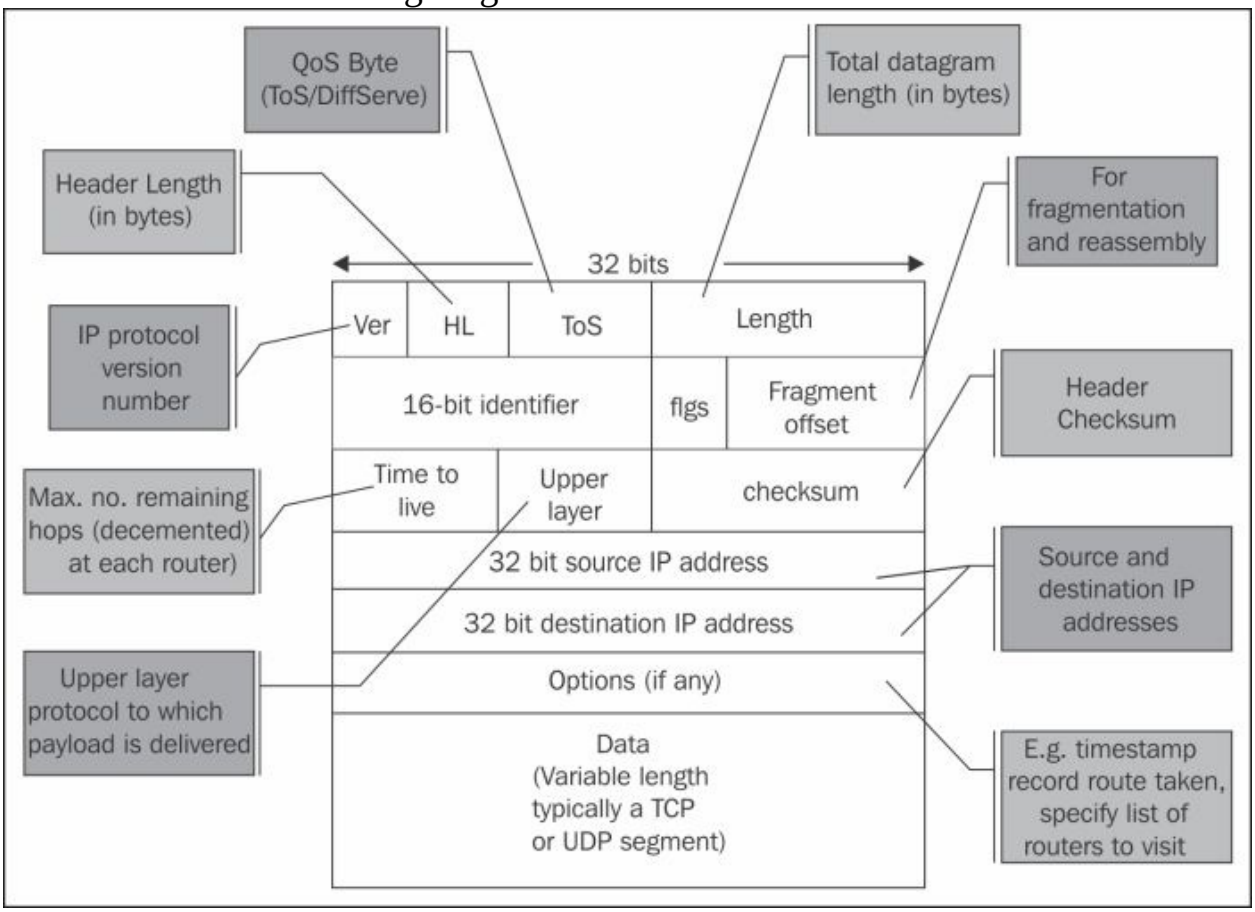
- To transfer all the window problems in a specific connection:
  - `tcp.stream eq 0 && (tcp.analysis.window_full || tcp.analysis.zero_window)`
  - `tcp.stream eq 0 and (tcp.analysis.window_full or tcp.analysis.zero_window)`
- To transfer all the packets from 10.0.0.5 to the DNS server: `ip.src == 10.0.0.5 && udp.port == 53`
- To transfer all the packets or protocols in TCP (for example HTTP) that

contains the string `cacti` (case sensitive): `tcp contains "cacti"`

- To check all the packets that are retransmitted from 10.0.0.3: `ip.src == 10.0.0.3 and tcp.analysis.retransmission`
- To transfer all the packets to any HTTP server: `tcp.dstport == 80`
- To check all the connections opened from a specific host (if in a form of scan, can be a worm!): `ip.src==10.0.0.5 && tcp.flags.syn==1 && tcp.flags.ack==0`
- To check all the cookies sent from and to a client: `ip.src==10.0.0.3 && (http.cookie || http.set_cookie)`

How it works...

The following are illustrations of the IP and TCP header structures respectively. UDP is quite simple; it has only source and destination port numbers, length, and checksum. In the following diagram we see the IP header structure:



Some important factors in the IP packet are as follows:

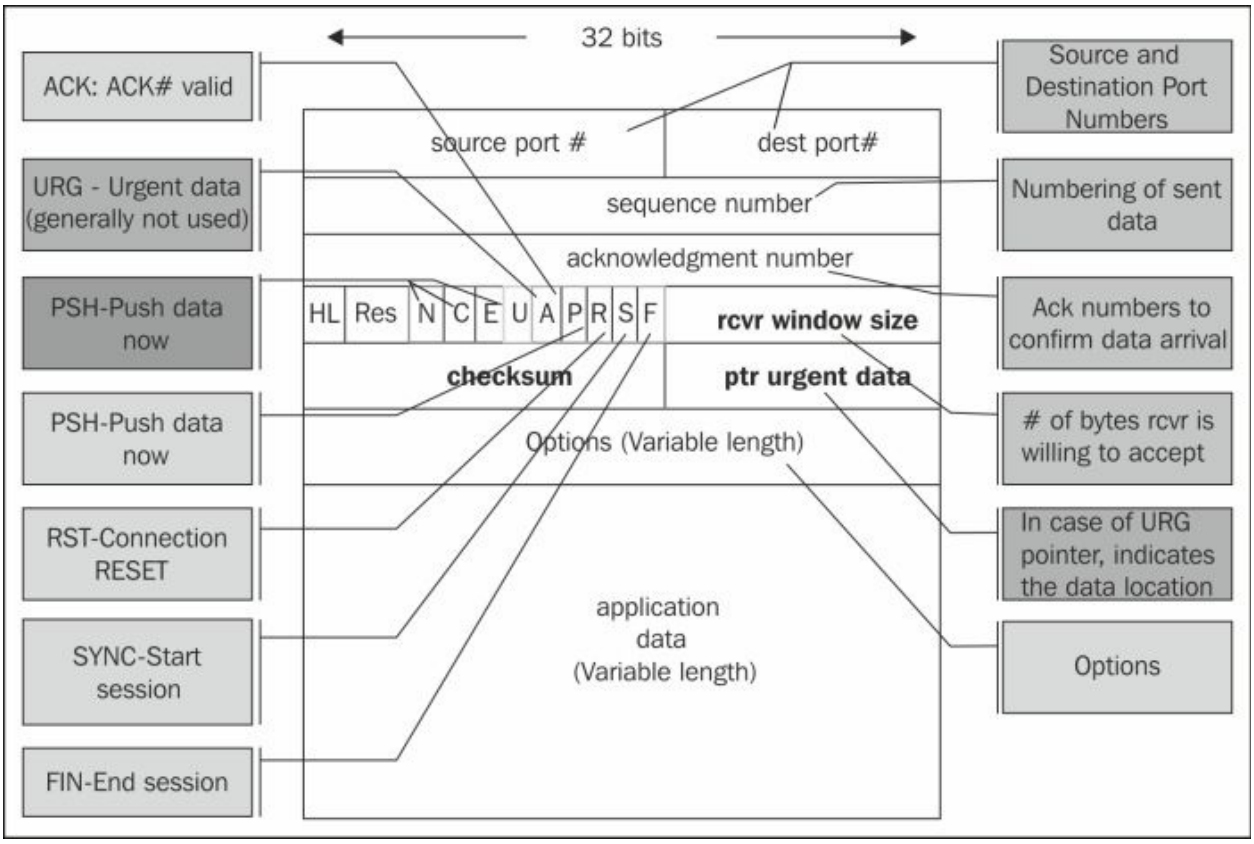
- **Ver**: The version is either 4 or 6.
- **Header length (HL)**: The header length is 20 to 24 bytes, with options.
- **Type of Service (ToS)**: This is usually implemented with **Differentiated Services** (**DiffServ**) and provides priority to preferred services.

  Tip

  IP standard (RFC 791 from September 1981) has named this field Type of

Service (ToS) and defined its structure. The standards for Differentiated Services were published later (RFCs 2474, 2475 from December 1998 and others) and are used for the implementation of the ToS byte in majority of the applications.

- **Length**: This field indicates the total datagram length in bytes.
- **16-bit identifier, flgs, and Fragment offset**: Every packet has it's own packet ID. When fragmented along with the flags and offset, these will enable the receiver to reassemble it.
- **Time to live (TTL)**: This starts with 64, 128, or 256 (depending on the operation system that sends the packet), when every router on the way decrements the value by one. This comes to prevent packets from traveling endlessly through the network. The router that sees 1 in the packet decrements it to 0 and drops the packet.
- **Upper layer**: This field consists of upper-layer protocols such as TCP, UDP and ICMP.
- **Checksum**: This field represents the packet checksum. The idea here is that the sender uses an error-checking mechanism to calculate a value over the packet. This value is set in the checksum field while the receiver of the packet calculates it again. If the sent value is not equal to the received value, it will be considered as a checksum error.
- **32-bit source and destination IP addresses**: As the names suggest, these are source and destination IP addresses.
- **Options**: This field is usually not in use in IPv4. In the following diagram you see the TCP header:



Some important factors in the TCP packet are as follows:

- **Source and destination ports**: These are the applications codes on either end.
- **Sequence number**: This field counts the bytes that the sender sends to the receiver.
- **Acknowledgement number**: This field indicates the ACK's received bytes. We will discuss this in detail in Chapter 9, *UDP/TCP Analysis*.
- **HL**: This is the header length field and it indicates whether we use the **Options** field or not.
- **Res**: This field is reserved for future flags.
- **Flags**: This field indicates flags to start a connection (SYN), close a connection (FIN), reset a connection (RST), and push data for fast processing (PSH). We will discuss this in detail in Chapter 9, *UDP/TCP Analysis*.
- **Rcvr window size**: This field indicates the buffer that the receiver has

allocated to the process.

- **Checksum**: This field indicates the packet checksum.
- **Options**: Timestamps and receiver window enhancement (RFC 1323), and MSS extension. **Maximum Segment Size** (**MSS**) is the maximum side of the TCP payload. It is indicated in this field. Further discussion on this will be done in Chapter 9, *UDP/TCP Analysis*.

There's more...

The TTL field in IP is quite a helpful field. While checking a TTL value, it explicitly indicates how many routers the packet has passed. Since operating system defaults are 64, 128, or 256, and the maximum number of hops that a packet will cross through the Internet are 30 (in private networks it is much less). For example, if we see a value of 120, the packet has passed 8 routers, and a value of 52 indicates that the packet has passed 12 routers.

See also

- For further information on the TCP/IP protocol stack, refer to Chapter 9, *UDP/TCP Analysis*

Configuring specific protocol filters

In this recipe we will have a look at the instructions and examples to configure display filters for common protocols such as DNS, HTTP, and FTP.

The purpose of this recipe is to learn how to configure filters that will help us in network troubleshooting. We will learn about network troubleshooting in the upcoming chapters.

Getting ready

To perform this recipe, you will need a running Wireshark software capture; there are no other prerequisites.

How to do it...

In this recipe we will see the display filters for some common protocols.

HTTP display filters

The following are some common HTTP display filters:

- To display all the HTTP packets going to `<"host name">`, use `http.request.method == <"Request methods">`
- To display packets with the HTTP `GET` method, use `http.request.method == "GET"`
- To display the URI requested by client, use `http.request.method == <"Full request URI">`; for example, `http.request.uri == "/v2/rating/mail.google.com"`
- To display the URI requested by the client that contains a specific string (all requests to Google in the preceding example), use `http.request.uri contains "URI String"`; for example, `http.request.uri contains "mail.google.com"`
- To check all the cookie requests sent over the network (note that cookies are always sent from the client to the server), use `http.cookie`
- To check all the cookie set commands sent from the server to the client, use `http.set_cookie`
- To check all the cookies sent by Google servers to your PC, use `(http.set_cookie) && (http contains "google")`
- To check all the HTTP packets that contain a ZIP file, use `http matches "\.zip" && http.request.method == "GET"`

DNS display filters

Here, we will look at some common DNS display filters.

To display DNS queries and responses, use:

- `dns.flags.response == 0 for DNS queries`
- `dns.flags.response == 1 for DNS response`

To display only DNS responses with four answers or more, use `dns.count.answers >= 4`.

FTP display filters

Some common FTP display filters are as follows:

- To fetch FTP request commands, use `ftp.request.command == <"requested command"> - ftp.request.command == "USER"`
- To fetch FTP commands from port 2, use `ftp`, and to fetch FTP data from port 20 or any other configured port, use `ftp-data`

How it works...

The Wireshark regular expression syntax for display filters uses the same syntax as regular expressions in Perl.

Some common modifiers are as follows:

- `^`: This is used to match the beginning of the line
- `$`: This is used to match the end of the line
- `|`: This is used for alternation purposes
- `()`: This is used for grouping purposes
- `*`: This is used to match either 0 or more times
- `+`: This is used to match 1 or more times
- `?`: This is used to match 1 or 0 times
- `{n}`: This is used to match exactly n times
- `{n,}`: This is used to match at least n times
- `{n,m}`: This is used to match at least n but not more than m times

You can use these modifiers to configure more complex filters. Have a look at the following examples:

- To look for HTTP `GET` commands that contain ZIP files, use `http.request.method == "GET" && http matches "\.zip" && ! (http.accept_encoding == "gzip, deflate")`
- To look for HTTP `GET` commands that contain ZIP files, use `http.request.method == "GET" && http matches "\.zip" && ! (http.accept_encoding == "gzip, deflate")`
- To look for HTTP messages that contain websites that end with `.com`, use `http.host matches "\.com$"`

See also

- The Perl regular expression syntax list can be found at http://www.pcre.org/, and the manual pages can be found at http://perldoc.perl.org/perlre.html

Configuring substring operator filters

Offset filters are filters in which you actually say, "Go to field x in the protocol header and check if the next y bytes equal to….".

These filters can be used in many cases in which a known string byte appears somewhere in the packet and you want to display packets that contain it.

Getting ready

To step through this recipe, you will need a running Wireshark software and a running capture; there are no other prerequisites. The general representation for offset filters is: Protocols[x:y] == <value> Here, x refers to the bytes from the beginning of the header and y refers to the number of bytes to check.

How to do it...

Examples for filters that use substring operators are as follows:

- **Packets to IPv4 multicast addresses**: `eth.dst[0:3] == 01:00:5e` (RFC 1112, section 6.4 allocates the MAC address space of 01-00-5E-00-00-00 to 01-00-5E-FF-FF-FF for multicast addressing)
- **Packets to IPv6 multicast addresses**: `eth.dst[0:3] == 33:33:00` (RFC 2464, section 7 allocates the MAC address space that starts with 33-33 for multicast addressing)

How it works...

Wireshark enables you to look into protocols and search for specific bytes in it. This is specifically practical for well-known strings in protocols, such as Ethernet in the given example.

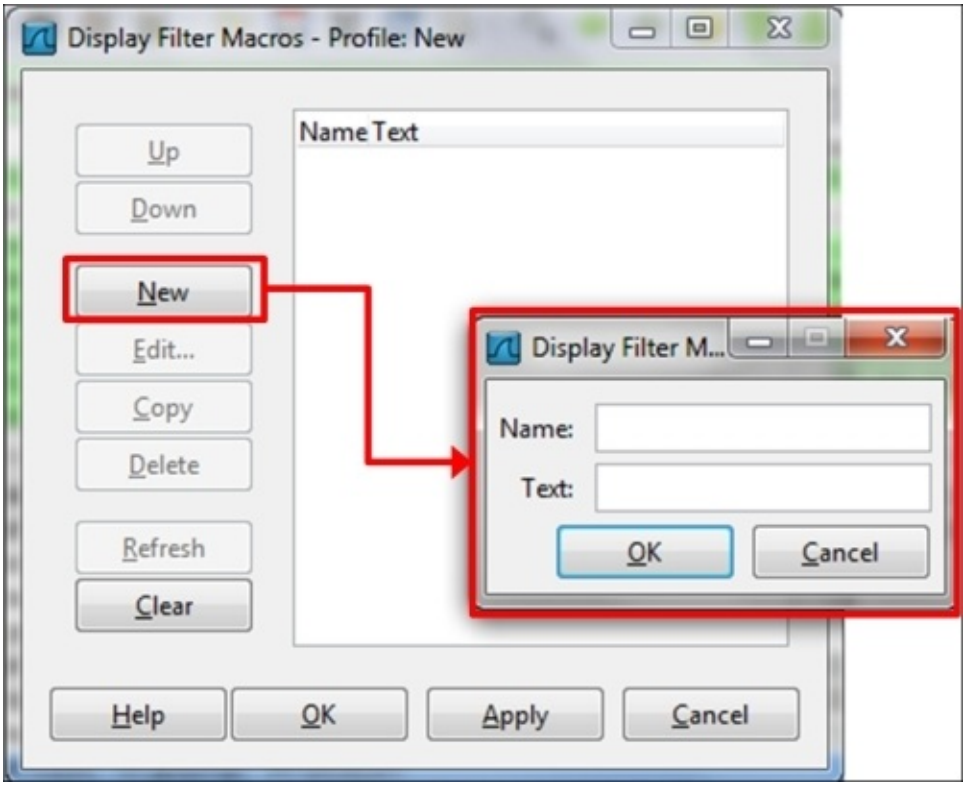
Configuring macros

Display filter macros are used to create shortcuts for complex display filters, which you can configure once and use later.

Getting ready

To configure display filter macros, navigate to **Analyze** | **Display Filter Macros** | **New**.

You will get the following window:



How to do it...

1. In order to configure a macro, you give it a name and fill the textbox with the filter string.
2. In order to activate the macro, you simply write `$(macro_name:parameter1;paramater2;parameter3 …)`.
3. Let's configure a simple filter name, `test01`, which takes the following parameters as values:
    - `ip.src == <value>`
    - `tcp.dstport == <value>`

This will be a filter that looks for packets from a specific source network that go out to the HTTP port.

A macro that takes these two parameters will be: `ip.src==$1 && tcp.dstport==$2`.

- Now, in order to get the filter results for the parameters `ip.src == 10.0.0.4` and `tcp.dstport == 80`, we should write the string `${test01:10.0.0.4;80}` in the display window bar.

How it works...

Macros work in a simple way; you write a filter string with the sign `$` ahead of every positional parameter. While running the macros, it will accept the parameters in order.

Chapter 4. Using Basic Statistics Tools

In this chapter you will learn:

- Using the **Summary** tool from the **Statistics** menu
- Using the **Protocol Hierarchy** tool from the **Statistics** menu
- Using the **Conversations** tool from the **Statistics** menu
- Using the **Endpoints** tool from the **Statistics** menu
- Using the **HTTP** tool from the **Statistics** menu
- Configuring **Flow Graph** for viewing TCP flows
- Creating IP-based statistics

Introduction

One of Wireshark's strengths is the statistical tools. While using Wireshark, we have various types of tools starting from simple tools for listing end nodes and conversations to the more sophisticated tools such as Flow and IO graphs.

In the next two chapters we will learn how to use these tools. In this chapter we will look at the simple tools that provide us with basic network statistics; that is, who talks to whom over the network, which are the "chatty" devices, what packet sizes run over the network, while in the next chapter we'll get into tools such as IO and Stream graphs, which provide us with much more information about the behavior of the network.

There are some tools that we will not talk about; those that are quite obvious (for example, **Packet sizes**), and those that are less common (such as **ANSP**, **BACnet**, and others).

To use the **Statistics** tool, start Wireshark and choose **Statistics** from the main menu.

Using the Summary tool from the Statistics menu

In this recipe we will learn how to get general information about the data that runs over the network.

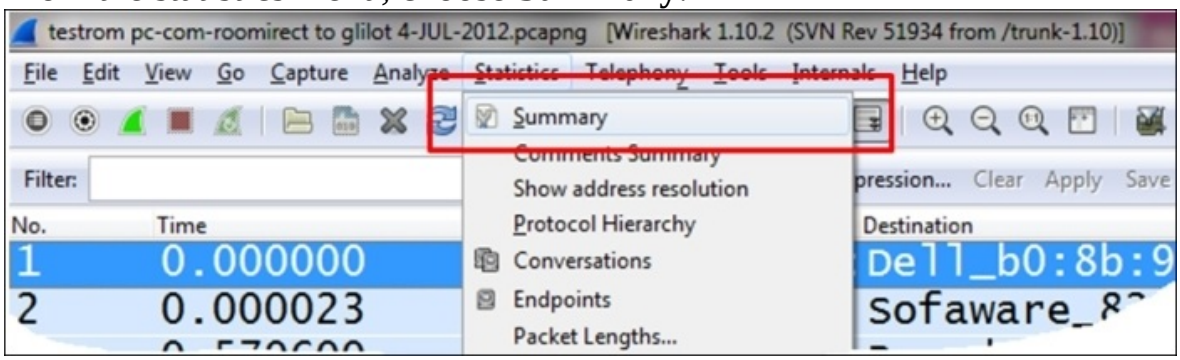
Getting ready

Start Wireshark, click on **Statistics**.

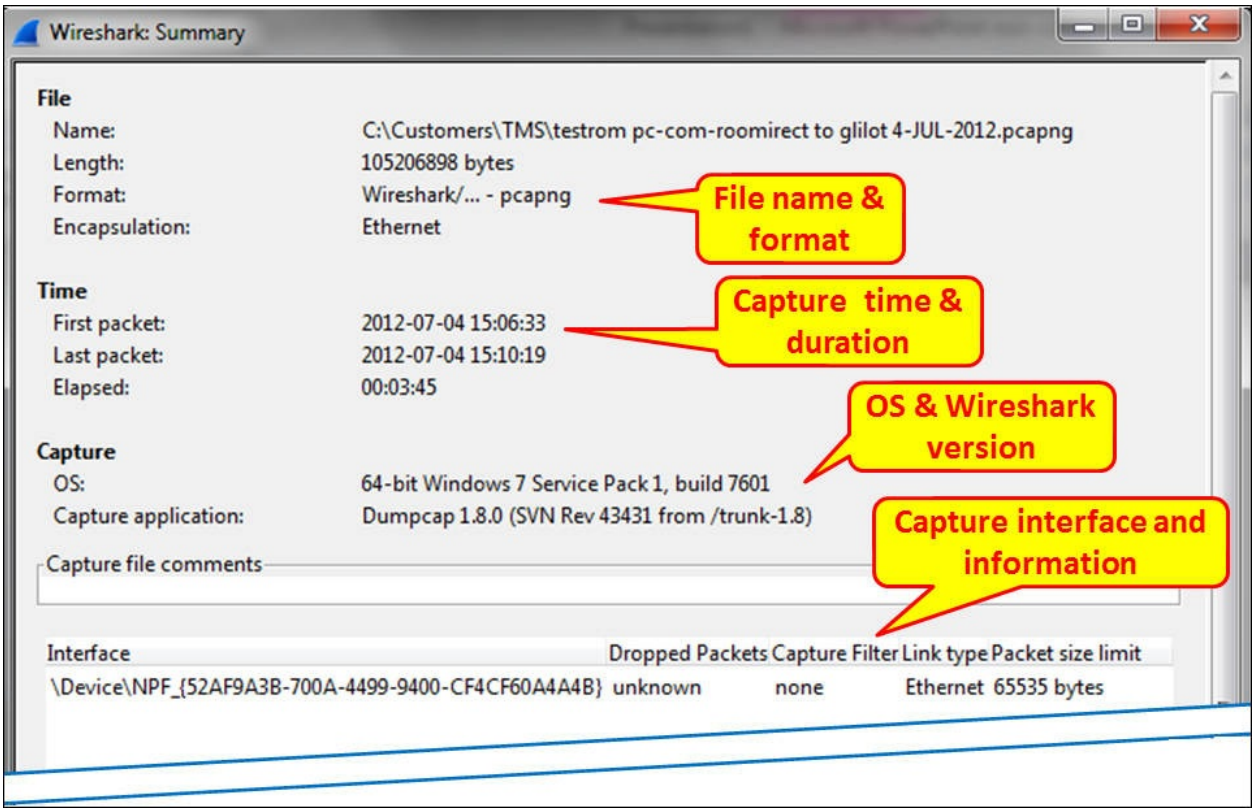
How to do it...

To use the **Summary** tool from the **Statistics** menu, follow the ensuing steps:

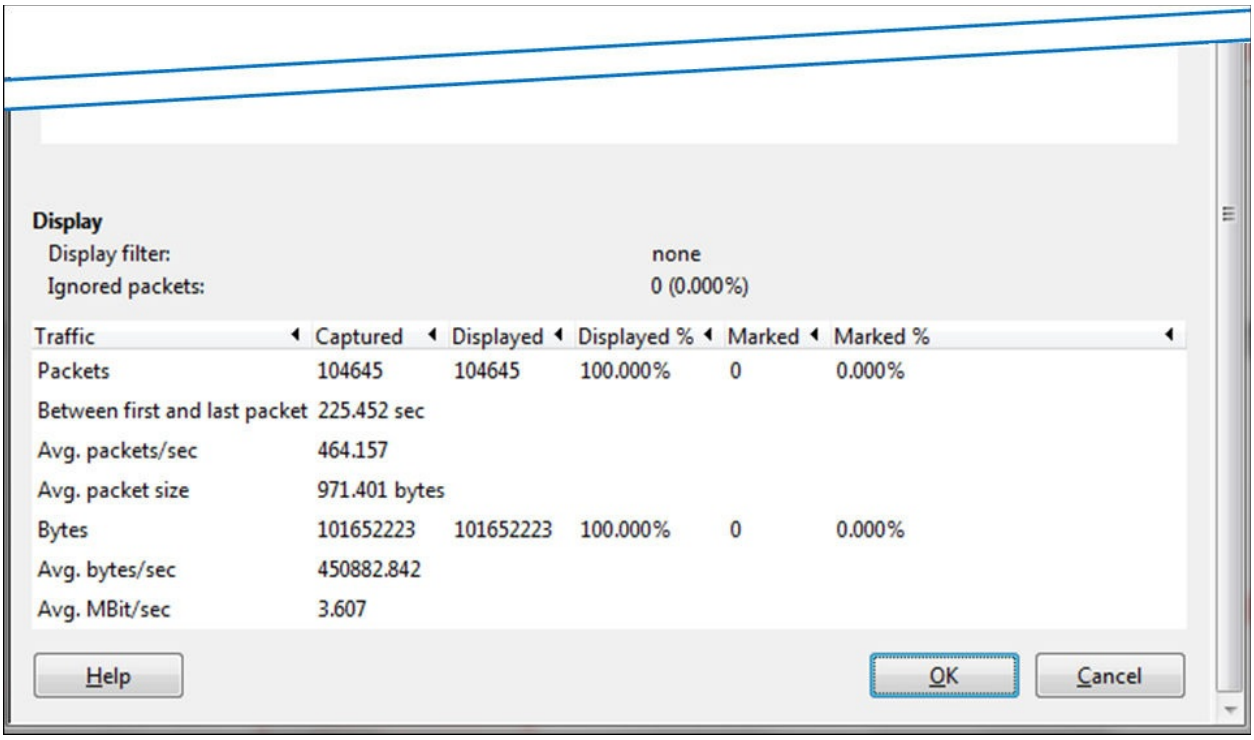
1. From the statistics menu, choose **Summary**.



   What you will get is the **Summary** window (displayed in the following two screenshots).
2. As shown in the following screenshot, in the upper side of the window, you will see:
   - **File**: This part of the window provides file data, such as file name and path, length, and so on
   - **Time**: This part on the window displays the start time, end time, and duration of capture
   - **Capture**: This part of the window shows on which interface the file was captured and also displays a remark window



- In the lower part of the window is the **Display** window, where you will get a summary of the capture file statistics; this includes:
  - The number of packets that were captured: their total number and percentage
  - The number of packets displayed (after passing the **Display Filter**)
  - The number of packets that are marked


Display
Display filter:
none
Ignored packets:
0 (0.000%)
Traffic
Captured
Displayed
Displayed %
Marked
Marked %
Packets
104645
104645
100.000%
0
0.000%
Between first and last packet
225.452 sec
Avg. packets/sec
464.157
Avg. packet size
971.401 bytes
Bytes
101652223
101652223
100.000%
0
0.000%
Avg. bytes/sec
450882.842
Avg. MBit/sec
3.607
Help
OK
Cancel

How it works...

This menu simply collects all the captured data, and when a filter is defined, it presents the filtered data. When the question is, "how do I use Wireshark simply to know what is the average packets or bytes per second?", this is your answer.

There's more...

From the **Summary** window, you can get the average packets/second and bits/second of the entire captured file and also for the displayed data.

Using the Protocol Hierarchy tool from the Statistics menu

In this recipe, we will learn how to get protocol hierarchy information of the data that runs over the network.

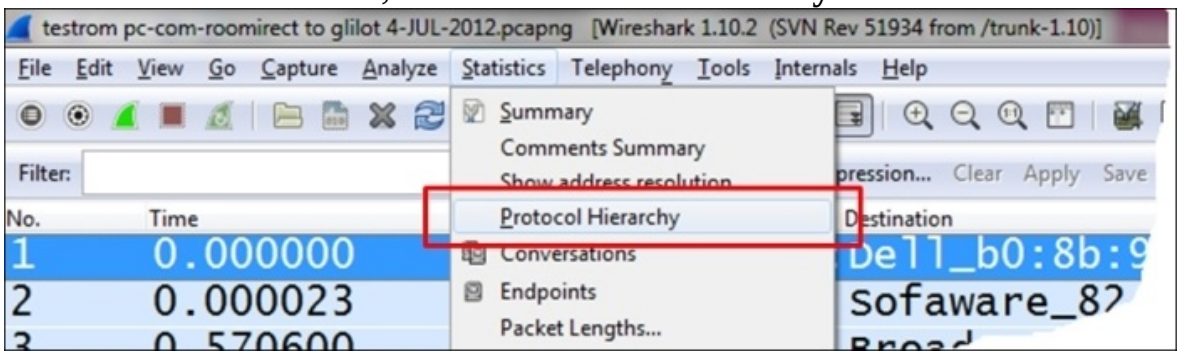
Getting ready

Start Wireshark, click on **Statistics**.

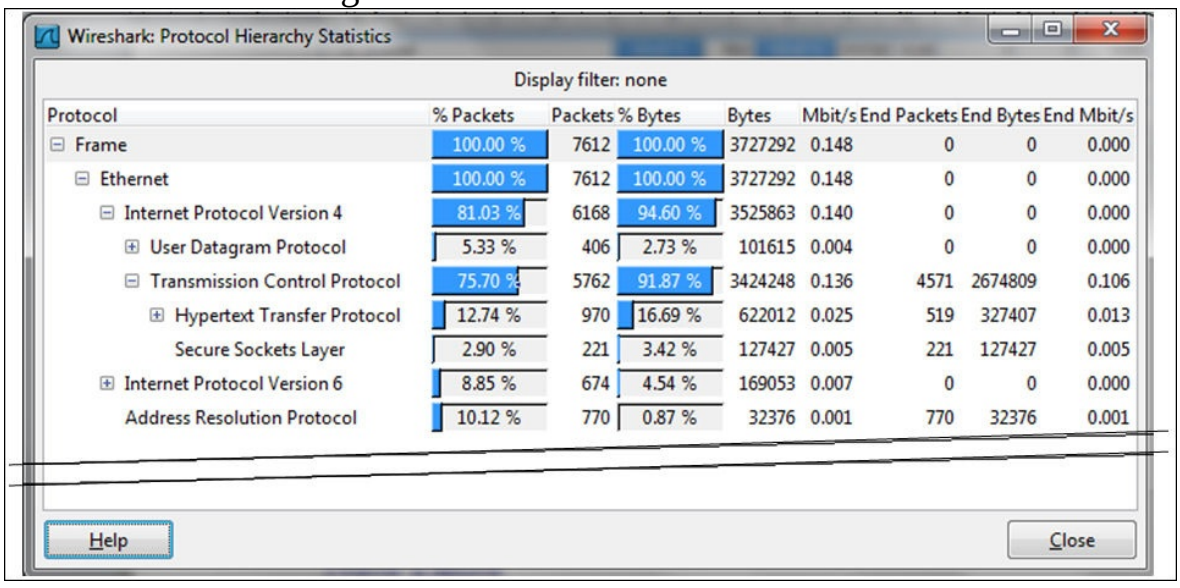
How to do it...

To use the **Protocol Hierarchy** tool from the statistics menu, go through the following steps:

1. From the statistics menu, choose **Protocol Hierarchy**.



2. What you will get here is data about the protocol distribution in the captured file. You will get the protocol distribution of the captured data, as shown in the following screenshot:



- You will get the following fields in the **Protocol Hierarchy** window:
  - **Protocol**: This field specifies the protocol name
  - **% Packets**: This field specifies the percentage of protocol packets from the total captured packets
  - **Packets**: This field specifies the number of protocol packets from the total captured packets

- **% Bytes**: This field specifies the percentage of protocol bytes from the total captured packets
- **Bytes**: This field specifies the number of protocol bytes from the total captured packets
- **Mbit/s**: This field specifies the bandwidth of this protocol in relation to the capture time
- **End Packets**: This field specifies the total number of packets in this protocol (for the highest protocol in the decode file)
- **End Bytes**: This field specifies the absolute number of bytes of this protocol (for the highest protocol in the decode file)
- **End Mbit/s**: This field specifies the bandwidth of this protocol relative to the capture packets and time (for the highest protocol in the decode file)

Tip

The **End Packets**, **End Bytes**, and **End Mbits/s** columns are those in which the protocol in this line is the last protocol in the packet (that is, when the protocol comes at the end of the packet, and there is no higher layer information). These can be, for example, TCP packets with no payload (for example, SYN packets), which do not carry any upper layer information. That is why you see a **0** count for Ethernet and IPv4 and UDP end packets because there are no frames where these protocols are the last protocol in the frame.

How it works...

In simple terms, it calculates statistics over the captured data. Some important things to notice are:

- The percentage always refers to the same layer protocols. For example, we see in the previous example that IPv4 has 81.03 percent of the packets, IPv6 has 8.85 percent of the packets, and ARP has 10.12 percent of the packets; a total of 100 percent of the protocols over layer-2.
- On the other hand, we see that TCP has 75.70 percent of the data, and within TCP, only 12.74 percent of the packets are HTTP, and there is nearly nothing more. This is because Wireshark counts only the packets with the HTTP headers. It doesn't count for example, the acknowledge packets or data packets that doesn't have HTTP header.

There's more...

In order to ensure that Wireshark will also count the data packets, for example, the data packets of HTTP within the TCP packet, disable the **Allow sub-dissector** option to reassemble the TCP streams. You can do this from the **Preferences** menu or by right-clicking on the TCP in the **Packet Details** pane.

Using the Conversations tool from the Statistics menu

In this recipe, we will learn how to get information about conversations that runs over the network.

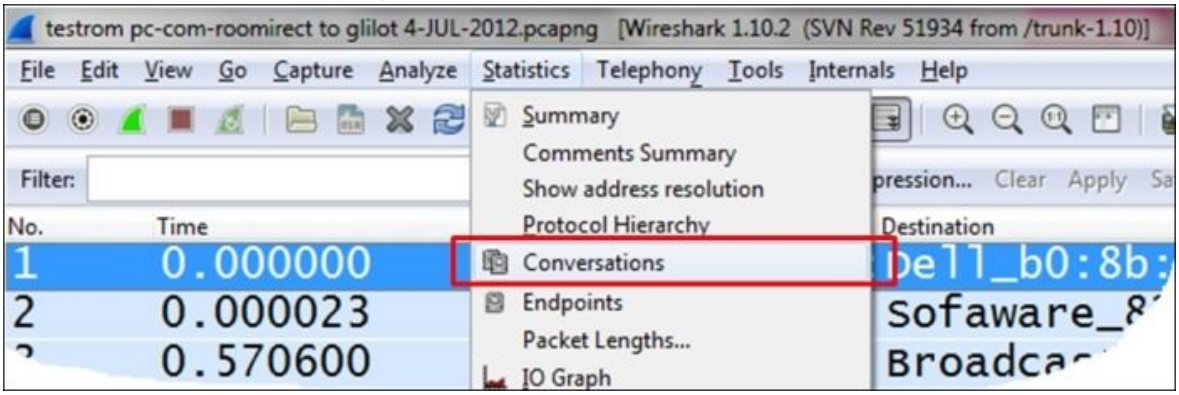
Getting ready

Start Wireshark, click on **Statistics**.

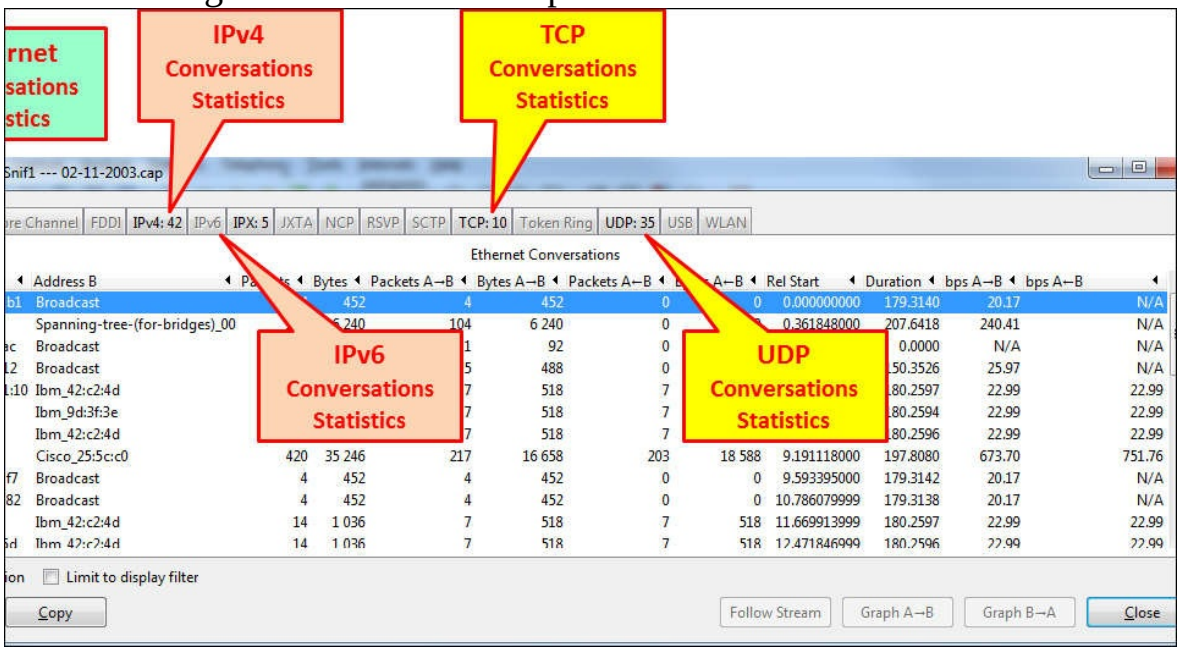
How to do it...

To use the **Conversations** feature from the **Statistics** menu, follow the ensuing steps:

1. From the statistics menu, choose **Conversations**.



2. The following window will come up:



- You can choose between layer 2 Ethernet statistics, layer 3 IP statistics, or layer 4 TCP or UDP statistics.
- You can use these statistics tools:

- **On layer 2** (**Ethernet**): To find and isolate broadcast storms or
- **On layer 3 or 4** (**TCP/IP**): To connect in parallel with the Internet router port and check who is loading the line to the ISP

  ### Tip

  If you see that there is a lot of traffic going out to port 80 (HTTP) on a specific IP address on the Internet, you just have to copy the address to your browser and see access to which website is most "popular" with your users.

  If you don't get anything, simply go to a standard DNS resolution website (just Google *DNS lookup*) and find out which is the traffic that loads your Internet line.

- You can also limit the conversations statistics to a display filter by selecting the **Limit to display filter** checkbox located down in the down to the left of the window. In this way, statistics will be presented on all the packets passing the display filter.
- For viewing the IP addresses as names, you can select the **Name resolution** checkbox. For viewing the name resolution, you will have to first enable it by navigating to **View** | **Name Resolution** | **Enable for Network layer**.
- For TCP or UDP, you can mark a specific packet in the **Packet list** and then select **Follow TCP Stream** or **Follow UDP Stream** (depending on whether it is a UDP or TCP packet) from the menu that appears on the screen. This will define a display filter that will show you the specific stream of data.

How it works...

A network conversation is the traffic between two specific endpoints. For example, an IP conversation is all the traffic between two IP addresses and TCP conversations represent all the TCP connections.

There's more...

There are many network problems that will simply "pop up" while using the **Conversations** list.

Ethernet conversations statistics

In the Ethernet conversations statistics, look for the following problems:

- Large amount of broadcasts: you might be viewing a broadcast storm (a minor one. In a major one, you might not see anything.)
  Tip

  What usually happens in a severe broadcast storm is that due to thousands, and even tens of thousands, of packets sent and received per second by Wireshark, the software simply stops showing us the data and the screen freezes. Only when you disconnect Wireshark from the network will you see it.

- If you see a lot of traffic coming from a specific MAC address, look at the first part of the conversation; this is the vendor ID that will give you a hint about the troublemaker.

Tip

Even though the first half of a MAC address identifies the vendor, it does not necessarily identify the PC itself. This is because the MAC address belongs to the Ethernet chip vendor that is installed on the PC or laptop board and is not necessarily from the PC manufacturer. If you are unable to identify the address where the traffic is arriving from, you can ping the suspect and get its MAC address by ARP, find the MAC address in the switches, and if you have a management system, use a simple `find` command to locate it.

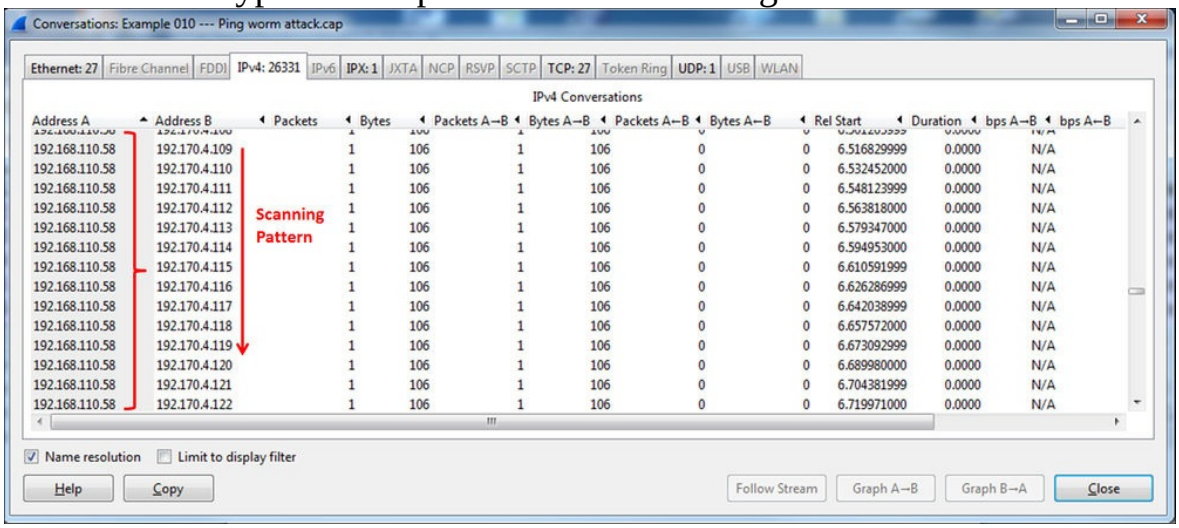
IP conversations statistics

In the IP conversations statistics, look for the following problems:

- Look for IP addresses with a lot of traffic going in or out of them. If it is a server you know (and probably you remember the server's address or address range), then it is OK; but it might also be that someone scanned the

network, or just a PC that generated too much traffic.

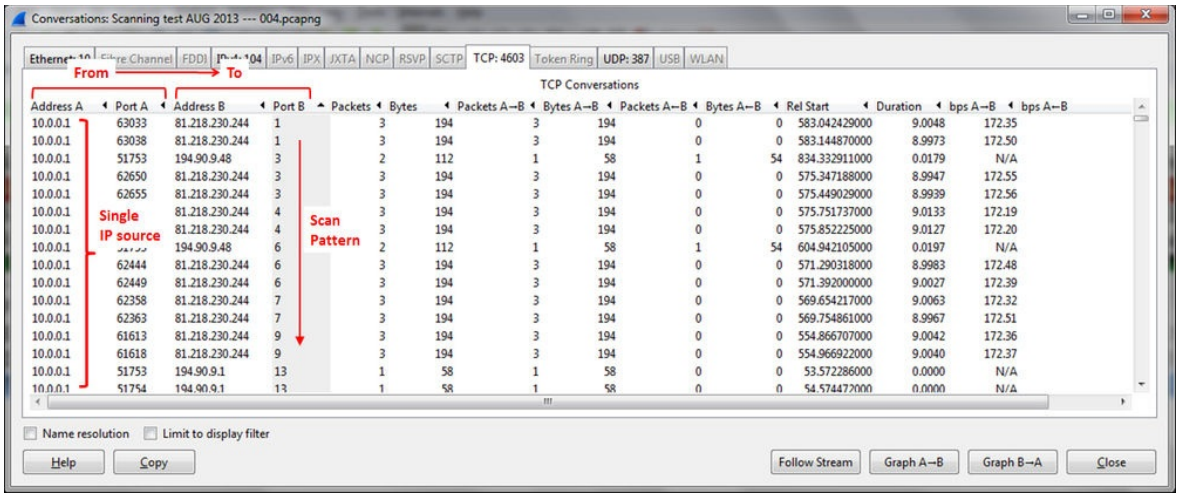
- Look for scanning patterns (presented in detail in Chapter 14, *Understanding Network Security*). It can be a good scan, such as an SNMP software that sends a ping to discover the network, but usually the scans in the network are not good.
- You can see a typical scan pattern in the following screenshot:



In this example, there is a scan pattern. A single IP address, **192.168.110.58**, sends ICMP packets to **192.170.3.44**, **192.170.3.45**, **192.170.3.46**, **192.170.3.47**, and so on (in the screenshot we see only a very small part of the scan). In this case we had a worm that infected all PCs on the network, and the moment it infects a PC, it starts to generate ICMP requests and sends them to the network; such narrow band links (for example, WAN connections) can easily be blocked.

TCP/UDP conversations statistics:

- Look for devices with too many open TCP connections. 10 to 20 connections per PC are reasonable, hundreds are not.
- Look and try to find unrecognized port numbers. It might be OK, but it can mean trouble. In the following screenshot, you can see a typical TCP scan:


Conversations: Scanning test AUG 2013 --- 004.pcapng
From
To
TCP Conversations
Address A | Port A | Address B | Port B | Packets | Bytes | Packets A→B | Bytes A→B | Packets A←B | Bytes A←B | Rel Start | Duration | bps A→B | bps A←B
10.0.0.1 | 63033 | 81.218.230.244 | 1 | 3 | 194 | 3 | 194 | 0 | 0 | 583.042429000 | 9.0048 | 172.35
10.0.0.1 | 63038 | 81.218.230.244 | 1 | 3 | 194 | 3 | 194 | 0 | 0 | 583.144870000 | 8.9973 | 172.50
10.0.0.1 | 51753 | 194.90.9.48 | 3 | 2 | 112 | 1 | 58 | 1 | 54 | 834.332911000 | 0.0179 | N/A
10.0.0.1 | 62650 | 81.218.230.244 | 3 | 3 | 194 | 3 | 194 | 0 | 0 | 575.347188000 | 8.9947 | 172.55
10.0.0.1 | 62655 | 81.218.230.244 | 3 | 3 | 194 | 3 | 194 | 0 | 0 | 575.449029000 | 8.9939 | 172.56
10.0.0.1 | 62657 | 81.218.230.244 | 4 | 3 | 194 | 3 | 194 | 0 | 0 | 575.751737000 | 9.0133 | 172.19
10.0.0.1 | 62658 | 81.218.230.244 | 4 | 3 | 194 | 3 | 194 | 0 | 0 | 575.852225000 | 9.0127 | 172.20
10.0.0.1 | [illegible] | 194.90.9.48 | 6 | 2 | 112 | 1 | 58 | 1 | 54 | 604.942105000 | 0.0197 | N/A
10.0.0.1 | 62444 | 81.218.230.244 | 6 | 3 | 194 | 3 | 194 | 0 | 0 | 571.290318000 | 8.9983 | 172.48
10.0.0.1 | 62449 | 81.218.230.244 | 6 | 3 | 194 | 3 | 194 | 0 | 0 | 571.392000000 | 9.0027 | 172.39
10.0.0.1 | 62358 | 81.218.230.244 | 7 | 3 | 194 | 3 | 194 | 0 | 0 | 569.654217000 | 9.0063 | 172.32
10.0.0.1 | 62363 | 81.218.230.244 | 7 | 3 | 194 | 3 | 194 | 0 | 0 | 569.754861000 | 8.9967 | 172.51
10.0.0.1 | 61613 | 81.218.230.244 | 9 | 3 | 194 | 3 | 194 | 0 | 0 | 554.866707000 | 9.0042 | 172.36
10.0.0.1 | 61618 | 81.218.230.244 | 9 | 3 | 194 | 3 | 194 | 0 | 0 | 554.966922000 | 9.0040 | 172.37
10.0.0.1 | 51753 | 194.90.9.1 | 13 | 1 | 58 | 1 | 58 | 0 | 0 | 53.572286000 | 0.0000 | N/A
10.0.0.1 | 51754 | 194.90.9.1 | 13 | 1 | 58 | 1 | 58 | 0 | 0 | 54.574472000 | 0.0000 | N/A
Single IP source
Scan Pattern
Name resolution
Limit to display filter
Help
Copy
Follow Stream
Graph A→B
Graph B→A
Close

Using the Endpoints tool from the Statistics menu

In this recipe we will learn how to get statistics on endpoints information of the captured data.

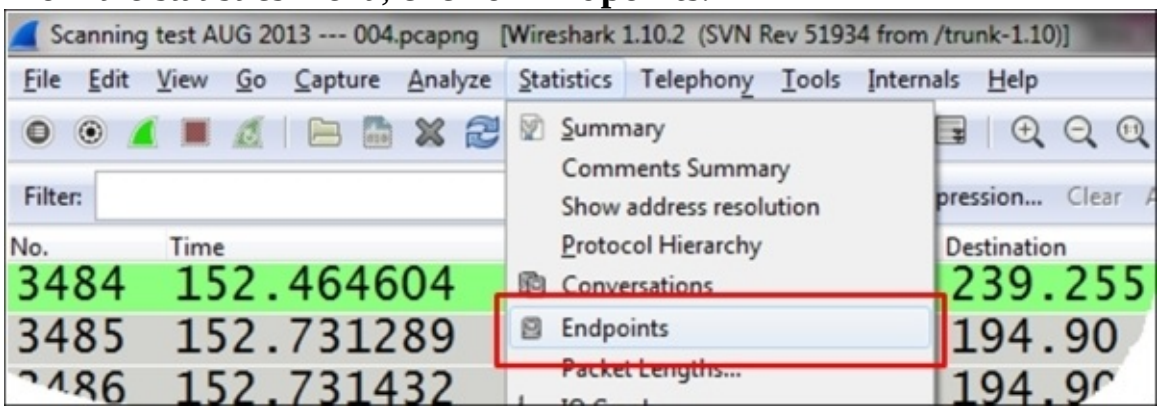
Getting ready

Start Wireshark, click on **Statistics**.

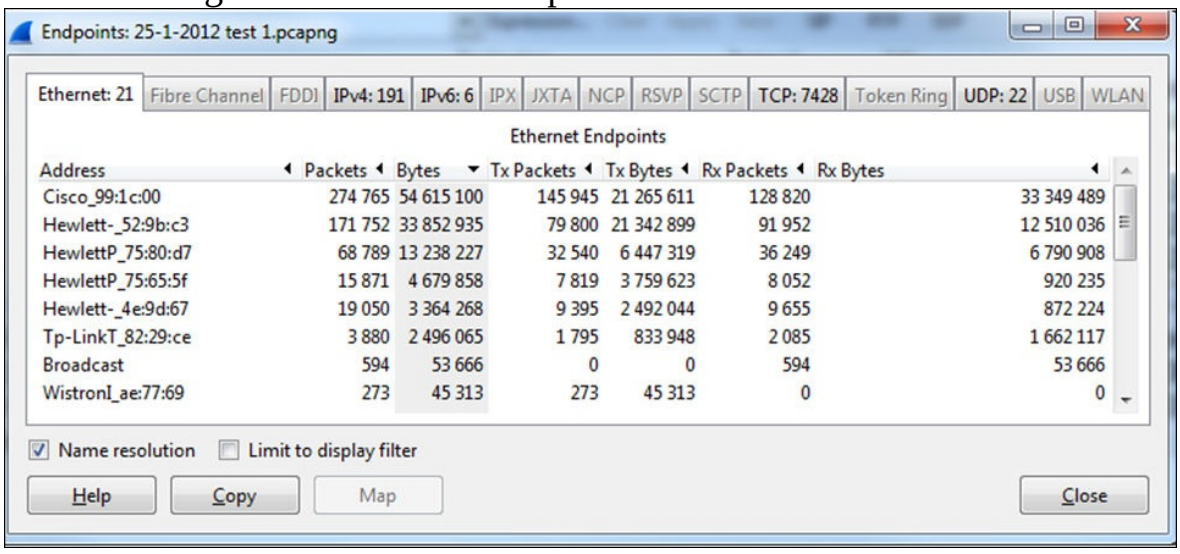
How to do it...

To view the endpoint statistics, follow these steps:

1. From the statistics menu, click on **Endpoints**.



2. The following window will come up:



3. In this window, you will be able to see layers 2 and 3 and 4 endpoints, which are Ethernet, IP, and TCP or UDP.

How it works...

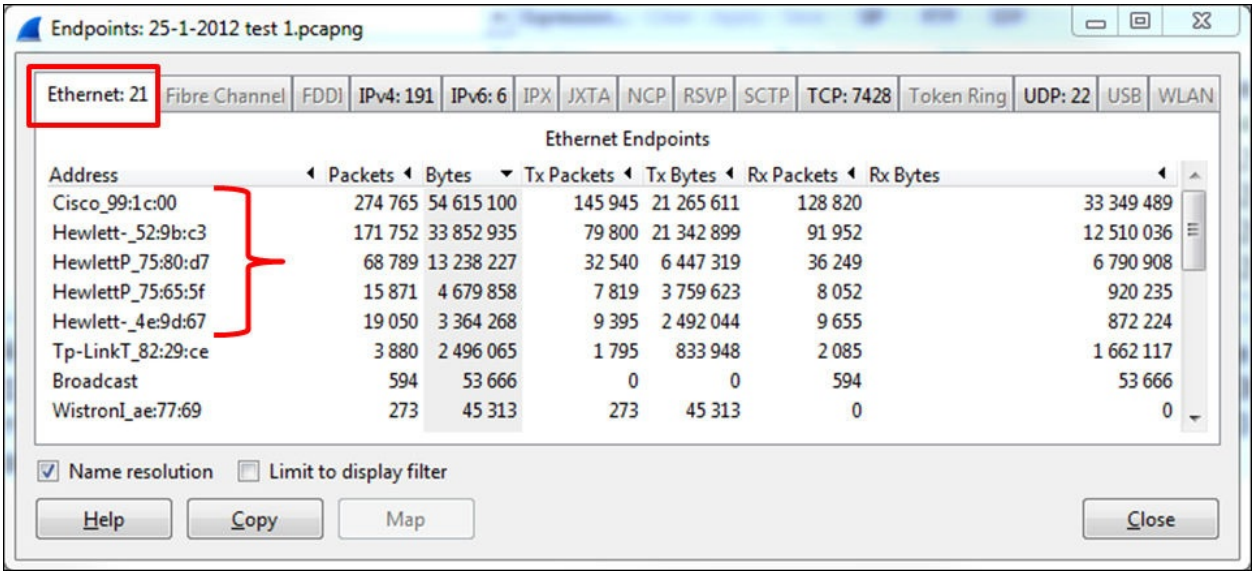
It simply gives statistics on all the endpoints that Wireshark has discovered. It could be any of the situations here:

- Few Ethernet endpoints (these are MAC addresses) with many IP end nodes (these are IP addresses): This will be the case where, for example, we have a router that sends/receives packets from many remote devices, and what we will see is the MAC address of the router and many IP addresses coming/going through it.
- Few IP end nodes with many TCP end nodes: this will be the case for many TCP connections per host. It can be a regular operation of a server with many connections, and it can also be a kind of attack that comes through the network (for example, an SYN attack).

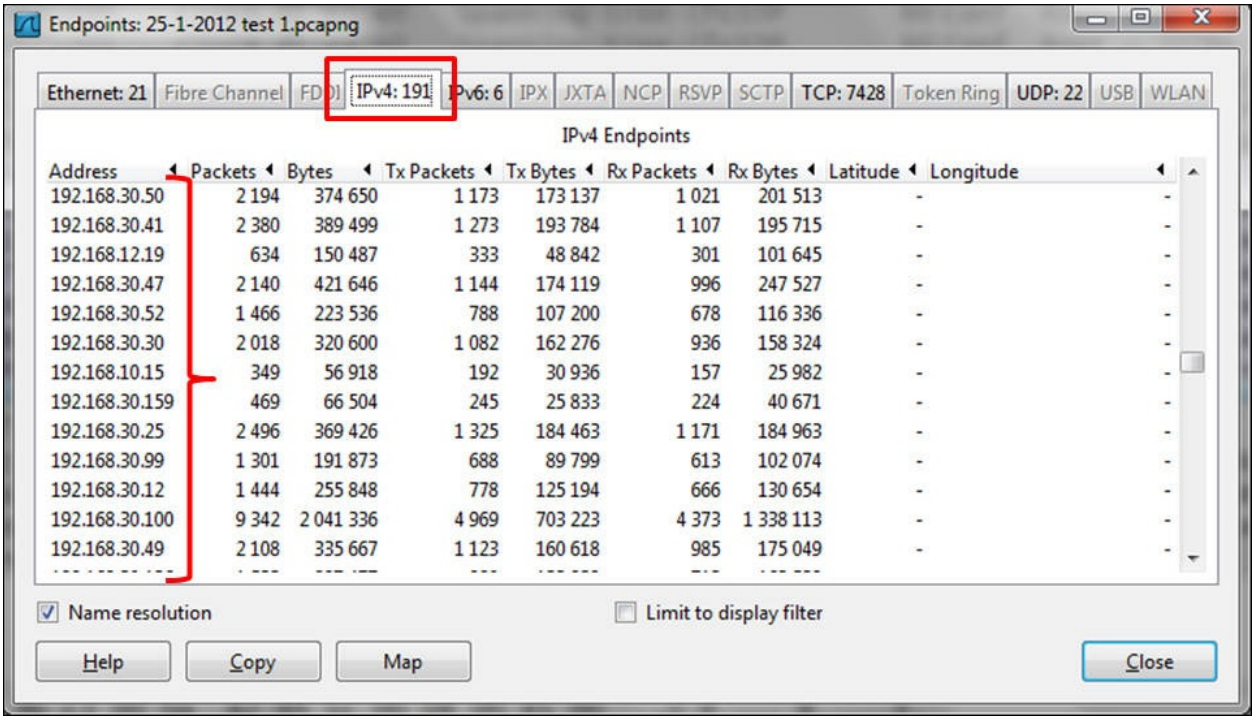
There's more...

Here you see an example for a capture file taken from a network center, and what we can get from it.

In the following screenshot, we see an internal network with four HP servers and a single Cisco router. We can see this from the first part of the MAC address that is resolved to vendor names:



When we choose to see the endpoints under **IPv4: 191**, we see many endpoints coming from the networks **192.168.10.0**, **192.168.30.0**, and also other networks.


Endpoints: 25-1-2012 test 1.pcapng
Ethernet: 21
Fibre Channel
FDDI
IPv4: 191
IPv6: 6
IPX
JXTA
NCP
RSVP
SCTP
TCP: 7428
Token Ring
UDP: 22
USB
WLAN
IPv4 Endpoints
Address Packets Bytes Tx Packets Tx Bytes Rx Packets Rx Bytes Latitude Longitude
192.168.30.50 2 194 374 650 1 173 173 137 1 021 201 513 - -
192.168.30.41 2 380 389 499 1 273 193 784 1 107 195 715 - -
192.168.12.19 634 150 487 333 48 842 301 101 645 - -
192.168.30.47 2 140 421 646 1 144 174 119 996 247 527 - -
192.168.30.52 1 466 223 536 788 107 200 678 116 336 - -
192.168.30.30 2 018 320 600 1 082 162 276 936 158 324 - -
192.168.10.15 349 56 918 192 30 936 157 25 982 - -
192.168.30.159 469 66 504 245 25 833 224 40 671 - -
192.168.30.25 2 496 369 426 1 325 184 463 1 171 184 963 - -
192.168.30.99 1 301 191 873 688 89 799 613 102 074 - -
192.168.30.12 1 444 255 848 778 125 194 666 130 654 - -
192.168.30.100 9 342 2 041 336 4 969 703 223 4 373 1 338 113 - -
192.168.30.49 2 108 335 667 1 123 160 618 985 175 049 - -
Name resolution
Limit to display filter
Help
Copy
Map
Close

Using the HTTP tool from the Statistics menu

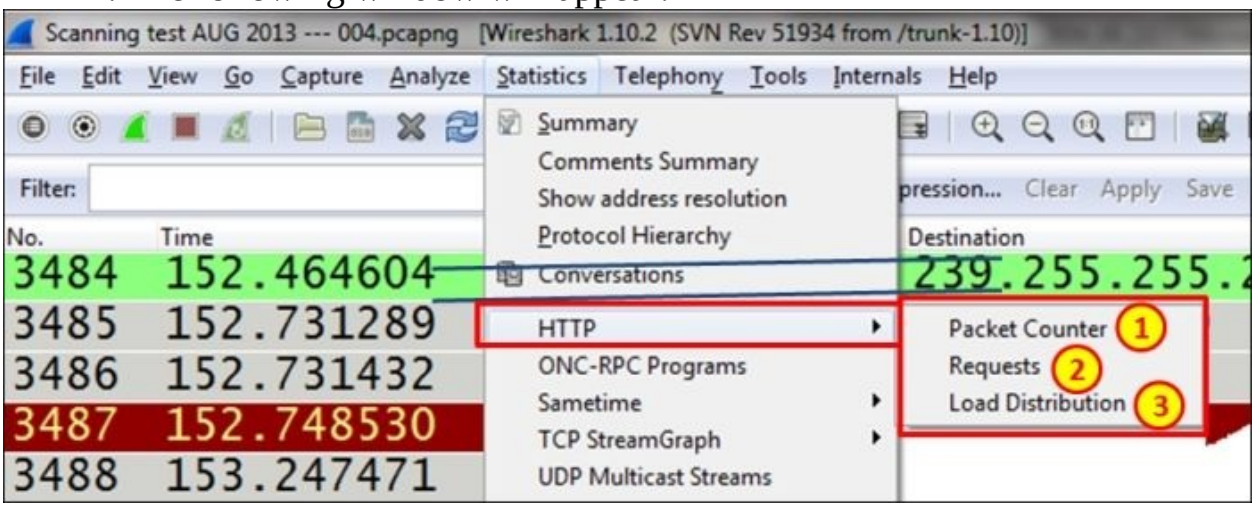
In this recipe we will learn how to use HTTP statistical information of the data that runs over the network.

Getting ready

Start Wireshark, click on **Statistics**.

How to do it...

To view the HTTP statistics follow these steps: From the **Statistics** menu, select **HTTP**. The following window will appear:

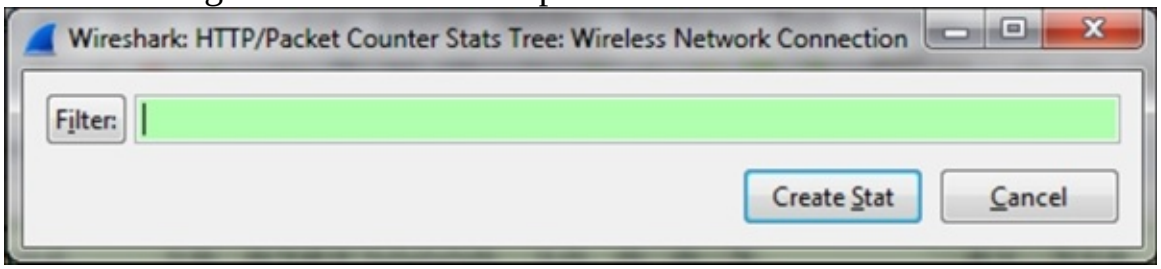


In the **HTTP** submenu, we have the following:

- **Packet Counter (marked as 1 in the preceding screenshot)**: This provides us with the number of packets to each website. This will help us to identify how many requests and responses we have had.
- **Requests (marked as 2 in the preceding screenshot)**: This is used to see request distribution to websites.
- **Load Distribution (marked as 3 in the preceding screenshot)**: This is used to see load distribution between websites.

We will perform the following steps to view the **Packet Counter** statistics:

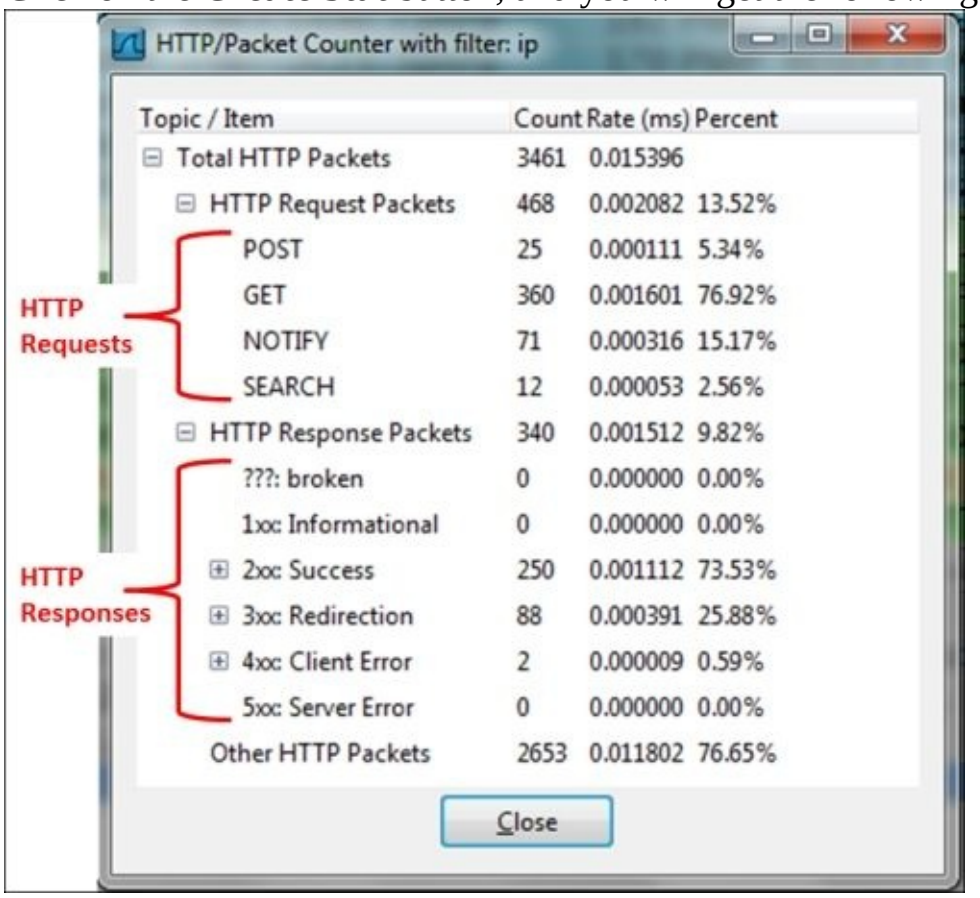
1. Navigate to **Statistics** | **HTTP** | **Packet Counter**.
2. The following filter window will open:



3. In this window, you configure a filter to see the statistics that are applied to

these filters. If you want to see statistics over the whole captured file, leave it blank. This will show you statistics over IP, that is, all the HTTP packets.

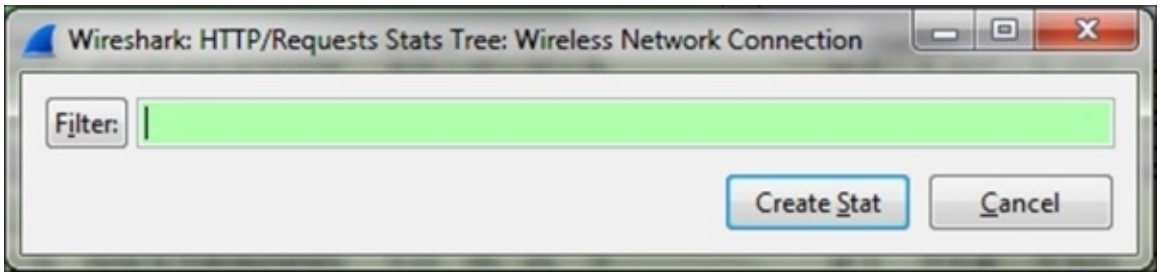
4. Click on the **Create Stat** button, and you will get the following window:



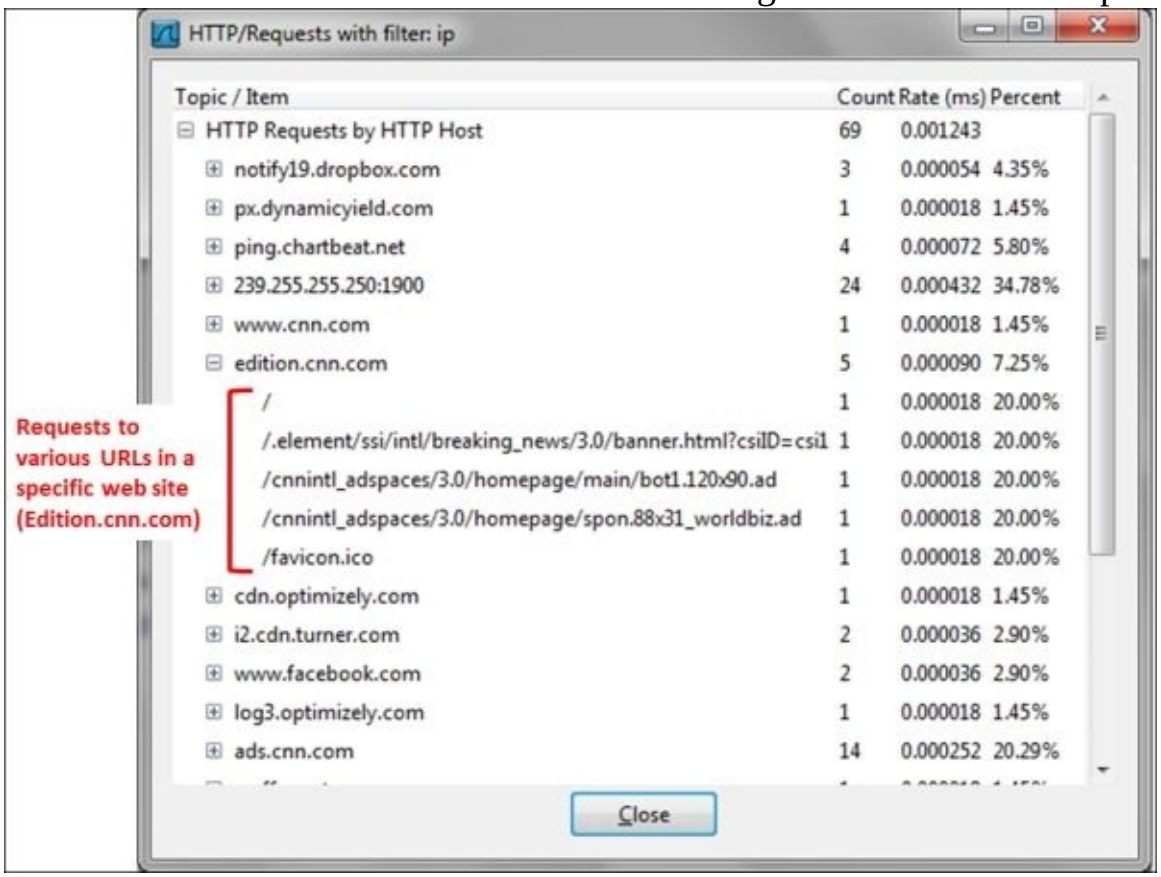
In order to see the HTTP statistics for a specific node, you can configure a filter for it using a display filter format.

We will perform the following steps to view HTTP Requests statistics:

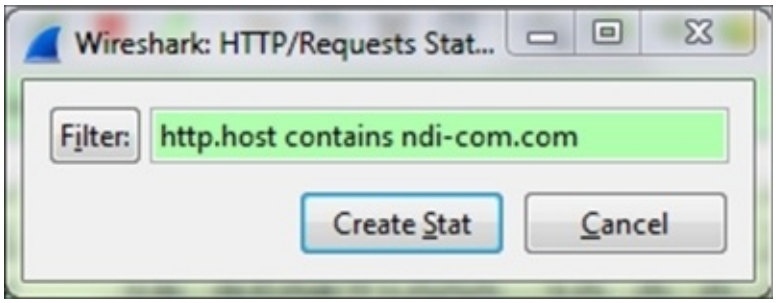
1. Navigate to **Statistics** | **HTTP** | **Requests**. The following window will appear:



2. Choose the filter you need. For all data, leave blank.
3. Click on the **Create Stat** button and the following window will come up:

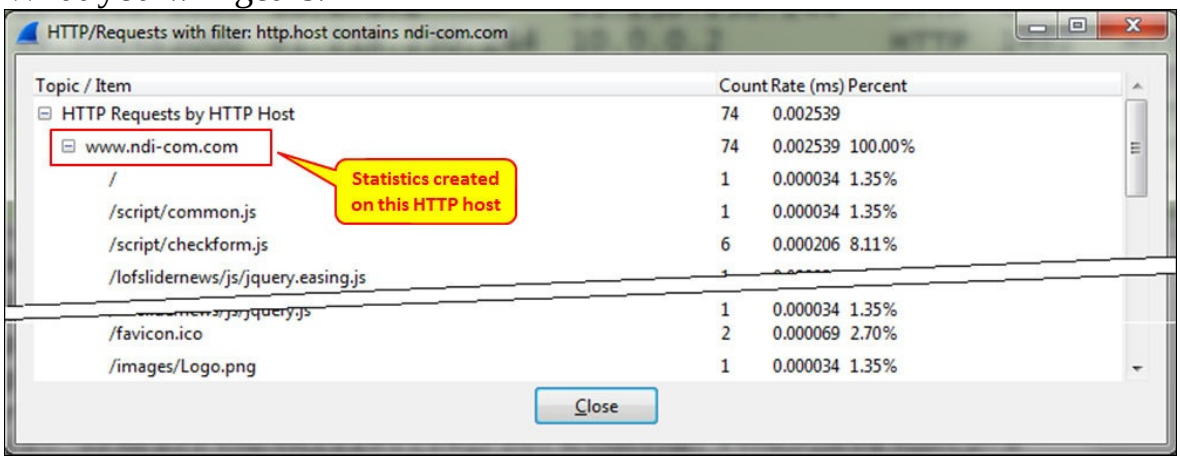


4. To get statistics for a specific HTTP host, you can set a filter `http.host contains <host_name>` or `http.host==<host_name>` (depends on whether you need a hostname with a specific name or a hostname that contains a specific string), and you will see statistics to this specific host.
5. For example, by configuring the filter `http.host contains ndi-com.com`, you will get the statistics for the website www.ndi-com.com (shown in the



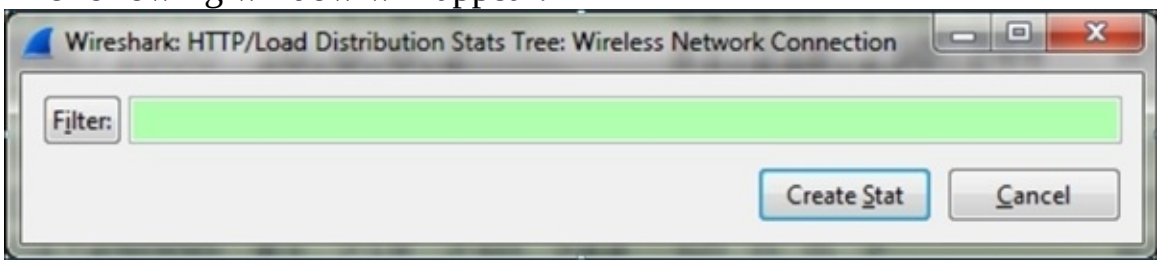
following screenshot):

6. What you will get is:

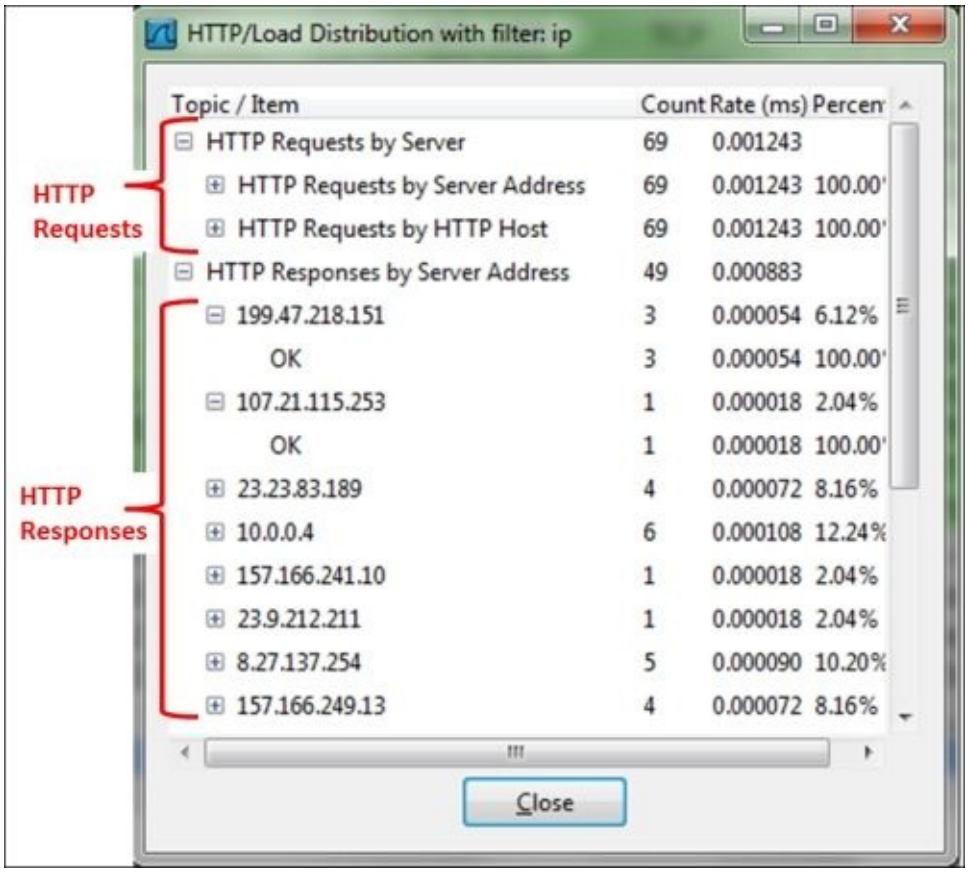


To see **Load Distribution** on the Web or a specific website:

1. Navigate to **Statistics** | **HTTP** | **Packet Counter**.
2. The following window will appear:



3. Choose the filter you need. For all data, leave it blank.
4. Click on the **Create Stat** button and the following window will come up:


HTTP/Load Distribution with filter: ip
Topic / Item
Count
Rate (ms)
Percen
HTTP Requests by Server
69
0.001243
HTTP Requests by Server Address
69
0.001243
100.00
HTTP Requests by HTTP Host
69
0.001243
100.00
HTTP Responses by Server Address
49
0.000883
199.47.218.151
3
0.000054
6.12%
OK
3
0.000054
100.00
107.21.115.253
1
0.000018
2.04%
OK
1
0.000018
100.00
23.23.83.189
4
0.000072
8.16%
10.0.0.4
6
0.000108
12.24%
157.166.241.10
1
0.000018
2.04%
23.9.212.211
1
0.000018
2.04%
8.27.137.254
5
0.000090
10.20%
157.166.249.13
4
0.000072
8.16%
Close
HTTP Requests
HTTP Responses

How it works...

When we open a website, it usually sends requests to several URLs. In this example, one of the websites we opened was www.cnn.com, which took us to edition.cnn.com, where we have sent several requests: to the root URL, to the `breaking_news` URL, and to two other locations on the home page.

There's more...

For deeper HTTP analysis, you can use purpose-specific tools. One of the most common ones is **Fiddler**. You can find it at http://www.fiddler2.com/fiddler2/

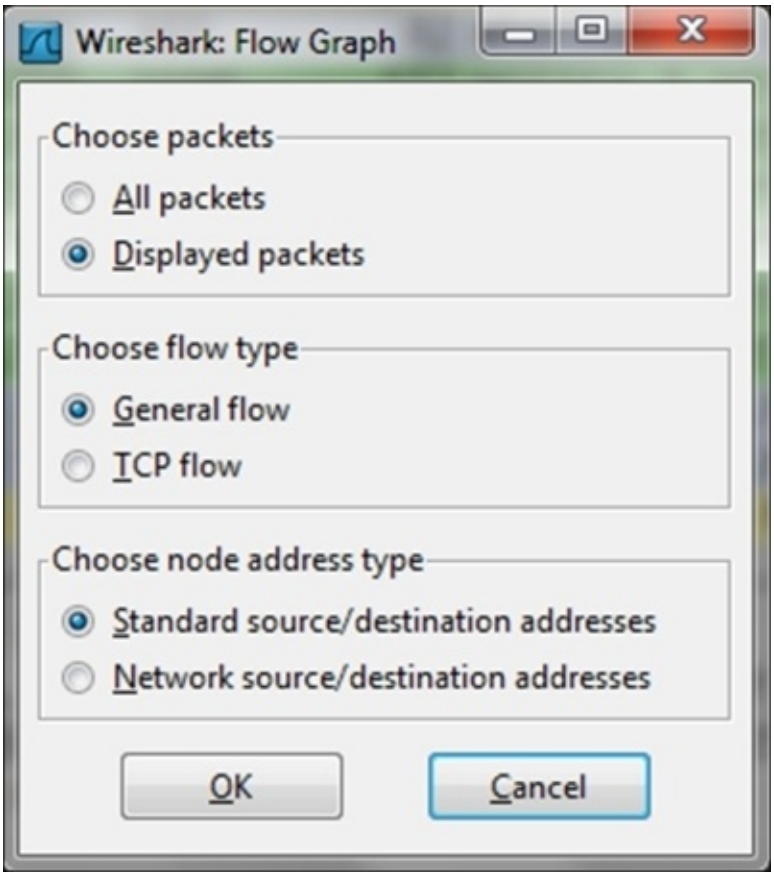
Fiddler is a software tool developed for HTTP troubleshooting and therefore it provides more data with a better user interface for HTTP.

Configuring Flow Graph for viewing TCP flows

In this recipe we will learn how to use the Flow Graph feature.

Getting ready

Open Wireshark and from the **Statistics** menu choose **Flow Graph**. The following window will open:



How to do it...

You can choose several options in the **Flow Graph** window, such as:

- What to view:
  - Choose **All packets**: for viewing all captured packets
  - Choose **Displayed packets**: for viewing only filtered packets
- Flow type:
  - General flow will show all captured or displayed packets (for what you choose before).
  - TCP flow will show only TCP flags, sequence, and ACK numbers. This graph provides a very partial picture of the flow.

How it works...

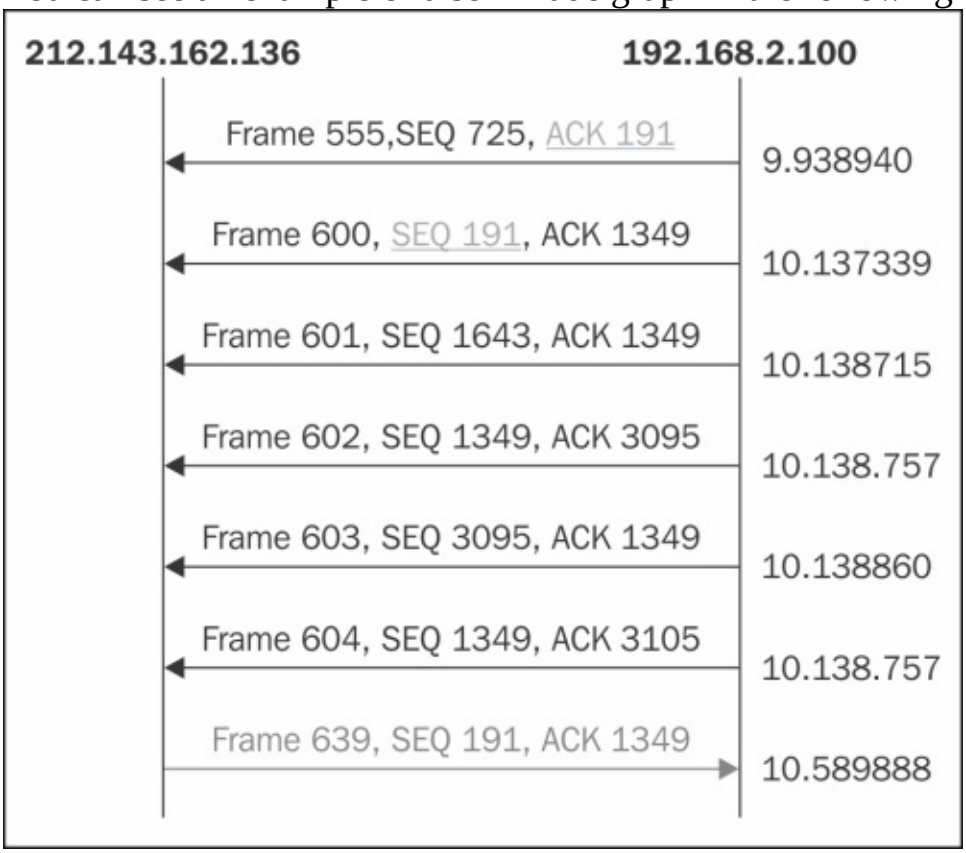
Simply by creating simple statistics from the captured file: nothing special to say here.

There's more...

Understanding TCP problems is sometimes quite complex. The best way to do it most of the time is to use graphical software that have better graphical interface, or simply take a piece of paper along with different colored pens and draw it yourself.

A friendly software that can do the job is the **Cascade Pilot** package by the developers of Wireshark which can be found at http://www.riverbed.com/us/products/cascade/wireshark_enhancements/cascade_

You can see an example of a self-made graph in the following image:



After preparing a few graphs, you will know them like the back of your hand.

Creating IP-based statistics

In this recipe we will learn how to create some IP-based statistics. We will discuss the following statistics tools:

- IP Addresses
- IP Destinations
- IP Protocols Types

Getting ready

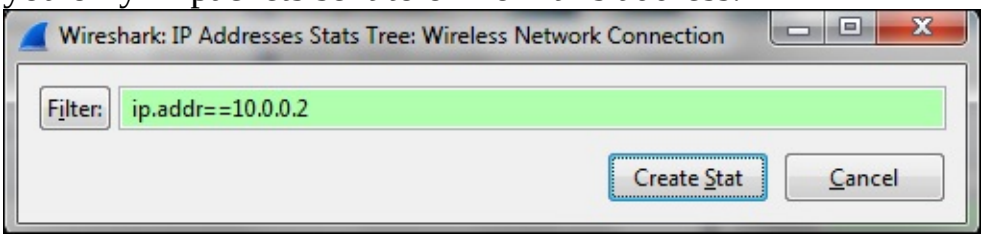
Open Wireshark and click on the **Statistics** menu.

How to do it...

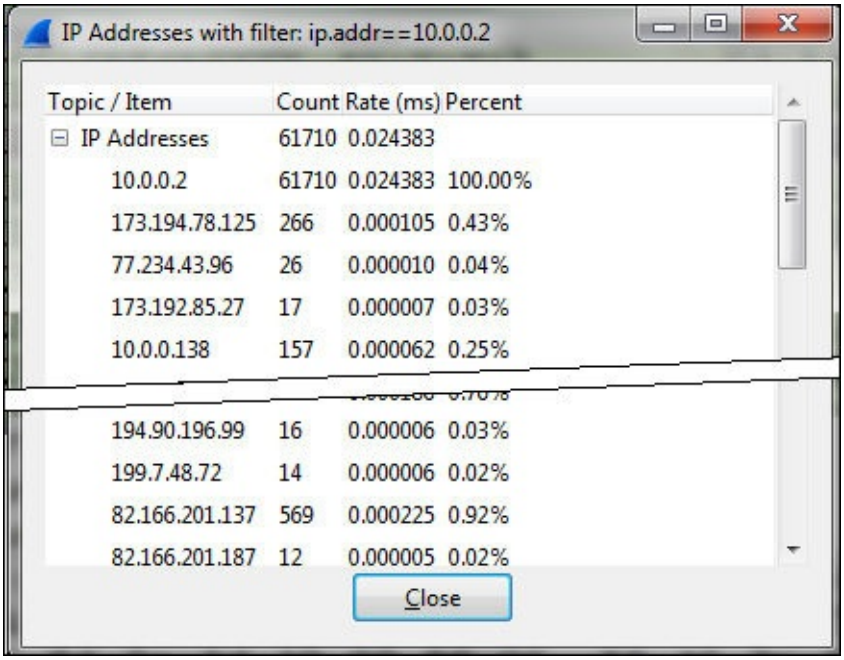
To get IP addresses statistics, perform the following steps:

1. Navigate to **Statistics** | **IP Addresses**.
2. In the window that comes up, select the filter you want to use by clicking on the **Filter** button:

- If you want to see statistics of the whole captured file, leave it blank and all the IP packet statistics will be shown.
- If you want to see only statistics up to a specific IP address, type the filter in the display filter syntax. For example, the filter `ip.addr==10.0.0.2` will show you only IP packets sent to or from this address.

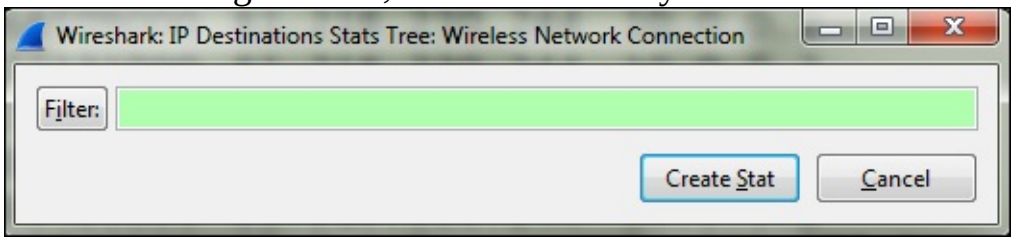


- After typing in the filter, you will get the following statistics:

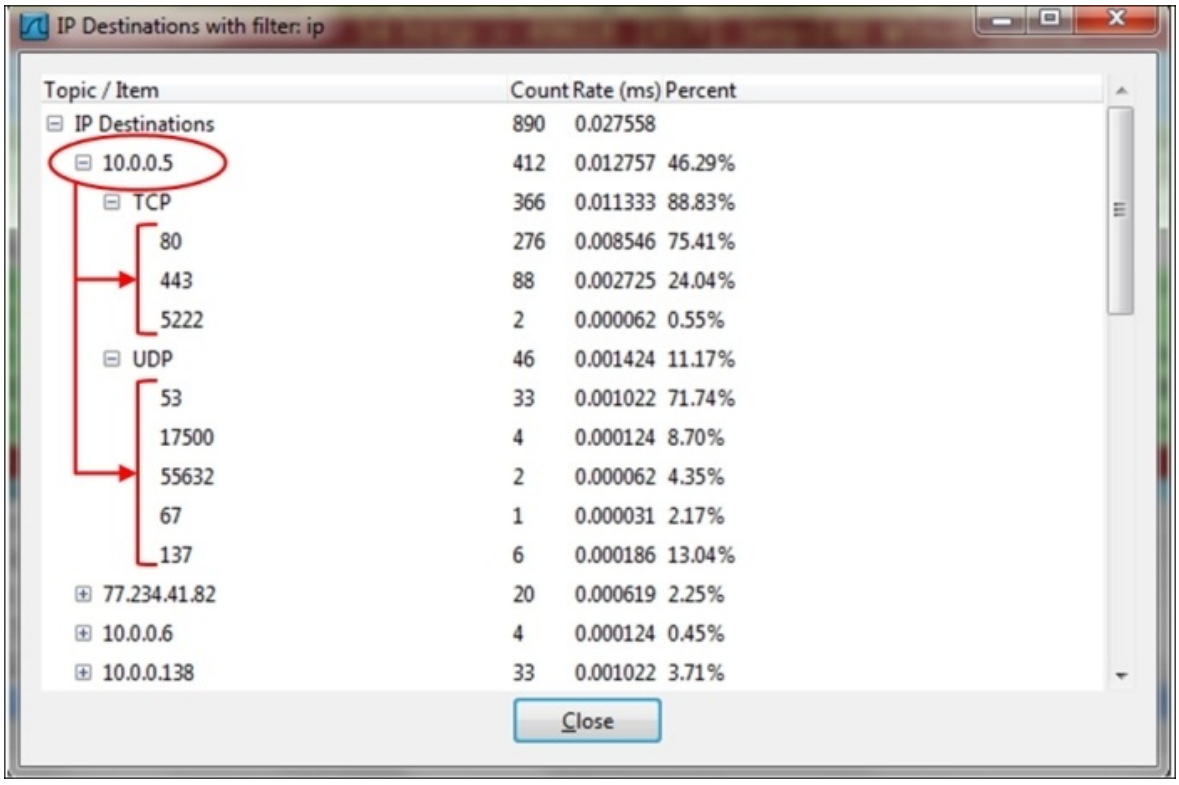


To get IP and TCP/UDP destination statistics, perform the following steps:

1. Navigate to **Statistics** | **IP Destinations**.
2. In the following window, choose the filter you want to use:



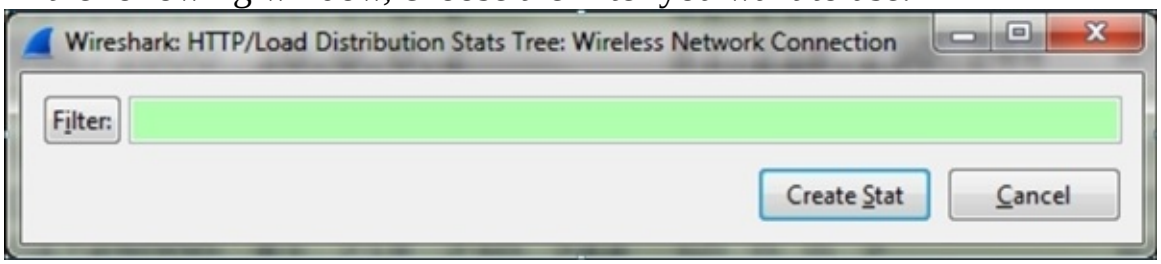
3. This window will show you all those IP addresses to whose destination IPs it has sent packets, and on what protocols.
4. You will get the following statistics:



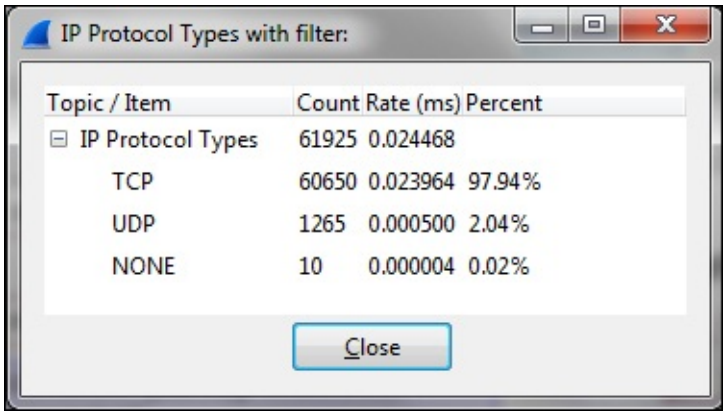
5. In this statistics table, you can see that host **10.0.0.5** has sent TCP packets to port **80**, **443**, and **5222**, and UDP packets to ports **53** and some others.

This is one of the tools that brings up suspected issues; for example, when you see a suspected port with too many packets sent to it, start looking for a reason. To get IP protocol types:

1. Navigate to **Statistics | IP Protocol Types**.
2. In the following window, choose the filter you want to use:



3. You will get the statistics of the protocols that run over IP that are mostly TCP and UDP.


IP Protocol Types with filter:
Topic / Item
Count
Rate (ms)
Percent
IP Protocol Types
61925
0.024468
TCP
60650
0.023964
97.94%
UDP
1265
0.000500
2.04%
NONE
10
0.000004
0.02%
Close

How it works...

Simply by creating statistics over the captured file.

There's more...

There are various options in Wireshark that give you quite similar statistics; these are **Conversations**, **Protocol Hierarchy**, and **Endpoint**, which were discussed at the beginning of this chapter. You can use them in conjunction with the methods we learned in this recipe.

Chapter 5. Using Advanced Statistics Tools

In this chapter we will learn the following:

- Configuring IO Graphs with filters for measuring network performance issues
- Throughput measurements with IO Graph
- Advanced IO Graph configurations with advanced Y-Axis parameters
- Getting information through TCP stream graphs – the Time-Sequence (Stevens) window
- Getting information through TCP stream graphs – the Time-Sequence (tcp-trace) window
- Getting information through TCP stream graphs – the Throughput Graph window
- Getting information through TCP stream graphs – the Round Trip Time window
- Getting information through TCP stream graphs – the Window Scaling Graph window

Introduction

In Chapter 4, Using *Basic Statistics Tools*, we discussed the basic statistics tools such as lists of end users, conversations, capture summary, and more. In this chapter we will look at the advanced statistical tools such as the IO Graph, TCP stream graphs, and in brief, the UDP multicast streams as well.

The tools we will talk about here enable us to have a better look at the network. Here we have two major tools:

- The IO Graph tool enables us to view statistical graphs for any predefined filter; for example, the throughput on a single IP address, the load between two or more hosts, the application throughput, the TCP phenomena distribution, and more
- We will have a deeper look at a single TCP connection using the TCP stream graphs, with the ability to isolate TCP problems and their causes

In this chapter we will learn how to use the tools, and in the next chapters we will use them to isolate and solve networking problems.

Configuring IO Graphs with filters for measuring network performance issues

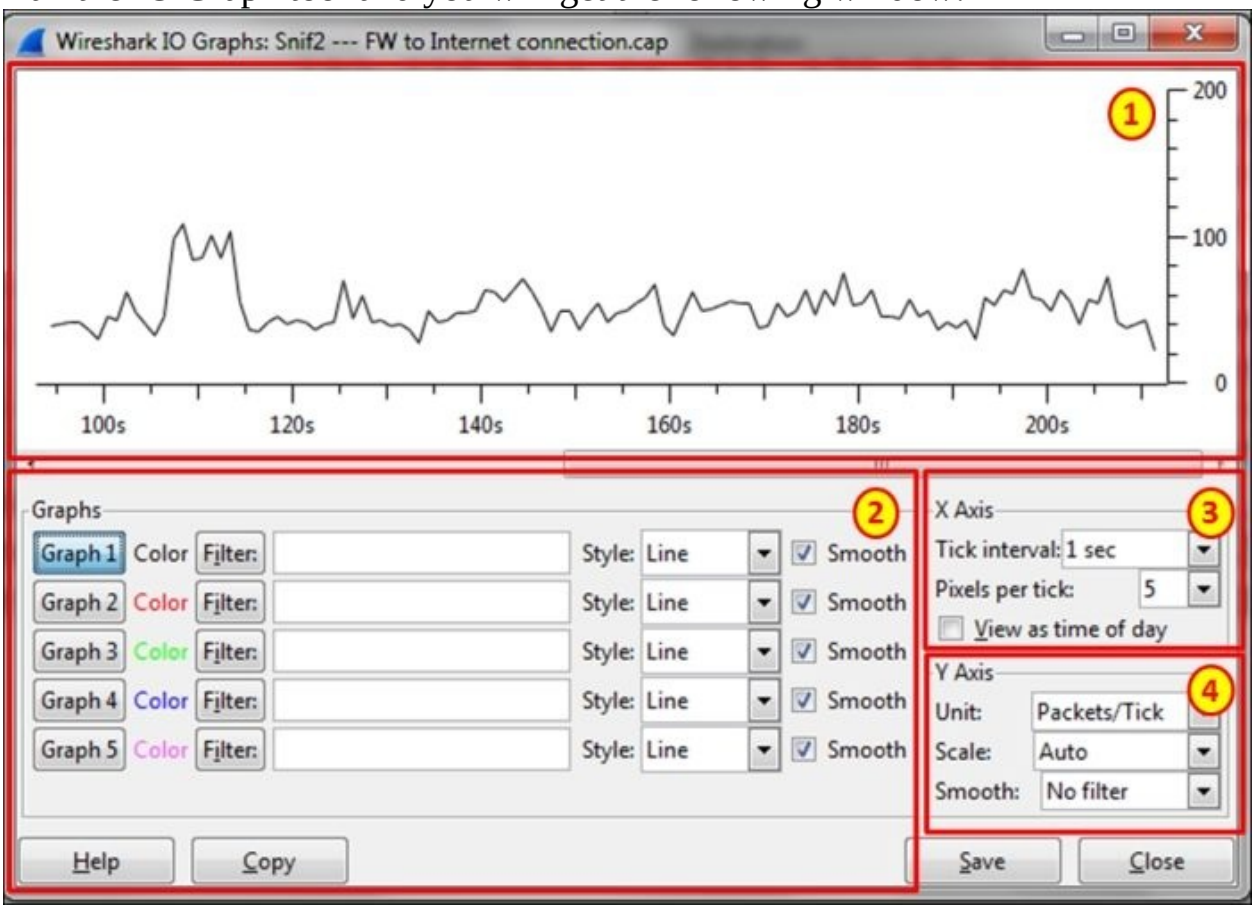
In this recipe we will learn how to use the IO Graph tool and how to configure it for network troubleshooting.

Getting ready

Under the **Statistics** menu, open the IO Graph tool by clicking on **IO Graph**. You can do this during an online file capture or on a file you've captured before. While using the IO Graph tool on a live capture, you will get live statistics on the captured data.

How to do it...

Run the IO Graph tool and you will get the following window:



On the upper part of the window, you will get the graph highlighted as area **1**. On the lower-left part, highlighted as area **2**, you will get the filters that enable you to configure display filters, which will enable specific graphs. On the right-hand side of the window, highlighted as areas **3** and **4**, you will get the **X-Axis** and **Y-Axis** configuration. Let's see what we can configure and how to do it.

Filter configuration

1. In the filter window, fill in a filter in the display-filter format. Only the packets that pass this filter will be taken into account for this graph. You have five optional filters to configure here.
2. Choose the type of graph you want to present: **Line**, **Impulse**, **FBar**, or

Dot.

3. Click on the **Graph** button. This is required in order to activate the filter graph. Don't forget it.

X-Axis configuration

1. Choose a value to enter in **Tick interval:**. The scale can be between 0.001 seconds and 10 minutes.

   ### Tip

   If, for example, we get a peak of 1,000 packets/second when the tick interval X Axis is configured with 1-second intervals, it means that in the last second we've got 1,000 packets. When we change the tick interval for X Axis to 0.1-second intervals, the peak will be different because now we see how many packets were captured in the last 0.1 second.

2. Choose the **Pixels per tick:** value to configure the pixels per tick interval.
3. Mark the **View as time of day** button for choosing the time of day format instead of time since the beginning of capture.

Y-Axis configuration

1. Choose the value for **Unit:** from **Packets/Tick**, **Bytes/Tick**, **Bits/Tick**, or **Advanced...** for choosing the Y-Axis scale.
2. Choose **Scale:** for the Y Axis. You can choose it to be **Linear** or change it to **Logarithmic**. You can also leave it as **Automatic** or change it to manual values when required.
3. Choose a value for **Smooth:** if you want to see a running average; that is, in every tick interval you will see the average of the past ticks. You can choose values from 4 to 1,024 to smooth the graph.

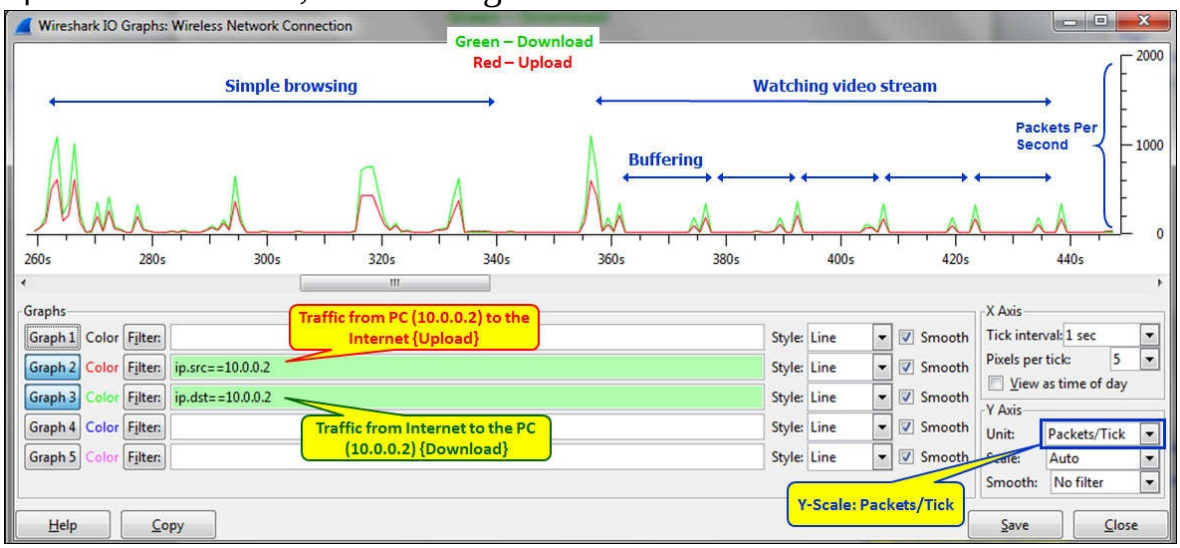
How it works...

The **IO Graphs** feature is one of the important Wireshark tools that enable us to monitor online performance along with offline capture file analysis.

While you are using this tool, it's important to configure the right filter with the right X-Axis and Y-Axis parameters.

Let's have a look at the next two graphs, in which a PC with an IP address of 10.0.0.2 is browsing the Internet. In these two IO graphs, we have configured two filters:

- The first graph is the upload (upstream) traffic graph, which indicates all the traffic from the IP address 10.0.0.2; this is the filter `ip.src==10.0.0.2`, colored in red.
- The second graph is the download (downstream) traffic graph, which indicates all the traffic to the IP address 10.0.0.2; this is the filter `ip.dst==10.0.0.2`, colored in green.

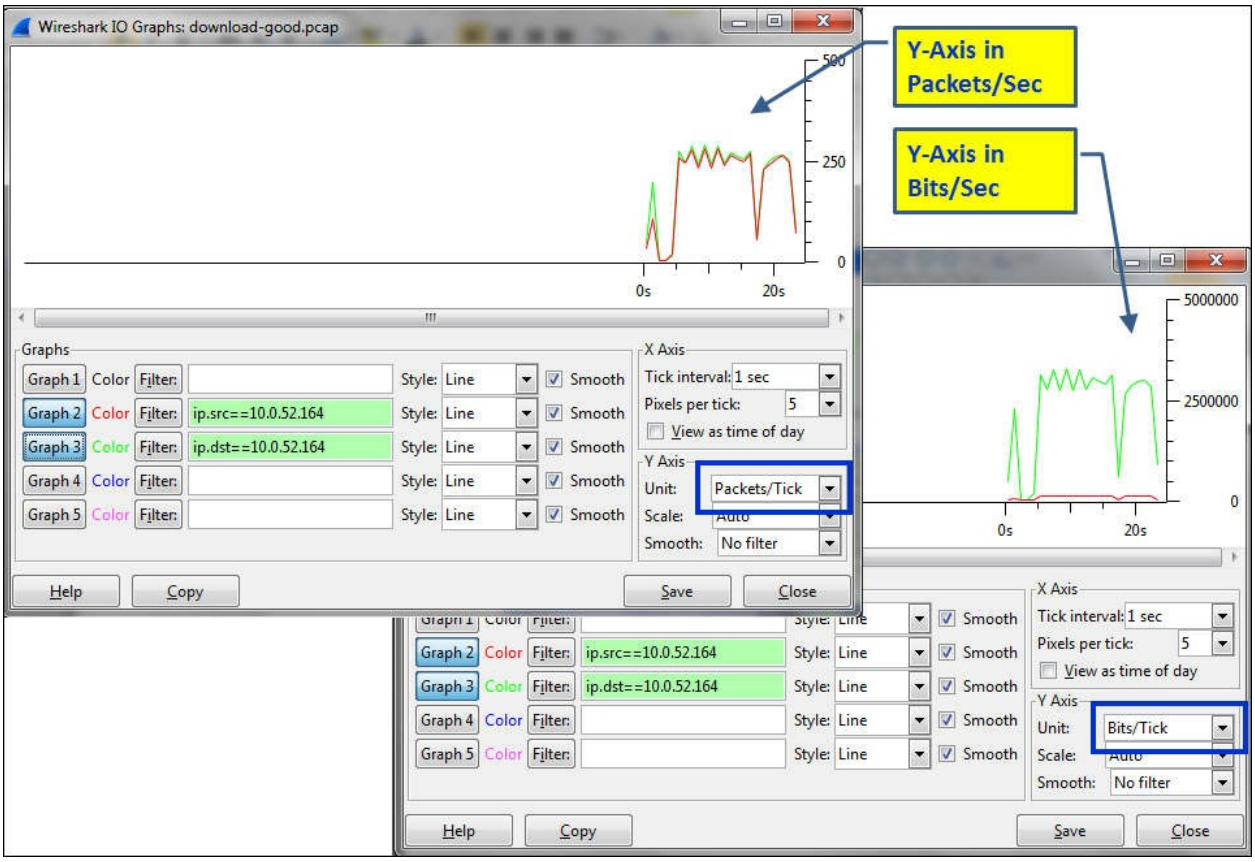


In the first graph, we see that we've measured the traffic when the X Axis is configured to a tick interval of one second and the Y-Axis scale is configured to packets/tick. The result that we've got is that while browsing (on the left-hand side of the graph) or while watching a movie (on the right-hand side of the graph), the upload and download traffic is nearly identical.

In the second graph, we see the traffic in bits/sec. Here, we see the bandwidth required from the network while using it to connect to the Internet; that is, an asymmetrical bandwidth when most of the traffic is in the download direction.

There's more...

Let's have a look at another example here. This is an example of a file download in FTP when 10.0.52.164 downloads a file. Again, you can see that in order to get the traffic on the network, we changed **Unit:** under **Y-Axis** to **Bits/Tick**. **Packets/Tick** is also important and we will see implementations for it in the applications chapters (chapters 7-14) later in the book.



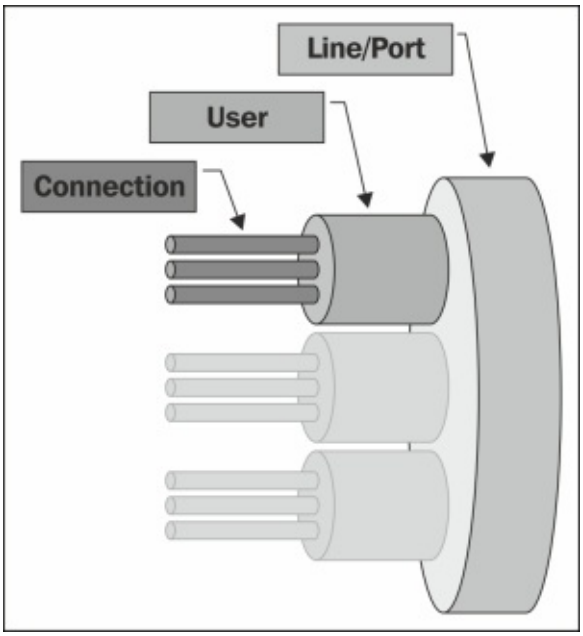
Throughput measurements with IO Graph

IO Graph is a convenient tool for measuring the throughput of a network. Using it, we can measure the traffic and throughput of any predefined filter. In this recipe we will see some examples for measuring the throughput of a network.

Getting ready

Connect your laptop with Wireshark to a network with a port mirror to the link you want to measure, as you learned in Chapter 1, *Introducing Wireshark*. Start a new capture or open an existing file, and open the **IO Graphs** tool from the **Statistics** menu.

While measuring the throughput, we can measure the throughput on a communication line between end devices (PC to server, phone to phone, PC to the Internet, and so on) or to a specific application.



The process of isolating network problems starts from measuring traffic over a link between end devices on single connections and seeing where it comes from.

Some typical measurements are host-to-host traffic, all the traffic to a specific server, all the traffic to a specific application on a specific server, all the TCP performance phenomena on a specific server, and more.

How to do it...

In this recipe, we will provide some basic filters for measuring traffic in the network.

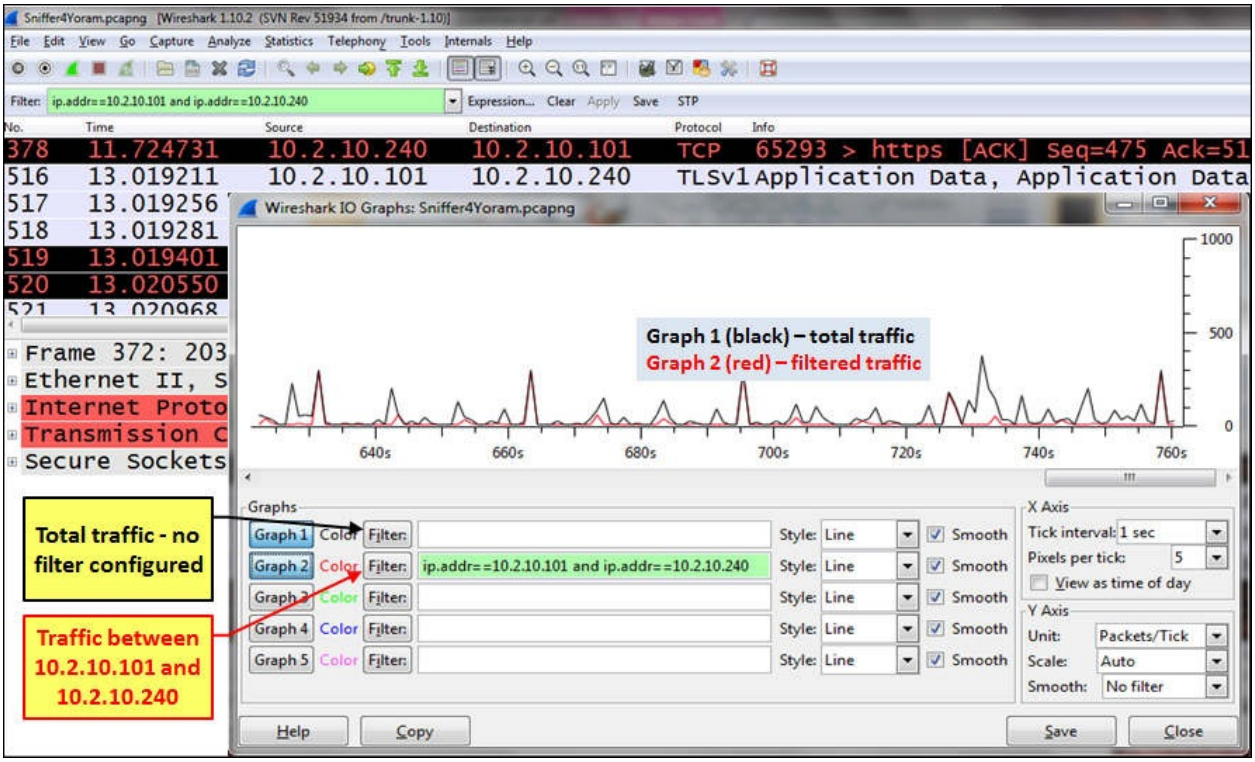
Measuring throughput between end devices

To measure the throughput between end devices, simply configure a display filter between their IP addresses.

For example, to see the traffic between 10.2.10.101 and 10.2.10.240, configure the filter: `ip.add req 10.2.10.240 and ip.add req 10.2.10.240`.

You can either type the filter in the IO Graph's **Filter:** box or perform the following steps:

1. Place the cursor on a packet in a specific connection.
2. Right-click on it and navigate to **Conversation filter** | **IP**. The filter string will appear in the upper display filter box.
3. Copy the filter string from the upper display filter box to one of the IO Graph **Filter:** boxes.
4. Click on the filter bow button in the **IO Graphs** window to activate it.

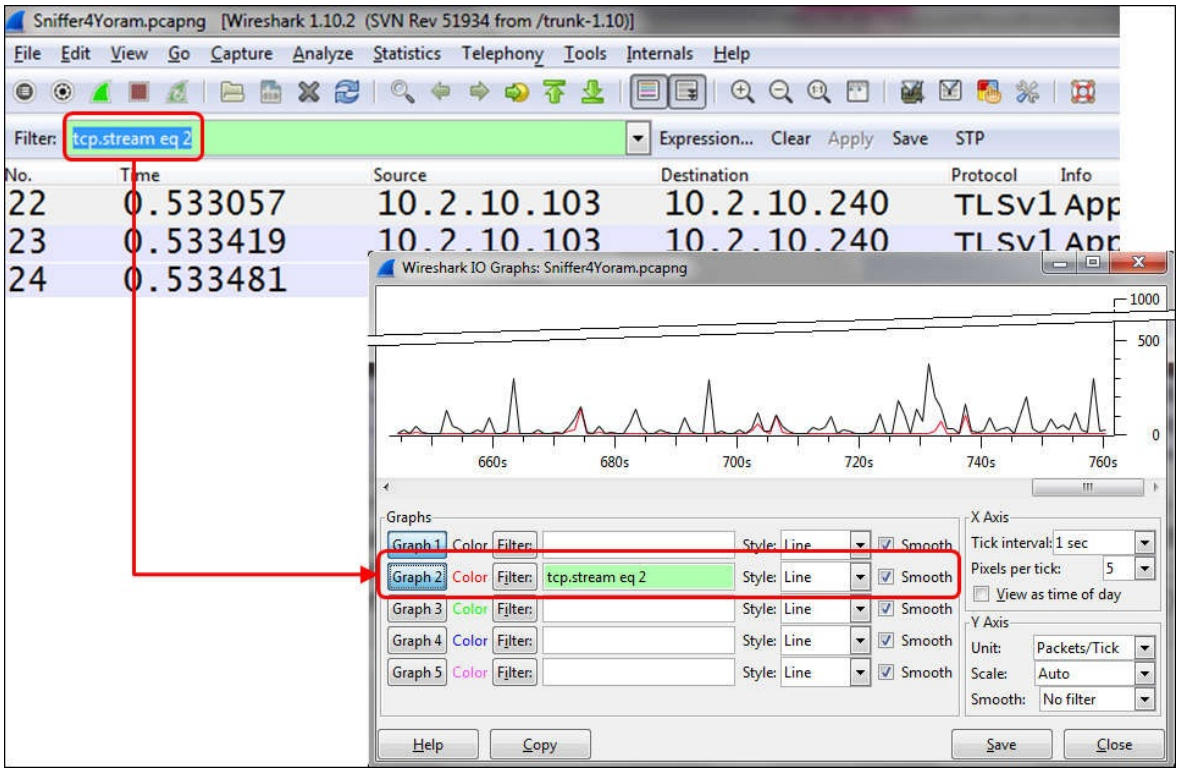


Measuring application throughput

In order to configure the performance measurement of a specific application, you can configure a filter that contains specific port numbers or a specific connection.

There are several ways to isolate an application graph. Here's one of them:

1. In the captured data, click on any packet that belongs to the traffic stream. In TCP it will be a specific connection; in UDP it will be just a stream between two IP/Port pairs.
2. Right-click on it and choose **Follow TCP stream** or **Follow UDP stream**. You will get `tcp.streameq<number>` or `udp.streameq<number>`. `<number>` is simply the number of the stream in the capture file.
3. Copy the string to the filter window in the **IO Graphs** window and you will get the graph of the specific stream.



If you want a graph for specific data on the stream, add information to the filter. For example (in the previous illustration):

- `tcp.streameq 2` and `tcp.analysis.retransmissions` will give all the TCP retransmissions on the specific stream (indicating, for example, a slow network, errors, or packet loss)
- `tcp.streameq 2` and `tcp.analysis.zero_window` will give all the TCP zero window phenomena on the specific stream (indicating a slow end device)

How it works...

The power of the IO Graph tool comes from the fact that you can configure any display filter and see it as a graph in various shapes and configurations. Any parameter in a packet can be filtered and monitored in this way.

There's more...

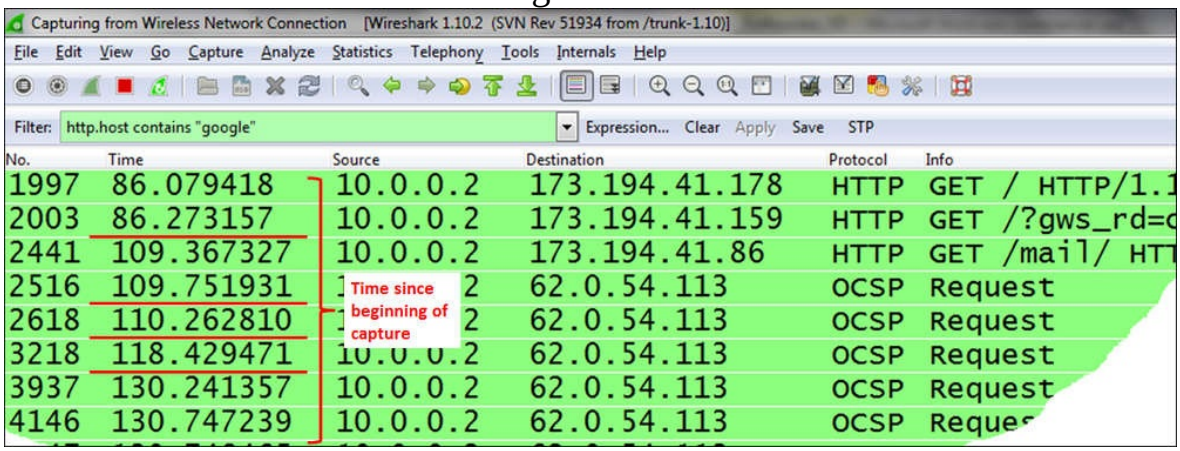
Some examples for parameters that can be monitored are explained in this section.

Graph SMS usage – finding SMS messages sent by a specific subscriber

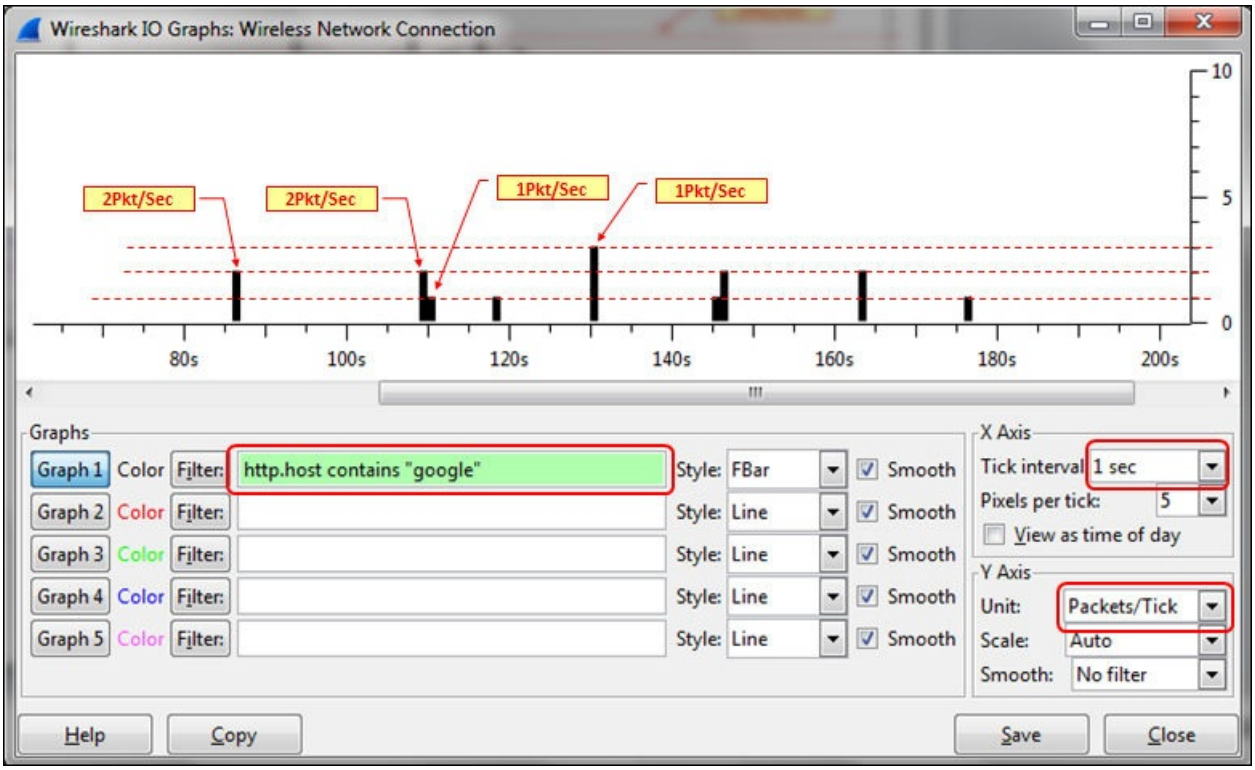
1. To configure the filter, choose **SMPP** (**Short Message Peer to Peer protocol**) packets with the command `Submit_SM`. This is the SMPP command that sends the SMS.
2. Type `smpp.destination_addr == "phone number"` in the filter. The filter `smpp.destination_addr == "972527098241"` was configured in the example.

Graphing number of accesses to the Google web page

1. Open the **IO Graphs** window. You can do it during the capture to view online statistics or open a saved capture file.
2. Configure the filter `http.host contains "<name>"`, in our case, `http.host contains "google"`.
3. In the packet list you will see (while configuring the same filter) the information shown in the following screenshot:



- In the **IO Graphs** window, you will see the following graph:



In the packet capture pane, you can see that we've had two accesses to Google after around 86 seconds, the next two after around 109 seconds, and so on.

Advanced IO Graph configurations with advanced Y-Axis parameters

In standard measurements with the IO Graph tool, we measure the performance of the network in units of packets/second, bytes/second, or bits/second. There are some types of data that cannot be measured with these parameters, and this is the reason we have the **Advanced...** feature in the **Y-Axis** options.

Getting ready

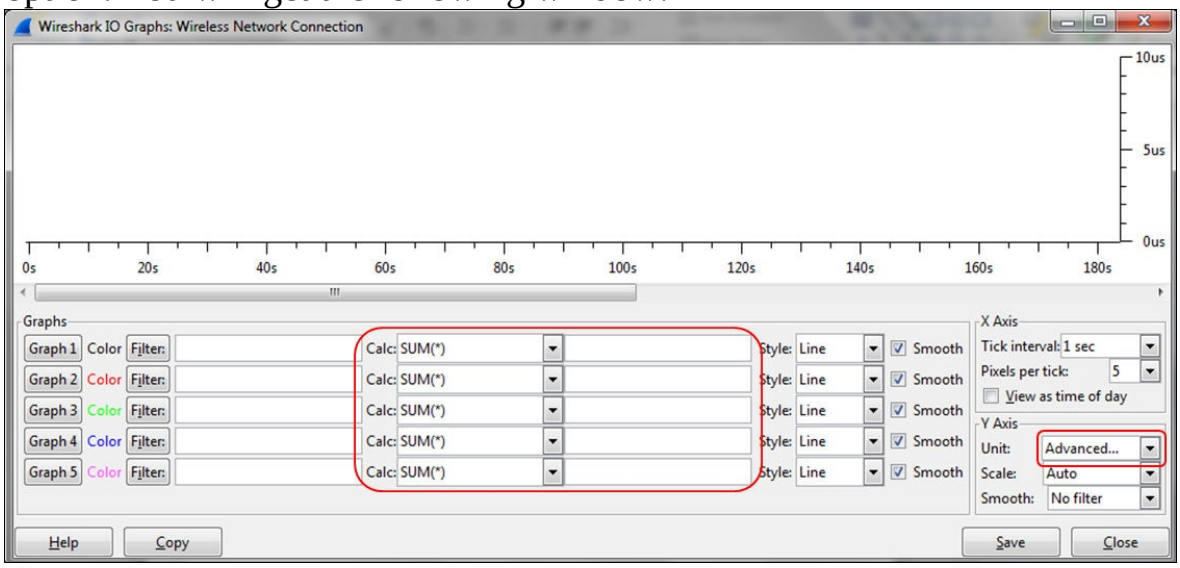
Choosing the **Advanced...** feature from the **Unit:** drop-down menu under **Y-Axis** opens a wider **IO Graphs** window, and provides the following options:

- **SUM (*)**: This draws a graph with the summary of a parameter in the tick interval
- **COUNT FRAMES (*)**: This draws a graph that counts the occurrence of the filtered frames in the tick interval
- **COUNT FIELDS (*)**: This draws a graph that counts the occurrence of the filtered field in the tick interval
- **MAX (*)**: This draws a graph with the maximum of a parameter in the tick interval
- **MIN (*)**: This draws a graph with the minimum of a parameter in the tick interval
- **AVG (*)**: This draws a graph with the average of a parameter in the tick interval
- **LOAD (*)**: This is used for response time graphs

How to do it...

To start using the **IO Graphs** window with the **Advanced** feature, perform the following steps:

1. Start the **IO Graphs** window from the **Statistics** menu.
2. In the **Unit:** drop-down menu under **Y-Axis**, choose the **Advanced…** option. You will get the following window:



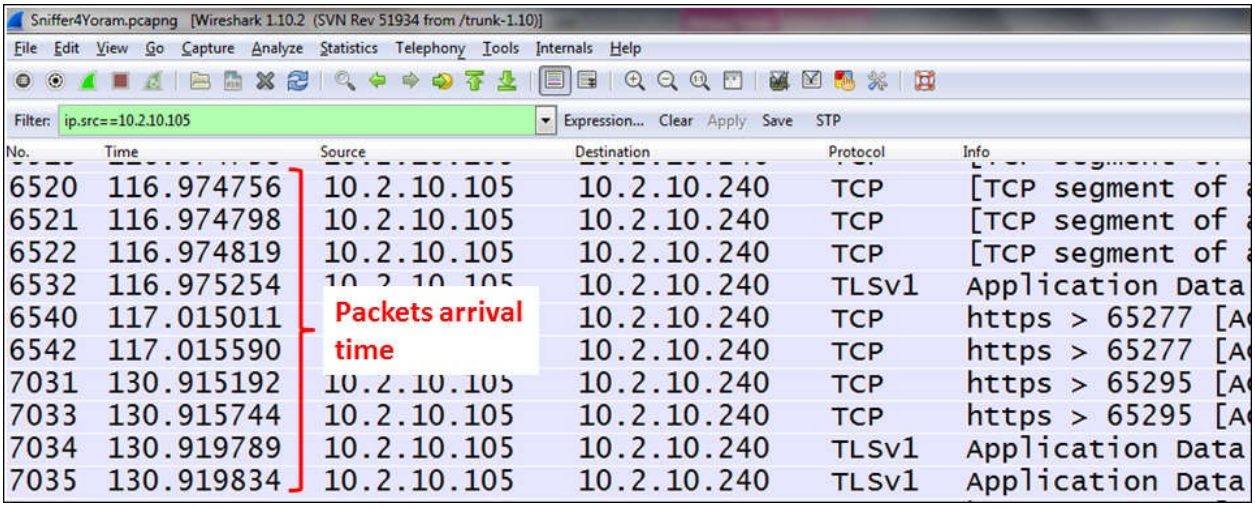
- You will see new drop-down menus with the string **SUM(*)**.
- Choose **SUM(*)/COUNT FRAMES (*)/COUNT FIELDS (*)/MAX(*)/MIN(*)/AVG(*)/ LOAD(*)**, and configure the appropriate filters.

In the next recipes we will see some useful examples.

How to monitor inter-frame time delta statistics

The time delta between frames can influence TCP performance, and there are cases in which we would like to correlate these with the performance we get from the network.

Let's look at the following capture file:

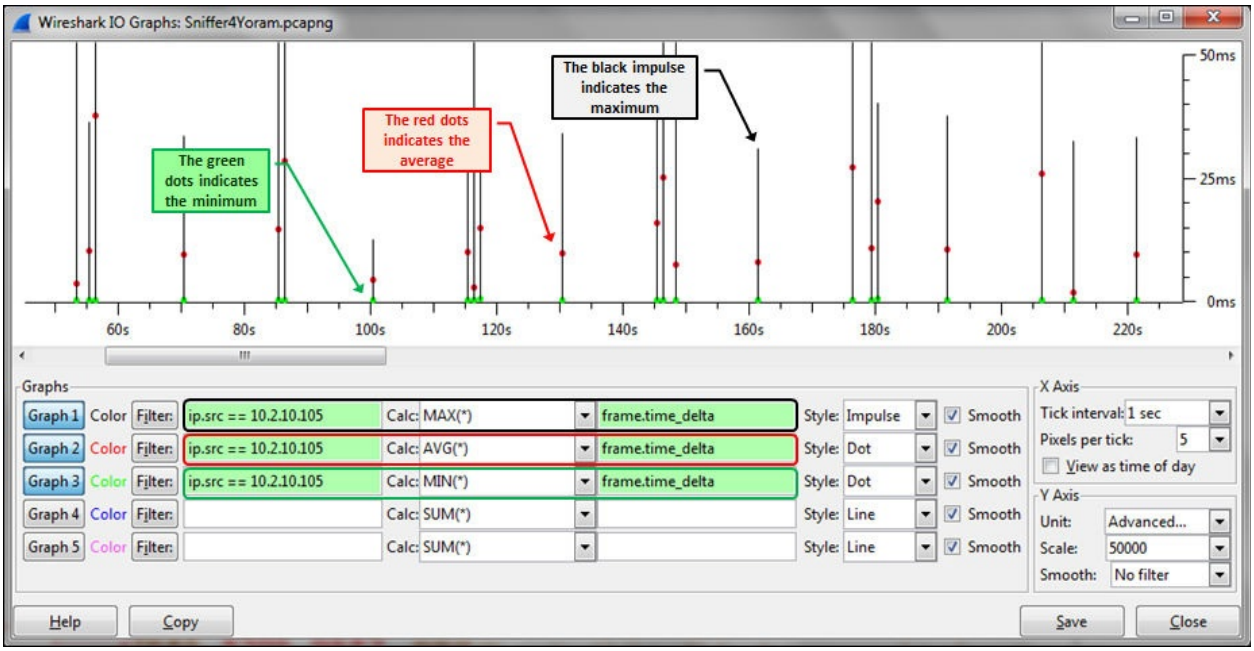


Here, we see packets sent from the source IP 10.2.10.105 as configured in the display filter.

To view the time variance between frames, configure the following parameters:

- To view the maximum `frame.time_delta` value, configure `ip.src == 10.2.10.105` in the field beside **Filter:** and choose **MAX(*)** and type `frame.time_delta` in the fields beside **Calc:**
- To view the average `frame.time_delta` value, configure `ip.src == 10.2.10.105` in the field beside **Filter:** and choose **AVG(*)** and type `frame.time_delta` in the fields beside **Calc:**
- To view the minimum `frame.time_delta` value, configure `ip.src == 10.2.10.105` in the field beside **Filter:** and choose **MIN(*)** and type `frame.time_delta` in the fields beside **Calc:**

The graph that we will get is as follows:



What we see in the screenshot is a graph of the minimum, average, and maximum time delta between frames. What do we do with it and how do we use it for network debugging? This will be covered in Chapter 10, *HTTP and DNS*.

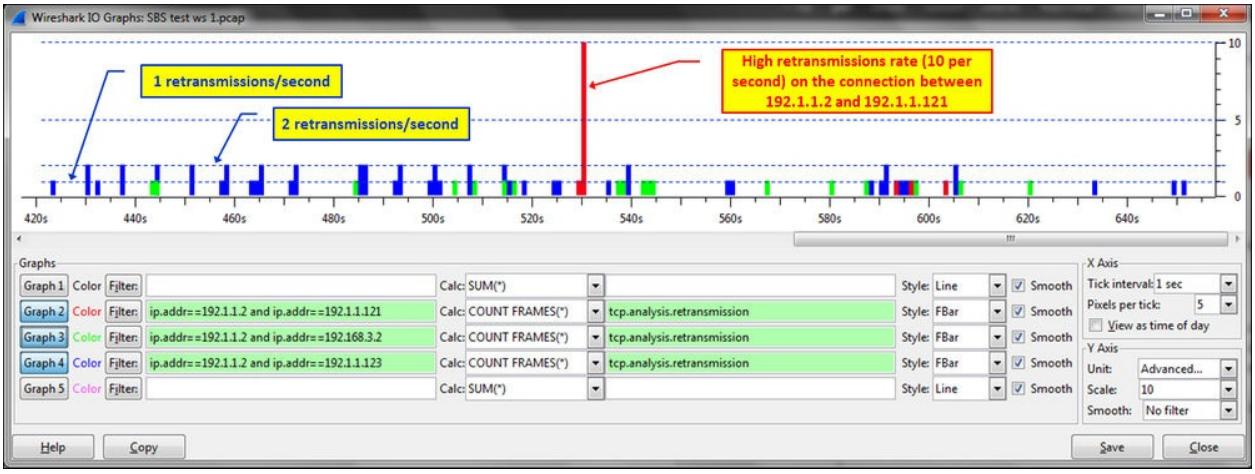
How to monitor the number of TCP retransmissions in a stream

TCP events can be of many types: retransmissions, sliding window events, ACKs (or lack of them), and others. To see the number of TCP events over time, we can use the IO Graph tool with the **Advanced...** feature and the **COUNT(*)** parameter.

To do this, perform the following steps:

1. Open **IO Graphs** from the **Statistics** menu.
2. Under **Y-Axis**, choose **Advanced...** for **Unit:**.
3. Configure the filters as follows:
    - IP source and destination filters in the fields beside the **Filter:** buttons
    - TCP events in the fields to the left of **Style:**
    - Choose **COUNT FRAMES (*)** in the **Calc:** field and type `tcp.analysis.retransmissions` in the filter field

In this example, filters were configured to monitor TCP retransmissions on three different TCP streams.



In the graph of the preceding screenshot, you can see that retransmissions from each TCP stream are presented in different colors.

How to monitor a number of field appearances

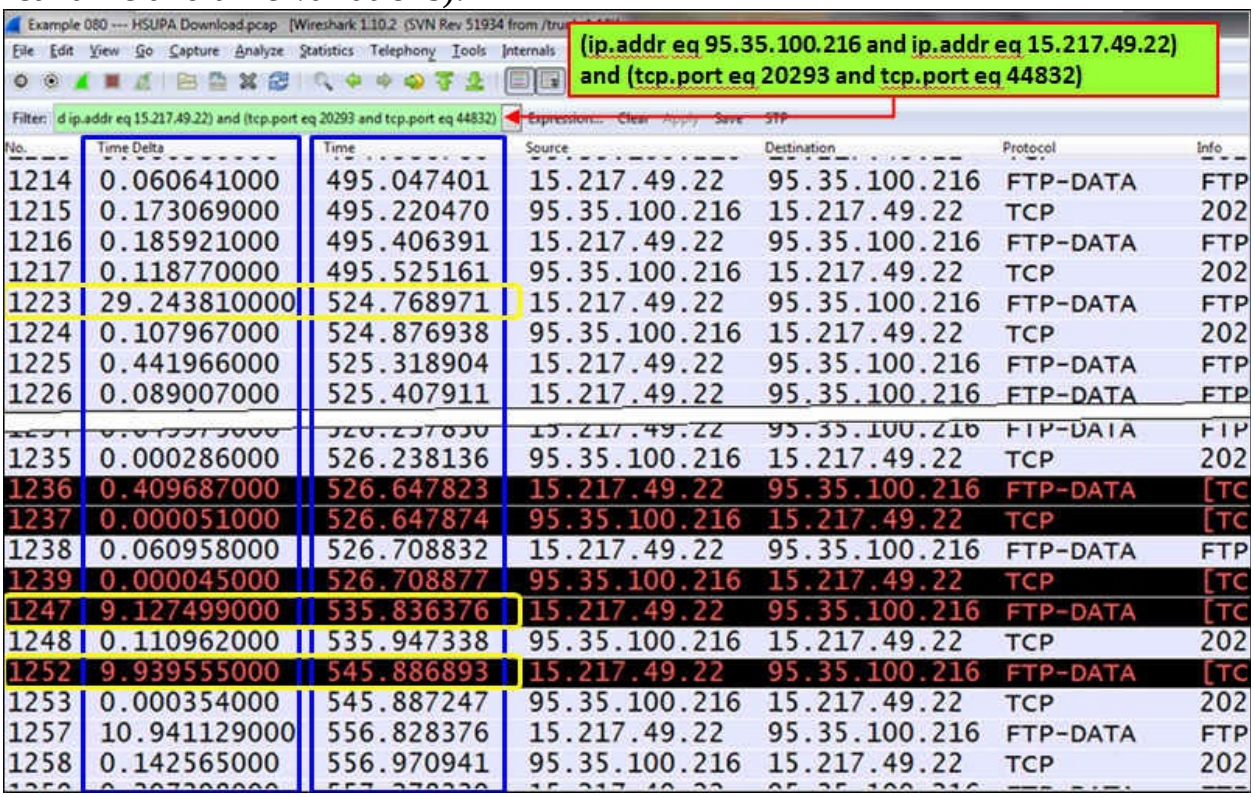
In various network protocols (mostly on those running over TCP), variations in time between frames (that is, the frame-time delta filter) can influence the performance significantly. One of the tools for viewing these changes in the **IO Graphs** window is the **Advanced...** configuration.

To do it, perform the following steps:

1. Right-click on a packet in the suspicious TCP stream and navigate to **Conversation filter** | **TCP**. A filter will appear in the main filter box.
2. Open **IO Graph** from the **Statistics** menu.
3. Under **Y-Axis**, choose **Advanced...** for **Unit:**.
4. Configure the filters as follows:
    - Copy the filter definition from the upper filter box on the right-hand side to the IO Graph filter box on the left-hand side
    - On the left-hand side, type the filter `frame.time_delta`
    - Choose **AVG(*)** to see the average delta.

- Choose the appropriate X-Axis resolution.

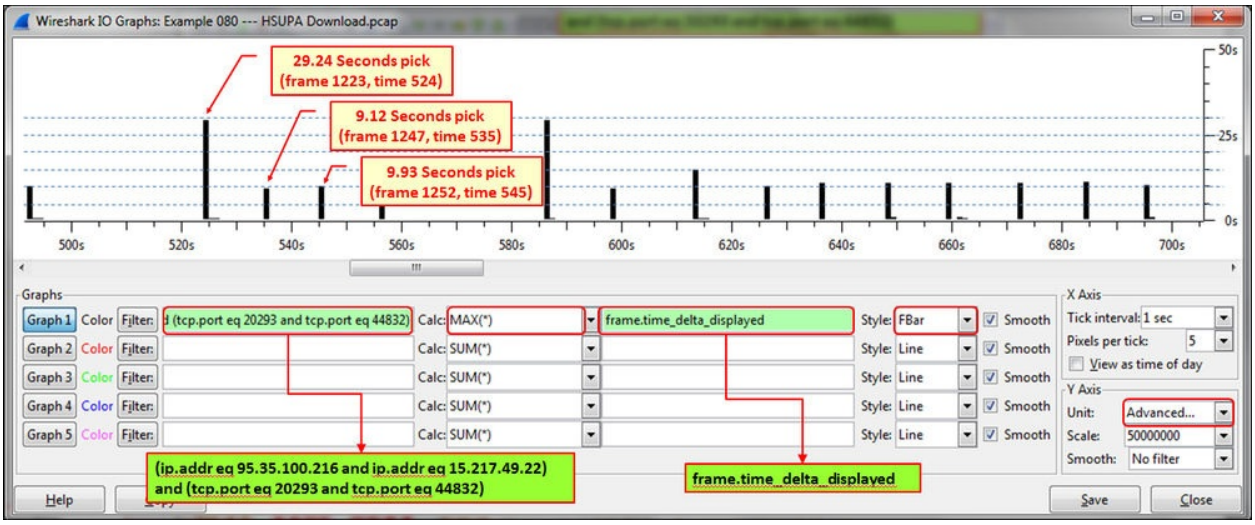
Here is an example. In the following screenshot, we see a packet list with time variations between frames (a second time column was added in order to see the real time and time variations):



No.	Time Delta	Time	Source	Destination	Protocol	Info
1214	0.060641000	495.047401	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1215	0.173069000	495.220470	95.35.100.216	15.217.49.22	TCP	202
1216	0.185921000	495.406391	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1217	0.118770000	495.525161	95.35.100.216	15.217.49.22	TCP	202
1223	29.243810000	524.768971	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1224	0.107967000	524.876938	95.35.100.216	15.217.49.22	TCP	202
1225	0.441966000	525.318904	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1226	0.089007000	525.407911	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1235	0.000286000	526.238136	95.35.100.216	15.217.49.22	TCP	202
1236	0.409687000	526.647823	15.217.49.22	95.35.100.216	FTP-DATA	[TC
1237	0.000051000	526.647874	95.35.100.216	15.217.49.22	TCP	[TC
1238	0.060958000	526.708832	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1239	0.000045000	526.708877	95.35.100.216	15.217.49.22	TCP	[TC
1247	9.127499000	535.836376	15.217.49.22	95.35.100.216	FTP-DATA	[TC
1248	0.110962000	535.947338	95.35.100.216	15.217.49.22	TCP	202
1252	9.939555000	545.886893	15.217.49.22	95.35.100.216	FTP-DATA	[TC
1253	0.000354000	545.887247	95.35.100.216	15.217.49.22	TCP	202
1257	10.941129000	556.828376	15.217.49.22	95.35.100.216	FTP-DATA	FTP
1258	0.142565000	556.970941	95.35.100.216	15.217.49.22	TCP	202

You see that there are some large time variations between frames; for example, 29.24 seconds in the frame 1,223, 9.12 seconds in the frame 1,247, and more.

In the **IO Graphs** window configured as described earlier, you will see the following:



As you see here, there are variations in time between frames. Later in this book, we will learn to see what causes these problems and how to solve them.

How it works...

The IO Graph tool is one of the strongest and most efficient tools of Wireshark. While the standard IO Graph statistics can be used for basic statistics, the **Advanced…** feature can be used for in-depth monitoring of response times, TCP analysis of a single stream or several streams, and more.

When we configure a filter on the left, we will filter the traffic between hosts, traffic in a connection, traffic on a server, and so on. The **Advanced…** feature provides us with more details on traffic. Here are a few examples:

- On the left you see the TCP stream; on the right you see the time delta between frames in the stream
- On the left you see the video/RTP stream; on the right you see the occurrence of a marker bit

There's more...

You can always click on **IO Graph**, and it will bring you to the reference packet in the packet pane.

Getting information through TCP stream graphs – the Time-Sequence (Stevens) window

One of the tools in Wireshark that enables us to dig deeper into applications behavior is the TCP stream graphs. These graphs, as we will see in the following recipes, enable us to get the filling of the application behavior along with the possibility to locate problems in it.

Getting ready

Open an existing capture or start a new capture. Click on a specific packet in the capture file. Even though you can use this feature on a running capture, it is not meant for online statistics; so it is recommended that you start a capture, stop it, and then use this tool.

How to do it...

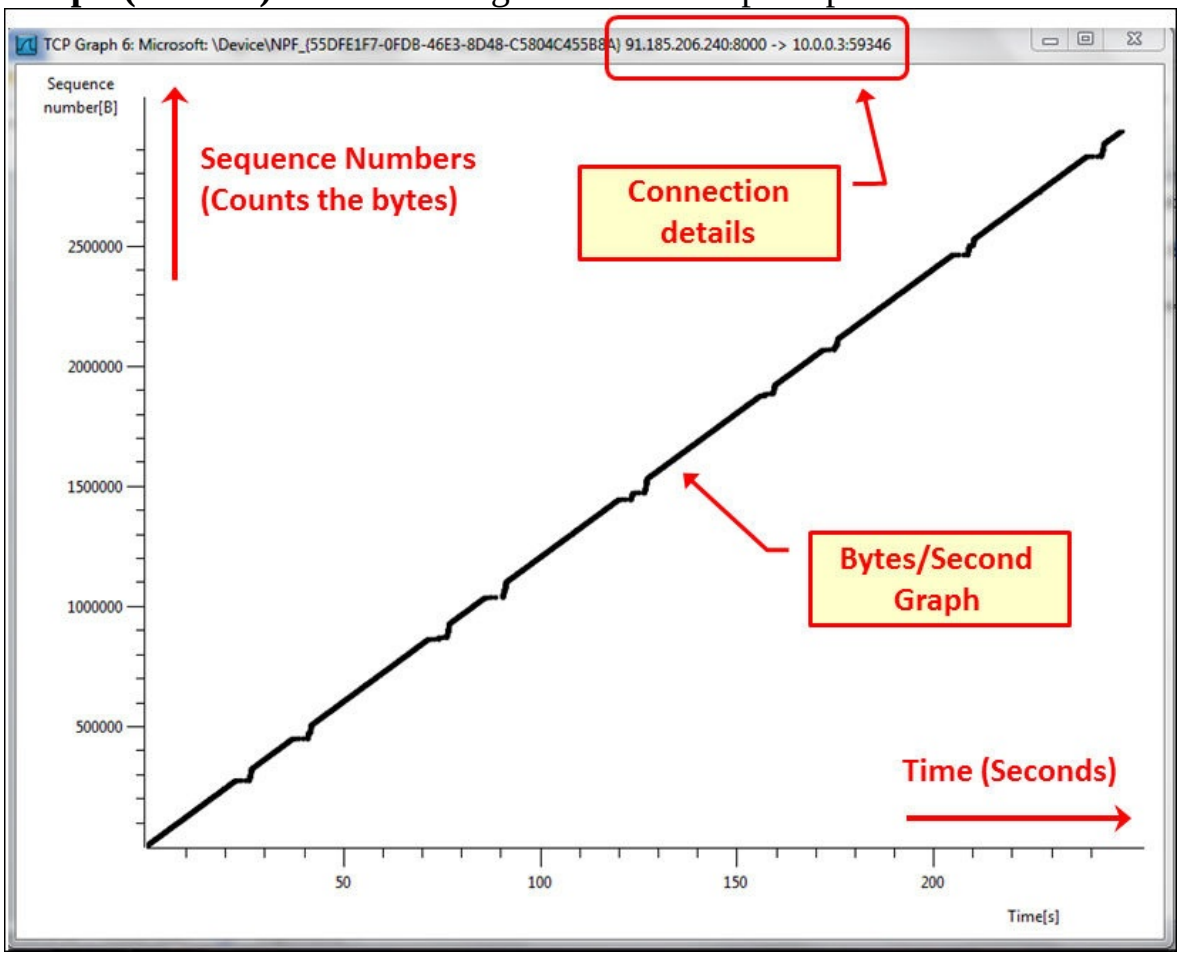
To view TCP stream graph statistics, perform the following steps:

1. Click on the packet of the stream you want to monitor.

   Tip

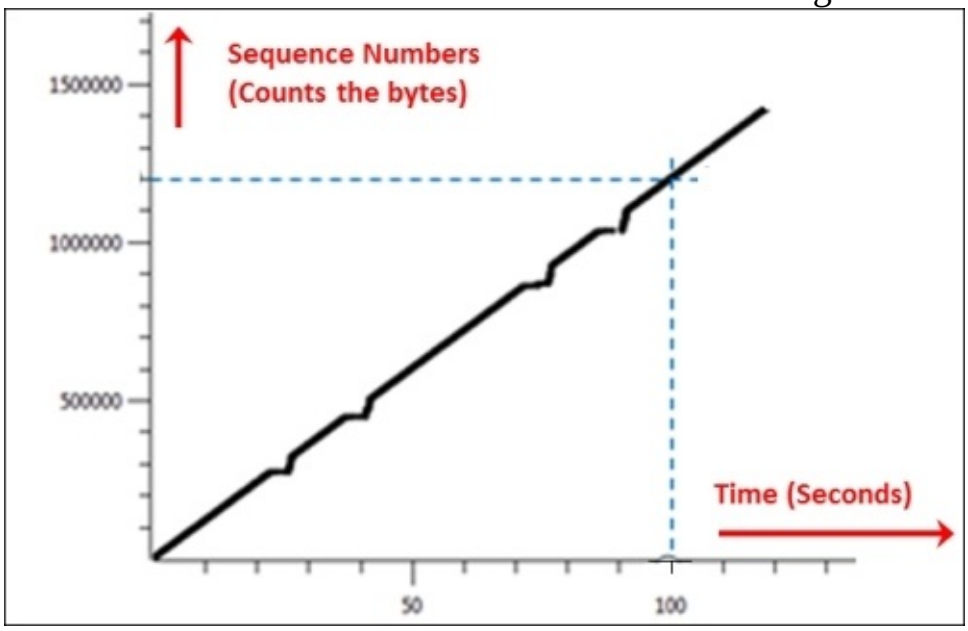
   The TCP Stream shows a directional graph, so when you click on a packet, it should be in the direction you want to view the statistics on. If, for example, you download a file and want to view the download statistics, click on a packet in the download direction.

2. From the **Statistics** menu navigate to **TCP StreamGraph | Time-Sequence Graph (Stevens)**. The following window will open up:



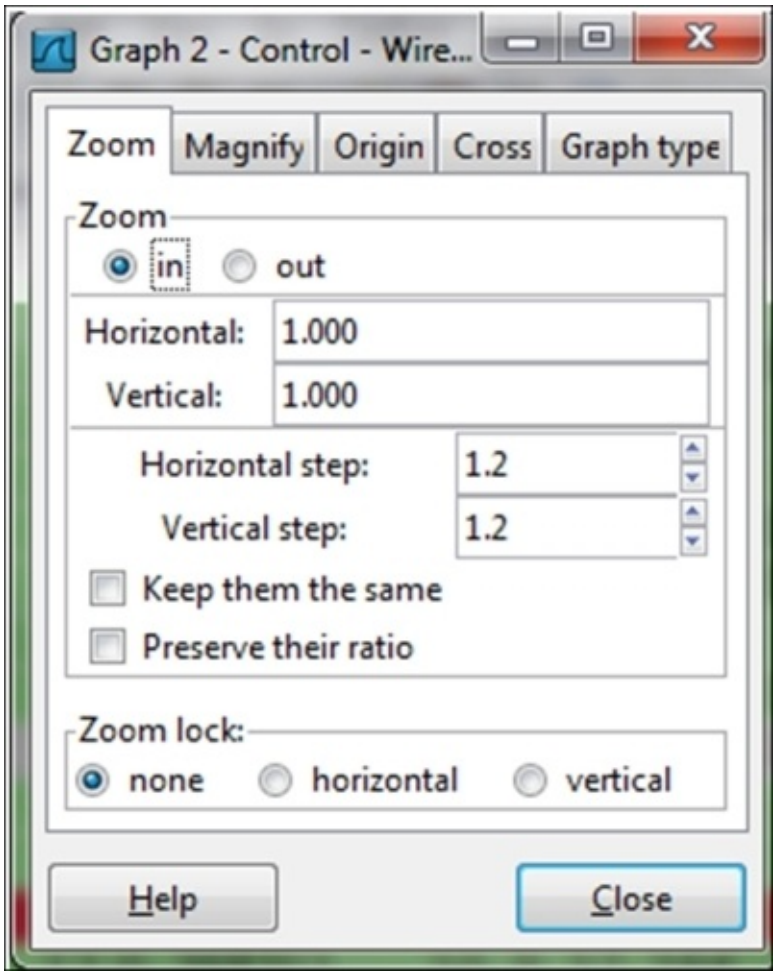
The graph actually shows the advance of byte transfer over time. In this example we see a continuous diagonal line, which is an indication of a good file transfer.

To measure the throughput of a file transfer, simply calculate the bytes transferred in a unit of time as shown in the following screenshot:



We see that the transfer rate is 1,200,000 bytes in 100 seconds, that is, 12,000 bytes/seconds or 95 Kbits/sec.

3. Clicking on a point in the graph using the scrollbar will magnify the graph around the point that you clicked on.
4. Right-clicking on a point in the graph will take us to the packet pane in the captured file.
5. For changing graph parameters, we have a small window opened parallel to the graph as shown in the following screenshot:



6. For changing from zoom in to zoom out, click on the **in** or **out** button.

How it works...

The Time-Sequence Graph (Stevens) is a simple graph that counts the TCP sequence numbers over time. Since TCP sequence numbers count the bytes sent by TCP, these are actually application bytes (including application headers) sent from one side to another.

This graph (as we will learn in the TCP and applications chapters) can give us a good indication of the application's behavior. For example, a diagonal line means a good file transfer, while a diagonal line with interrupts shows a problem in transfer. A diagonal line with a high gradient indicates fast data transfer, while a low gradient indicates a low rate of transfer (depends on the scale of course).

There's more...

Left-clicking on a point in the graph will take you to the packet in the packet pane. When you see a problem, zoom into it, left-click on it, and check what went wrong with the packets.

While viewing a graph, it is important to know what the application is. A graph that indicates a problem in one application can be a perfect network behavior for another application.

Getting information through TCP stream graphs – the Time-Sequence (tcp-trace) window

TCP time-sequence graphs based on the UNIX `tcpdump` command provide us with additional data on the connection that we monitor. In addition to the standard sequence/seconds in Time-Sequence (Stevens), we also get information on the ACKs that were sent, retransmissions, window size, and more details that enables us to analyze problems on the connection.

Getting ready

Open an existing capture or start a new capture. Click on a specific packet in the capture file. Even though you can use this feature on a running capture, it is not meant for online statistics; so it is recommended that you start a capture, stop it, and then use this tool.

How to do it...

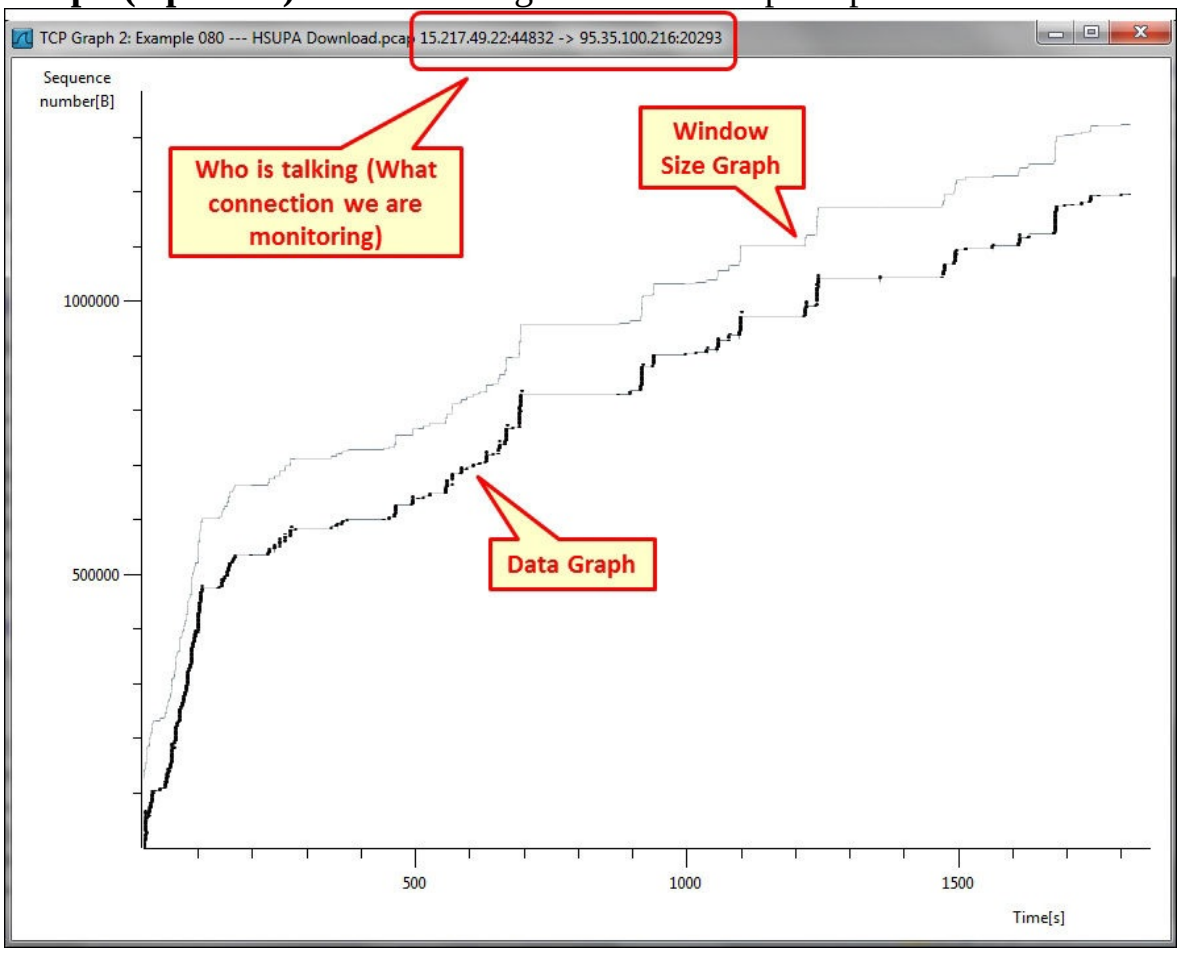
To view TCP stream graph statistics, perform the following steps:

1. Click on a packet in the stream you want to monitor.

   Tip

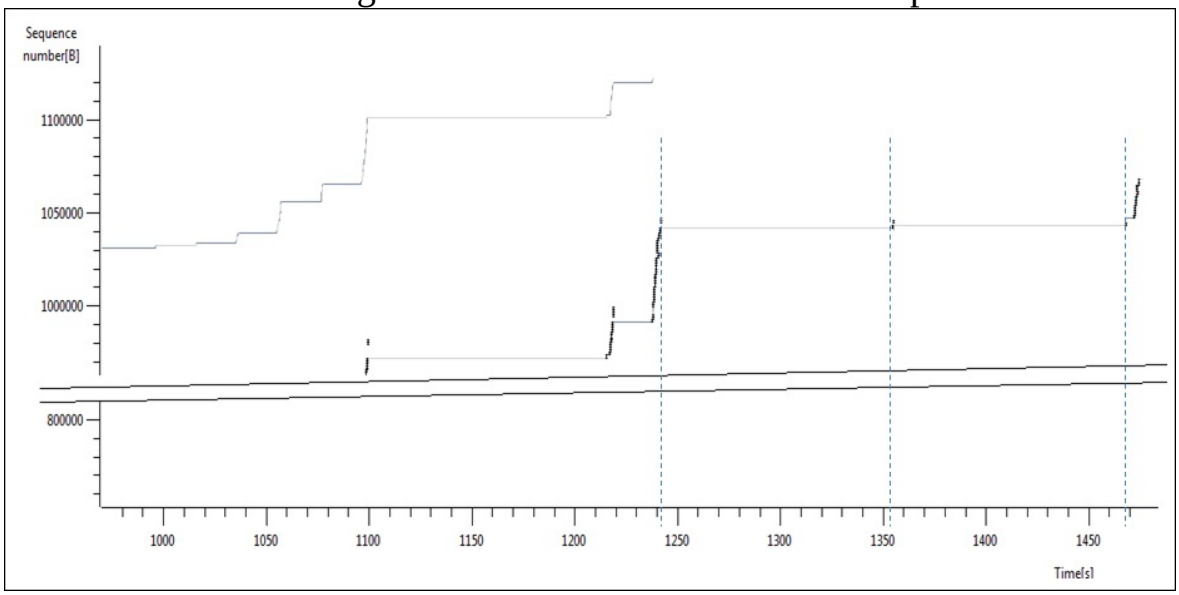
   The TCP stream shows a directional graph, so when you click on a packet, it should be in the direction you want to view the statistics on. If, for example, you download a file and want to view the download statistics, click on a packet in the download direction.

2. From the **Statistics** menu navigate to **TCP StreamGraph | Time-Sequence Graph (tcp-trace)**. The following window will open up:

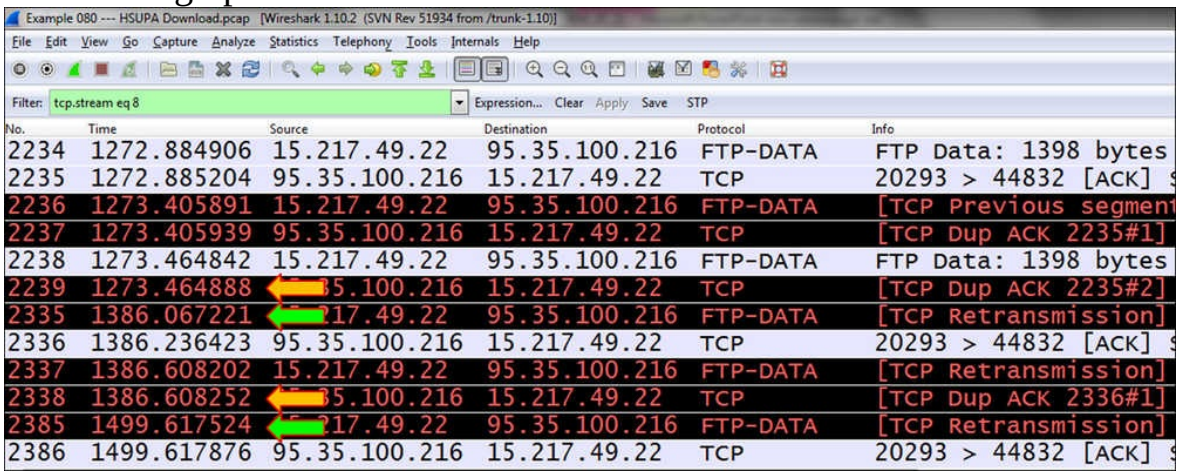


The graph shows the advance of byte transfer over time in the lower black graph and the window size in the upper gray graph. When there is space between the two, it means that there is some TCP buffering left and TCP will transfer bytes. Once they get closer and touch each other, it would be a window-full phenomenon that does not enable further data transfer.

3. We obtain the following screenshot when we zoom into a specific area:



4. We obtain the following captured packets when we zoom into a particular area in the graph:



You can see that in the packet capture, there is a frame in time 1,273

(seconds after the beginning of the capture), a break, a packet in time 1,386, a break, and a packet in 1499.

In the TCP stream graph you see the breaks in transmission, and we can look for its reason when we are back to the packets pane.

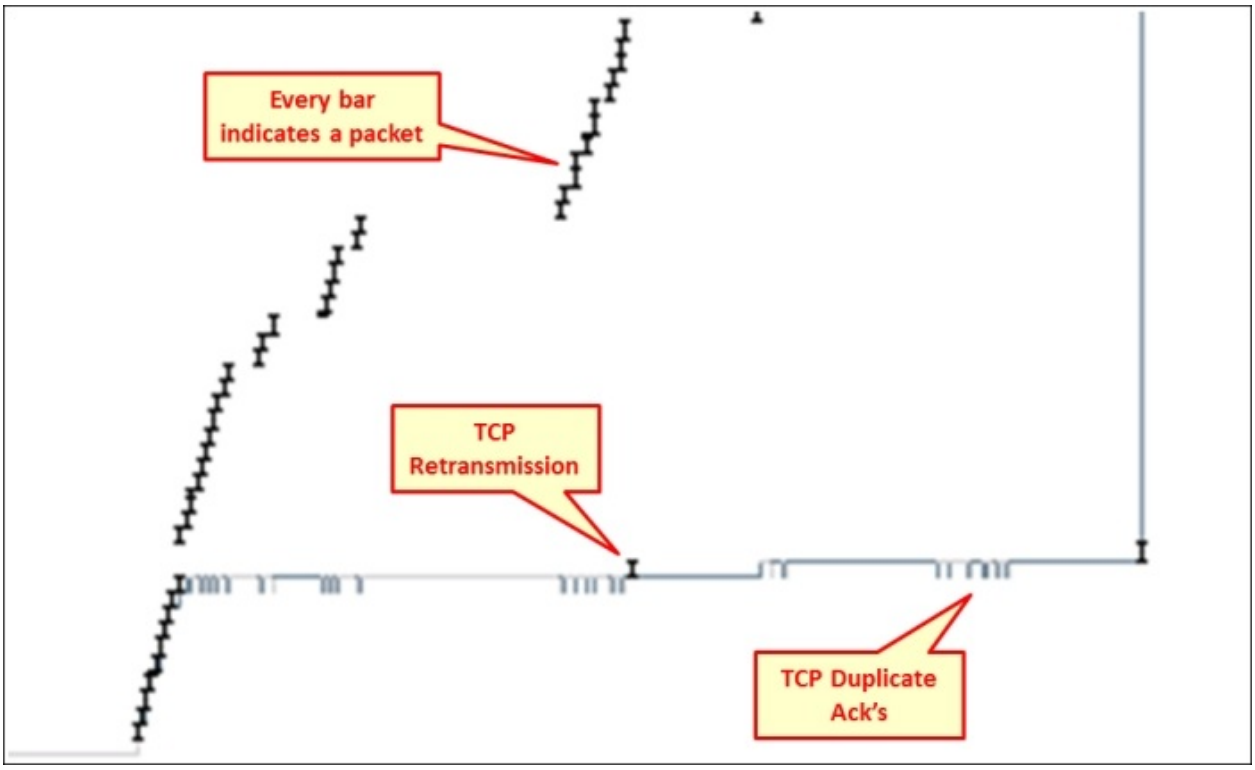
How it works...

The **Time sequence** (**TCP-trace**) graph is taken from the UNIX `tcpdump` command, which also checks the window size published by the receiver (this is the buffer size allocated by the receiver to the process), along with retransmitted packets and ACKs.

Working with this graph provides us with a lot of information, which we will use later for network debugging. The phenomena from a window that is being filled faster than expected to a lot of retransmissions and others will become visual with this graph that will help us to solve them.

There's more...

The more we zoom in, the more details we will get as shown in the following screenshot:



A bar is an indication of a packet that carries data between the initial and final sequence numbers. The bar that is not in the regular graph and looks like it runs away from it is a retransmission and the gray bar is a duplicate ACK. We will learn about these phenomena in Chapter 9, *UDP/TCP Analysis*.

Getting information through TCP stream graphs – the Throughput Graph window

The **Throughput Graph** window of the TCP stream graphs enables us to look at the throughput of a connection and check for instabilities.

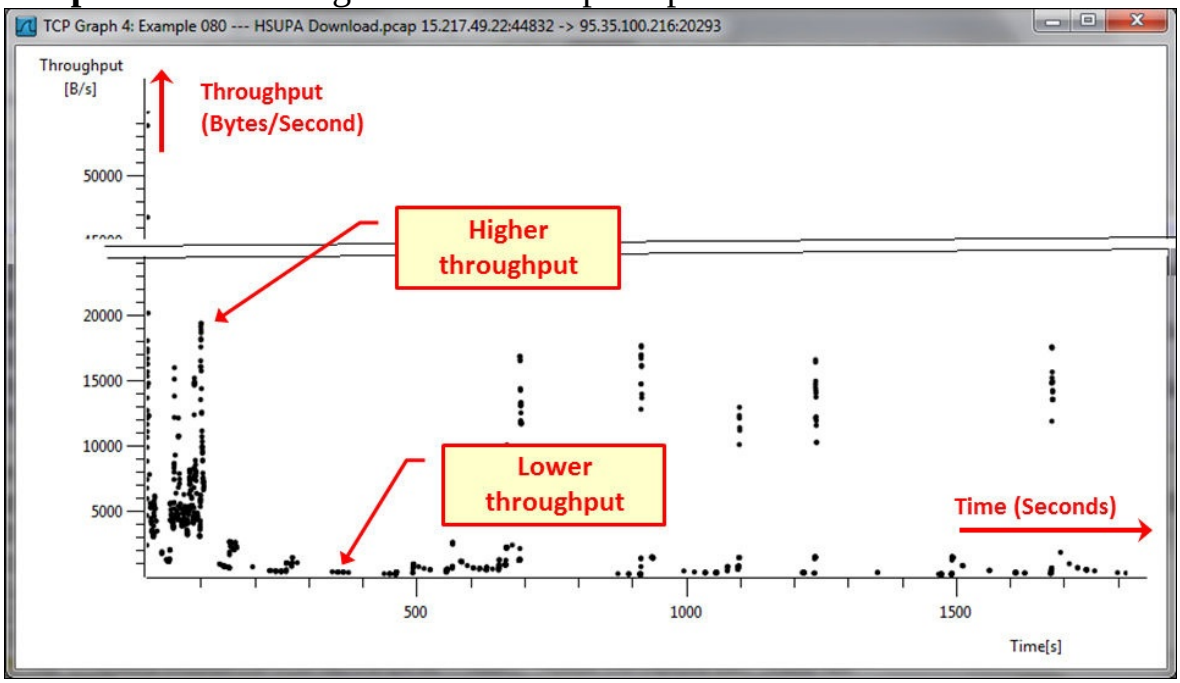
Getting ready

Open an existing capture or start a new capture. Click on a specific packet in the capture file. Even though you can use this feature on a running capture, it is not meant for online statistics; so it is recommended that you start a capture, stop it, and then use this tool.

How to do it...

To view TCP stream graph statistics, perform the following steps:

1. Click on a packet in the stream you want to monitor.
2. From the **Statistics** menu, navigate to **TCP StreamGraph** | **Throughput Graph**. The following window will open up:



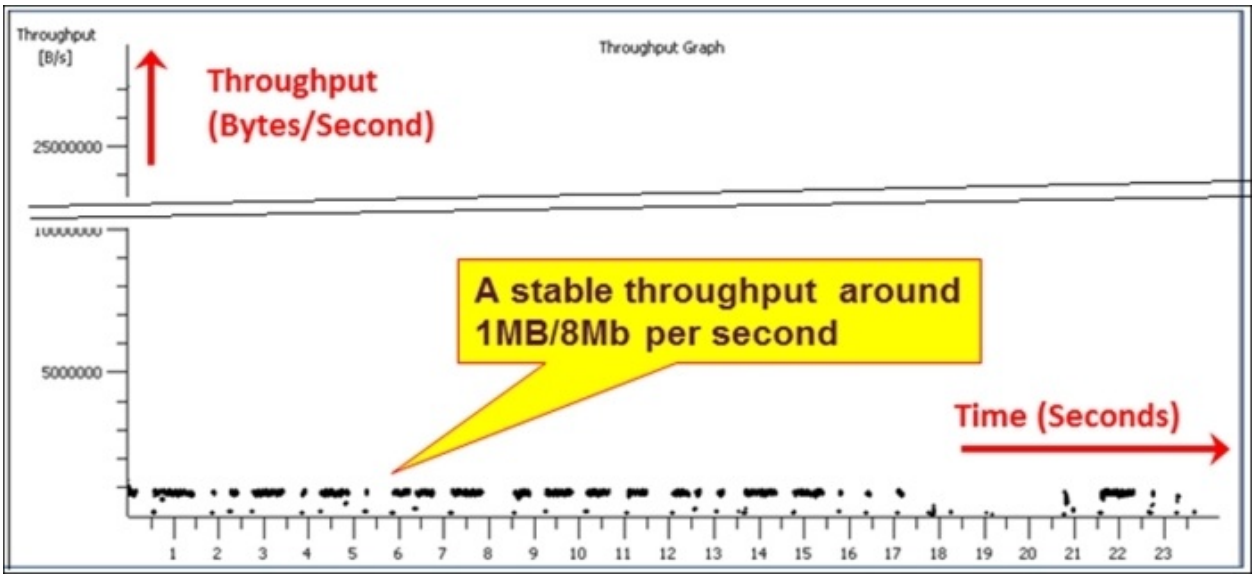
In the graph, we see that the throughput is not stable and varies between around 20,000 bytes/sec to 1000 bytes/sec. This can be due to an unstable file transfer (which is the case in this FTP download over the HSUPA cellular connection), or just an application that works this way (for example, browsing the Internet).

How it works...

The throughput graph simply counts the TCP sequence numbers over time and since sequence numbers are actually the application's data, this gives us the application throughput in bytes per second.

There's more...

A stable file transfer should look almost like a solid line, as shown in the following graph:



Here, MB is mega bytes and Mb is mega bits.

Getting information through TCP stream graphs – the Round Trip Time window

The **Round Trip Time** window of the TCP stream graphs enables us to look at the round trip between sequence numbers and the time they were acknowledged. Along with other graphs, it provides us with a look at the performance of the connection.

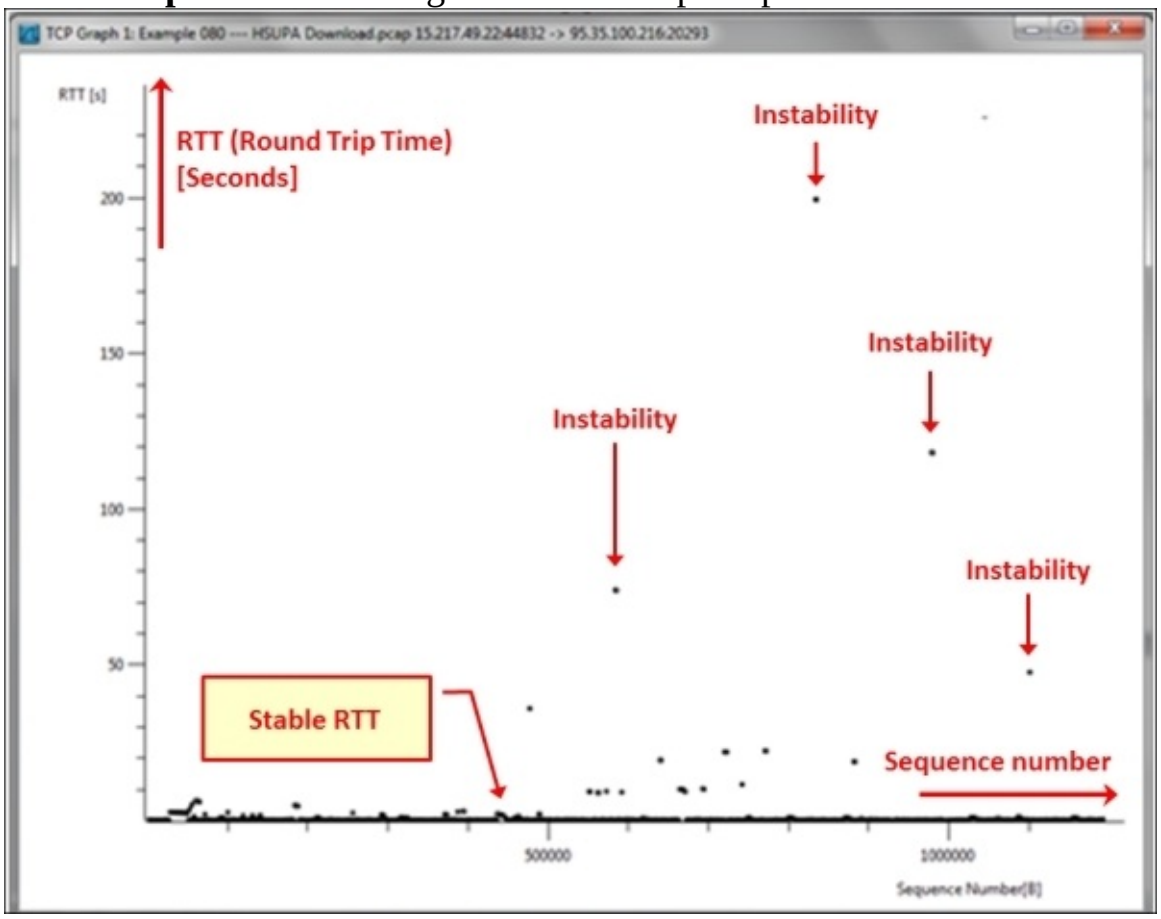
Getting ready

Open an existing capture or start a new capture. Click on a specific packet in the capture file. Even though you can use this feature on a running capture, it is not meant for online statistics, so it is recommended that you start a capture, stop it, and then use this tool.

How to do it...

To view the TCP stream graph statistics, perform the following steps:

1. Click on a packet in the stream you want to monitor.
2. From the **Statistics** menu navigate to **TCP StreamGraph** | **Round Trip Time Graph**. The following window will open up:



In the preceding graph, we see that most of the sequence numbers were acknowledged in a short time; however, there is some instability that will influence the TCP performance.

How it works...

What we see in the graph is a plot of TCP sequence numbers versus the time that took to acknowledge them. Actually, this is the time between a sent packet and the ACK received for that packet.

There's more...

When you see a graph that shows instabilities, it's not necessarily a problem. It can also be that this is how the application works. You can see that it took time to acknowledge a packet because there is a problem, or because a server is waiting for a response, or because a client is browsing a web server and the user is waiting between clicks on new links.

Getting information through TCP stream graphs – the Window Scaling Graph window

The Window Scaling Graph of the TCP stream graph enables us to look at the window size published by the receiving side, which is an indication of the receiver's ability to process data. Along with the other graphs, it provides us with a look at the performance of the connection.

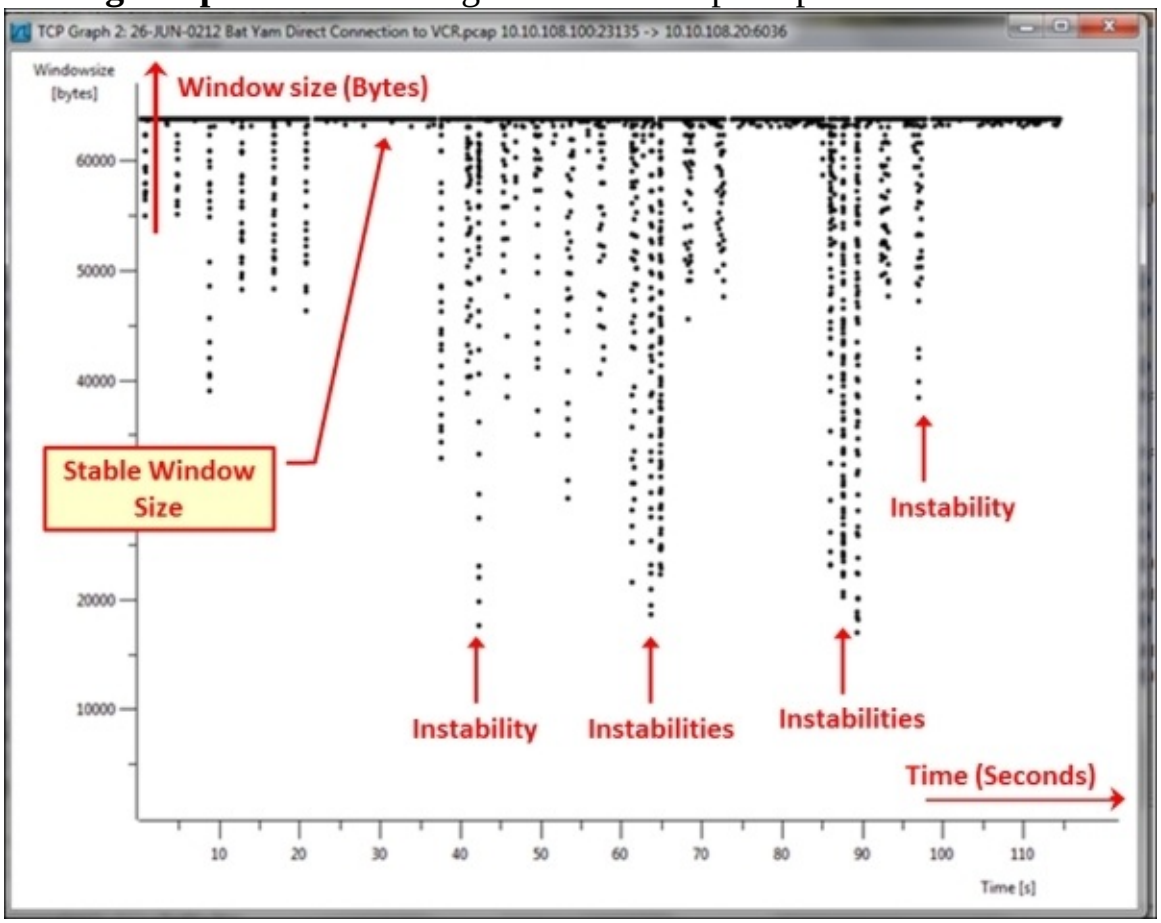
Getting ready

Open an existing capture or start a new capture. Click on a specific packet in the capture file. Even though you can use this feature on a running capture, it is not meant for online statistics, so it is recommended that you start a capture, stop it, and then use this tool.

How to do it...

To view TCP stream graph statistics, perform the following steps:

1. Click on a packet in the stream you want to monitor.
2. From the **Statistics** menu navigate to **TCP StreamGraph** | **Window Scaling Graph**. The following window will open up:



In this graph, we see the instability caused by one of the sides. This can be an indication of a slow server or client that cannot process all the data it receives and therefore, by reducing the received window size, it tells the other side to send less data.

How it works...

The software here simply watches the window size on the connection and draws it. In Chapter 9, *UDP/TCP Analysis*, we will get into the details.

There's more...

When the window size decreases, the application throughput should decrease as well. The window size is completely controlled by the two ends of a connection, for example, a client and a server; variations in the window size do not have anything to do with network performance.

Chapter 6. Using the Expert Infos Window

In this chapter we will talk about the following:

- The **Expert Infos** window and how to use it for network troubleshooting
- Error events and understanding them
- Warnings events and understanding them
- Notes events and understanding them

Introduction

One of Wireshark's strongest capabilities is the ability to analyze network phenomena and suggest to us a probable cause for it. Along with other tools, it gives us detailed information on network performance and problems. In this chapter, we will learn about the Expert System. It is a tool that provides us with a deeper analysis of network phenomena, including events and problems. Later in this book, we will provide detailed recipes on how to use the **Expert Infos** window along with other tools to find and resolve network problems.

In the first recipe, we will learn how to work with the **Expert Infos** window. In the next recipes, we will learn about the probable causes for the majority of events that you can expect.

The Expert Infos window and how to use it for network troubleshooting

The **Expert Infos** window provides us with a list of events and network problems discovered by Wireshark. In this recipe, we will learn how to start the **Expert Infos** window and how to refer to the various events.

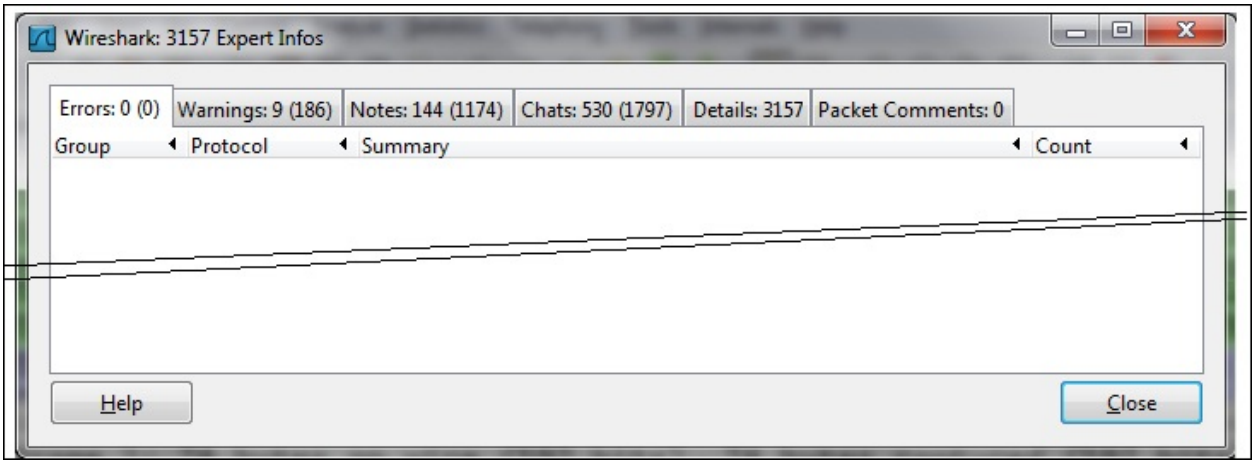
Getting ready

Start Wireshark, and start a live capture or open an existing file.

How to do it...

To start the **Expert Infos** window, perform the following steps:

Navigate to the **Analyze** menu and click on **Expert Info**. The following window will open:



Now you can choose any one of the upper bars: **Errors:**, **Warnings:**, **Notes:**, **Chats:**, **Details:**, or **Packet Comments:**.

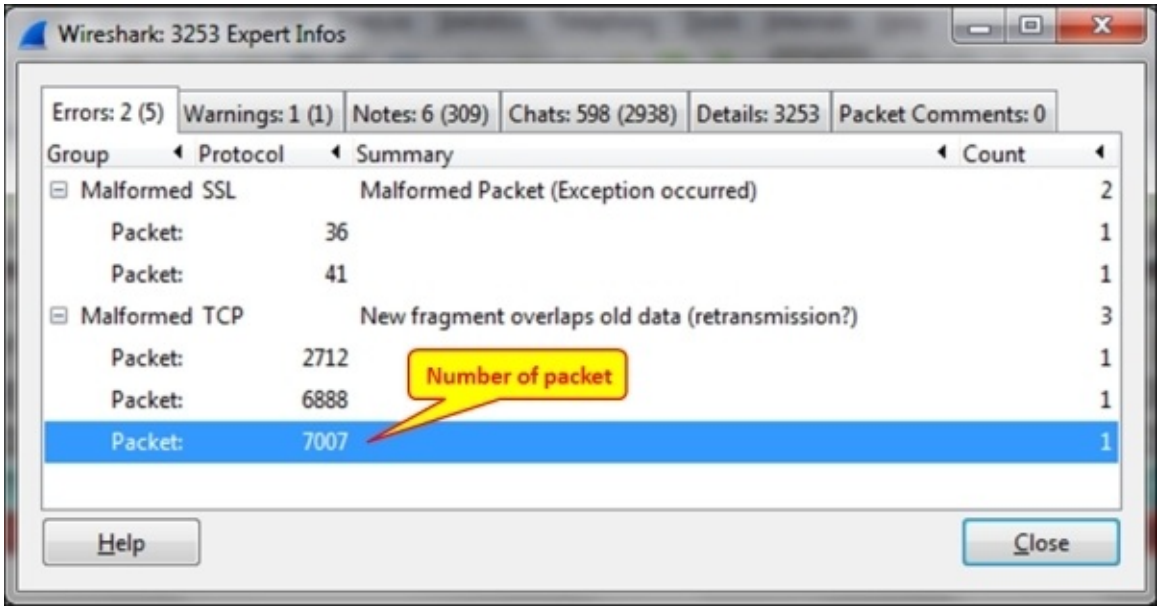
Tip

The number at the right-hand side of the bar shows the number of events in this category.

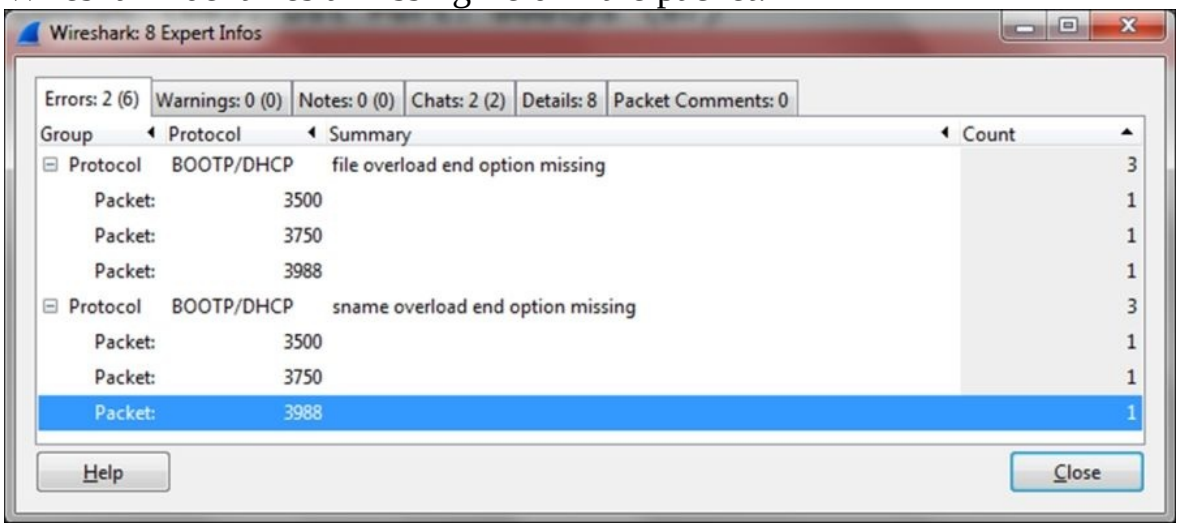
The upper bars give you the following information:

- **Errors:** These are serious problems, mostly malformed packets or missing fields in a protocol header. These can be malformed packets of various types such as malformed SPOOLSS, GTP, or others. These can also be bad checksum errors such as IPv4 bad checksum.

  In the following screenshot you can see malformed TCP and SSL packets:

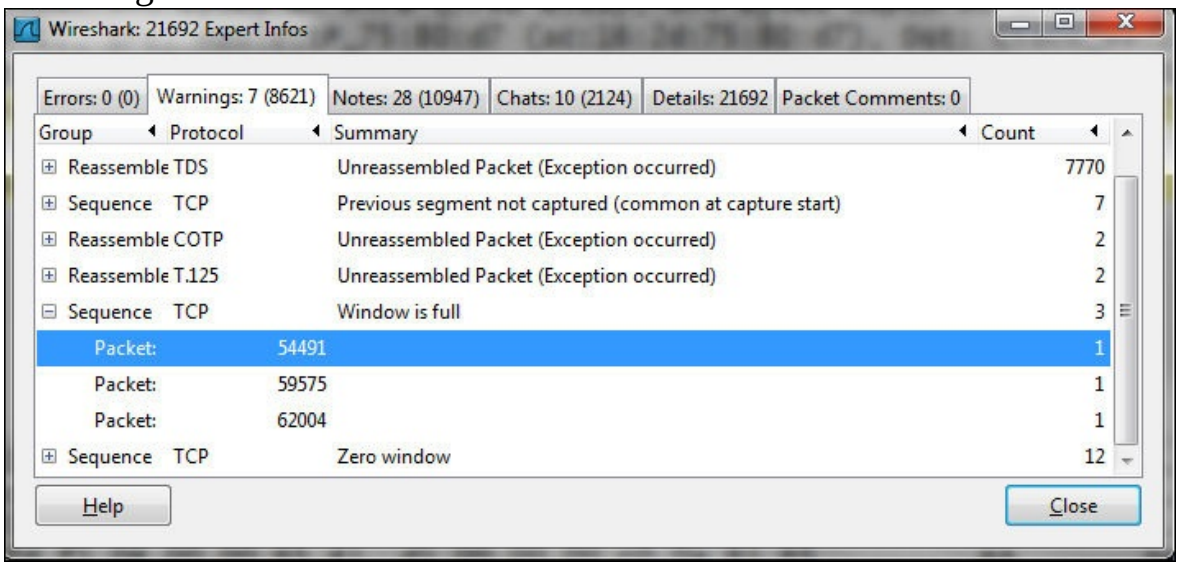


In the following screenshot, you can see another type of error, which is a protocol (in this case the **BOOTP/DHCP**) option error, that is, when Wireshark identifies a missing field in the packet:



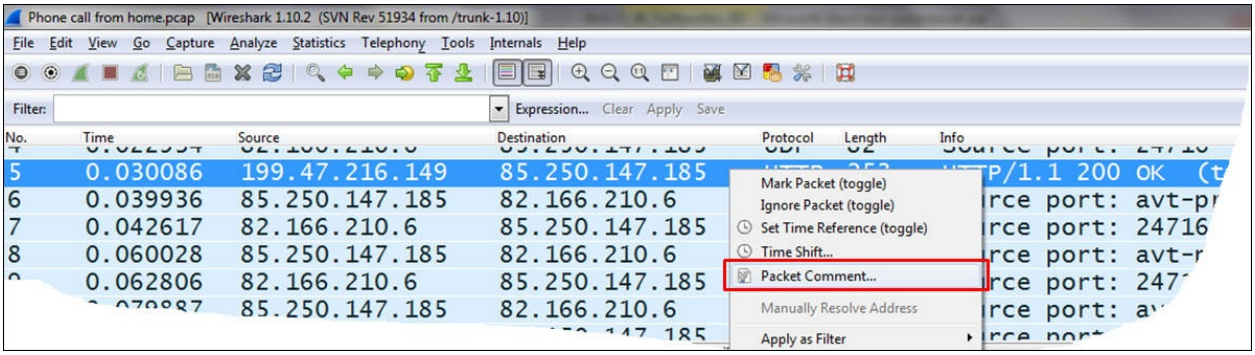
- **Warnings:** A warning indicates a problem in the application or in communication, things such as TCP zero window, TCP window full, previous segment not captured, out-of-order segment, and others that are unnatural to the protocol behavior. You can see an example of this in the

following screenshot:



- **Notes:** A note is when Wireshark indicates an event that may cause a problem, but is still within the normal behavior of the protocol. TCP retransmission, for example, will be displayed here because even though it is a critical problem that slows down the network, it is still under the normal behavior of TCP. Other events here are duplicate ACK, fast retransmission, and so on.
- **Chats:** This tab provides information about the usual workflow, for example, TCP connection start (SYN), connection end (FIN), connection reset (RST), HTTP Post, HTTP codes, and so on.
- **Details:** This tab provides all the events in an ordered list. In older versions of Wireshark, this was directly under the **Analyze** menu.
- **Packet Comments:** You can manually add a comment to every packet. This column will show all the comments in the capture file.

To add a comment to a packet, right-click on it and choose **Packet Comment...**. A window will open in which you will be able to add or change your comment. You can see this in the following screenshot:



To go to the event in the packet capture pane, simply click on the packet under the event in the **Expert Infos** window, and it will lead you to it.

Note

It is important to note that although a warning event may have no importance, a note event can influence the network badly. Always get into the problem details, see where is it coming from, and what is its meaning.

How it works...

The Wireshark **Expert Infos** window is an expert system that provides us with information about problems in the network and also some suggestions to the probable cause of it in some cases. Although it gives reasonable results, always double-check its findings.

There are cases where Wireshark finds problems that are not genuine, and there are other cases where the real problems that exist do not show up.

Tip

Don't forget that the best troubleshooting tool is your brain (and your knowledge of networking). Wireshark is a very smart tool, but still it is only a tool.

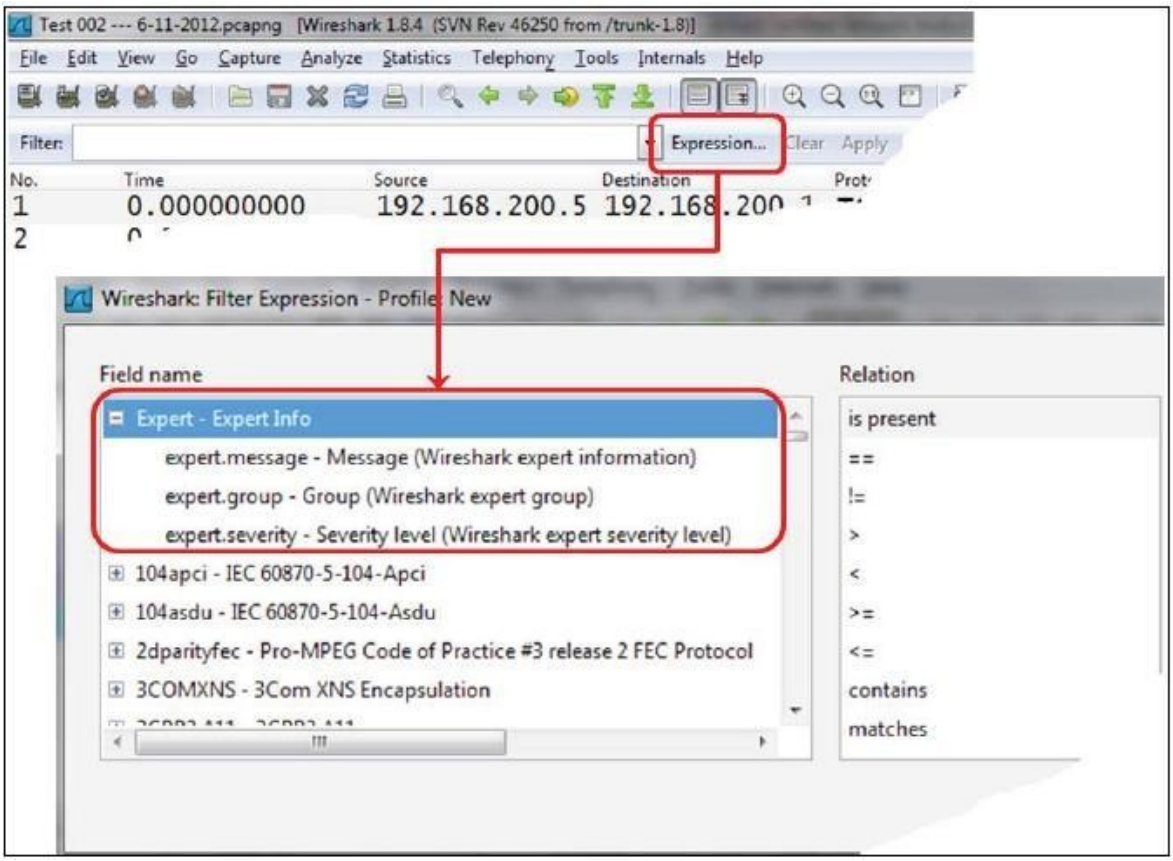
It can be that you started the capture during a data transfer; so you will see the previous segment's loss messages or even more sophisticated problems, when for some reason (good or not) you have captured only a part of the data, and Wireshark refers to it as a complete stream of data and displays many errors about it. We will see many examples of these issues later in the book.

There's more...

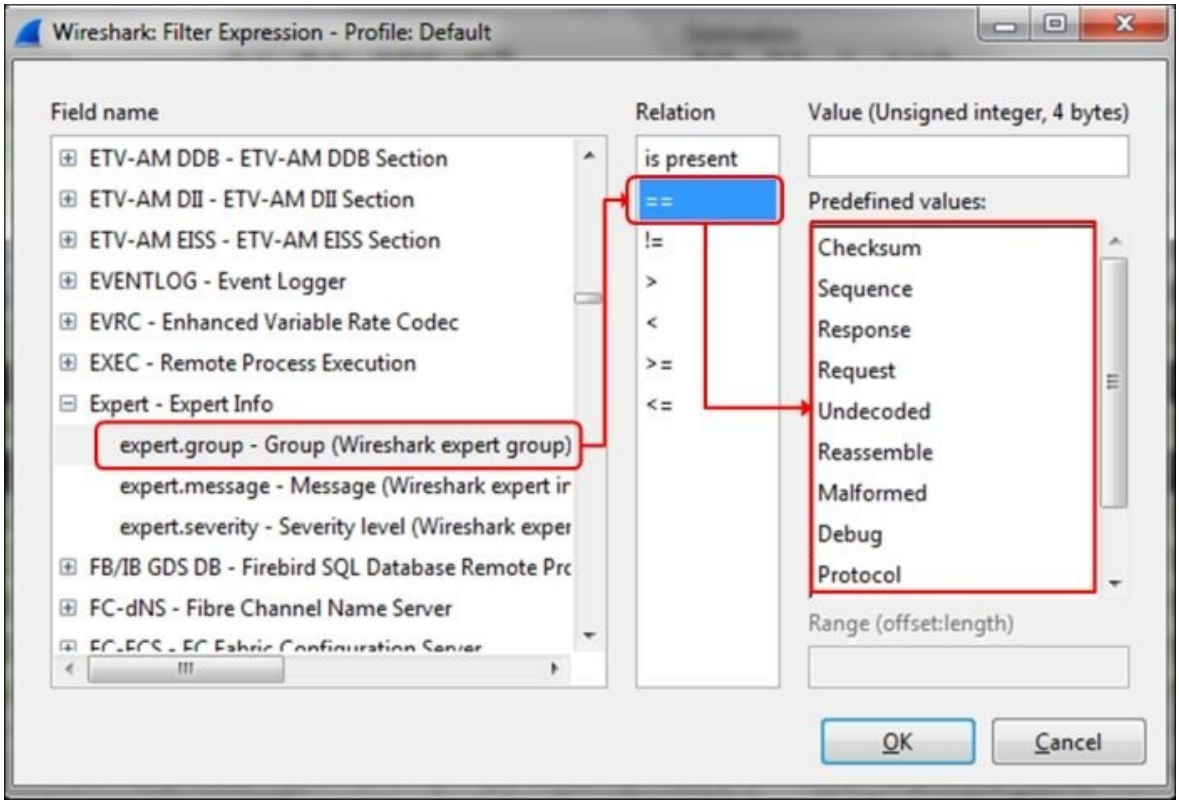
Expert Info severities can also be filtered and presented in the packet pane by displaying filters. To view events according to display filters, perform the following steps:

1. Click on **Expression...** on the right-hand side of the display filter window.
2. Scroll down to get the **Expert** messages (you can just type the word `expert` and you will get there).

   As illustrated in the following screenshot, you will get the following filters: `expert.message`, `expert.group`, and `expert.severity`:



   - `expert.group` refers to expert message groups. This filter categorizes problems according to their types, for example, checksum problems, TCP sequence-related problems, and so on. Have a look at the following screenshot and you will see a list of these issues:



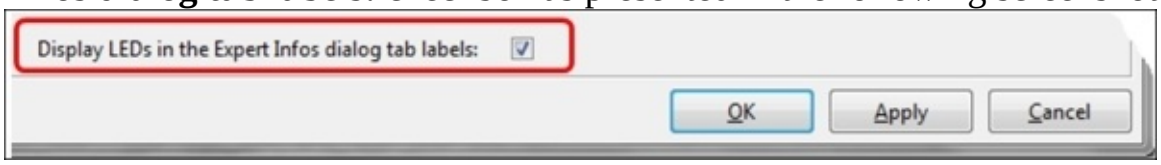
The main categories in `expert.group` are as follows:

- **Checksum**: This indicates an invalid checksum.
- **Sequence**: This indicates TCP sequence-related problems.
- **Response**: This indicates application response code problems (4xx response code files).
- **Request**: This indicates application requests.
- **Undecoded**: This indicates data that cannot be decoded by dissector.
- **Reassemble**: This indicates problems while reassembling (usually when a fragment is missing).
- **Malformed**: This indicates a malformed packet or dissector problem, and the dissection of this packet is aborted.
- **Debug**: This indicates debugging (should not occur in released versions).
- **Protocol**: This indicates the violation of protocol specification (for example, missing field, wrong length, and so on), dissection of this packet will probably be continued.
- **Comment**: This indicates packets with a comment added to them (comments can be added to a packet by right-clicking on it and choosing the **Packet comment …** option.

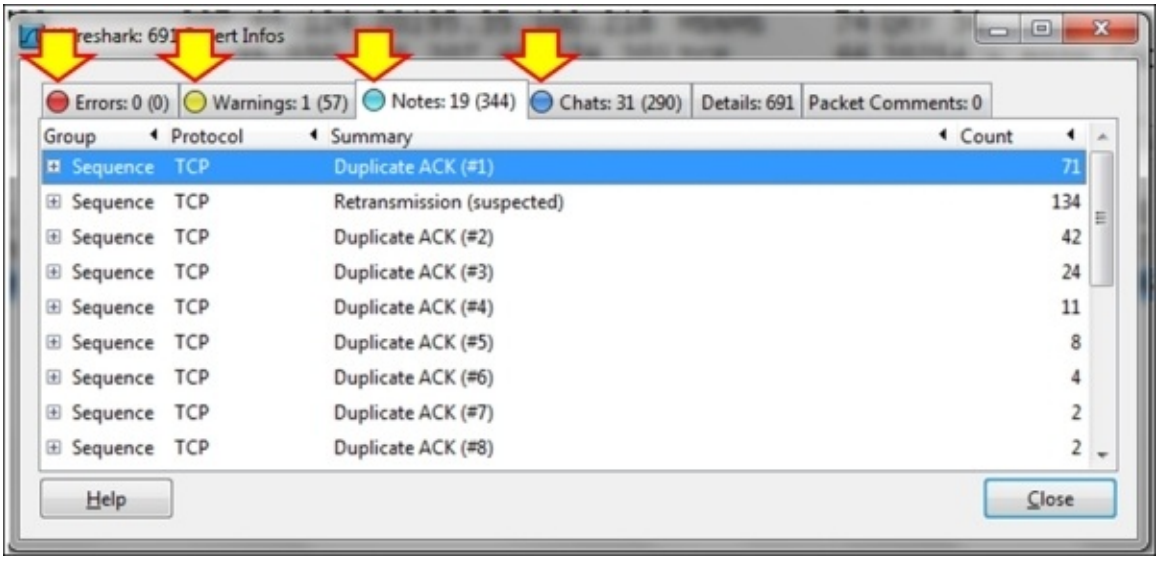
- `expert.message` refers to specific messages. Here, for example, you can configure a filter that displays a message that contains or matches a specific string.
- `expert.severity` refers to messages with specific severities, that is error, warning, note and so on.

You can also choose to show events severities on the **Expert Info** window.

1. Navigate to **Edit** | **Preferences...**.
2. Choose **User Interface**.
3. In the lower half of the right pane, mark the **Display LEDs in the Expert Infos dialog tab labels:** checkbox as presented in the following screenshot:



- Click on **OK**.
- Open the **Expert Infos** window and the severity LEDs will appear on each bar.



The severity level LED will also appear on the lower left corner of the Wireshark main window.

See also

- Chapter 9, *UDP/TCP Analysis*

Error events and understanding them

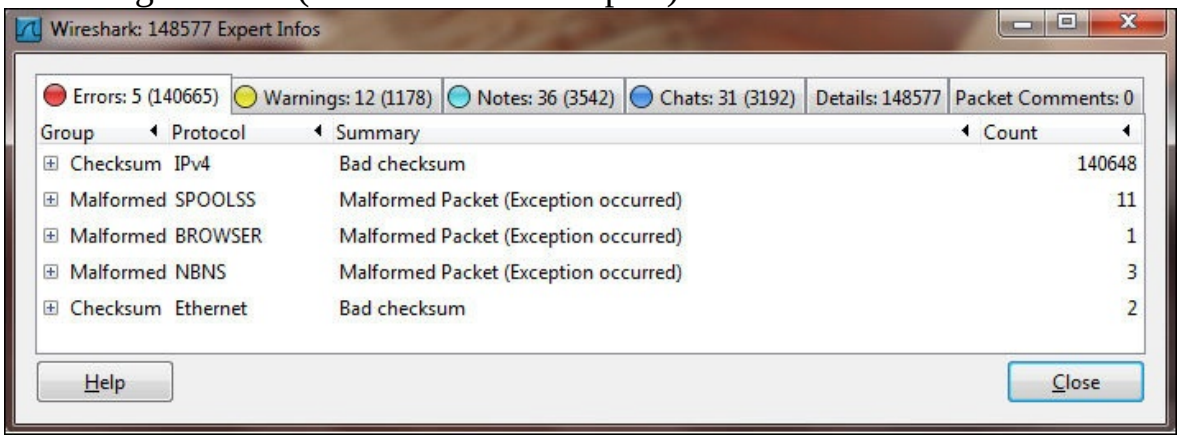
In this recipe, we will get into error and event types, checksum errors, malformed packets, and other types of errors, and what we can understand from them.

Getting ready

Start capturing or open an existing file, and then start the **Expert Infos** window.

How to do it...

1. From the **Analyze** menu, open **Expert Infos** by clicking on **Expert Info**.
2. Click on the **Errors:** bar (should be opened as default). You will get the following window (all events are examples):



In the preceding window, you can see the following two types of errors:

- **Checksum errors**: These can be in Ethernet, IP, or other protocols. In this case, it can be because of real errors or offload.
- **Malformed packets**: These are usually in the application protocols. In this case also, it can be due to a real problem or a dissector error.

How it works...

Checksum is an error-checking mechanism that uses a byte or a sequence of bytes inserted in the packet in order to implement a frame verification algorithm. The principle of error-checking algorithms is to calculate a formula over the entire message (layer 4), packet (layer 3) or frame (layer 2), insert the result in bytes inside the packet, and when the packet arrives at the destination, it calculates the formula again. If we get the same result, it is a good packet; if not, there is an error. The error-checking mechanism can be calculated over the entire packet or only over the header, depending on the protocol.

Offload mechanisms are mechanisms on which the IP, TCP, and UDP checksums are calculated on the NIC just before they're transmitted to the wire. In Wireshark, these show up as corrupt packets because Wireshark captures packets before they are sent to the network adapter; therefore, it will not see the correct checksum because it has not been calculated yet.

For this reason, even though it might look like severe errors, in many cases checksum errors are actually Wireshark errors of misconfiguration. In cases where you see many checksum errors on packets that are sent from your PC, it is probably because of offload.

To cancel the checksum validation, you can do either of the following depending on your protocol:

- For IPv4, when you see many checksum errors and you are sure they are because of the offload, navigate to **Edit** | **Preferences...**. Further, navigate to **Protocols** | **IPv4** and uncheck the **Validate the IPv4 checksum if possible:** checkbox.
- For TCP, when you see many checksum errors and you are sure they are because of the offload, navigate to **Edit** | **Preferences...**. Further, navigate to **Protocols** | **TCP** and uncheck the **Validate the TCP checksum if possible:** checkbox.

There's more...

Malformed packets can be Wireshark bugs or real malformed packets. Use other tools for isolating the problem. Suspected bugs can be reported on the Wireshark website.

Tip

When you see a large amount of malformed packets of checksum errors, it is probably because of offload or dissector errors. Networks with more than 1-2 percent errors of any kind will cause many other events (retransmissions for example) and will become much slower than expected, and therefore, you cannot have a high error rate with a functioning network!

See also

- Chapter 9, *UDP/TCP Analysis*

Warning events and understanding them

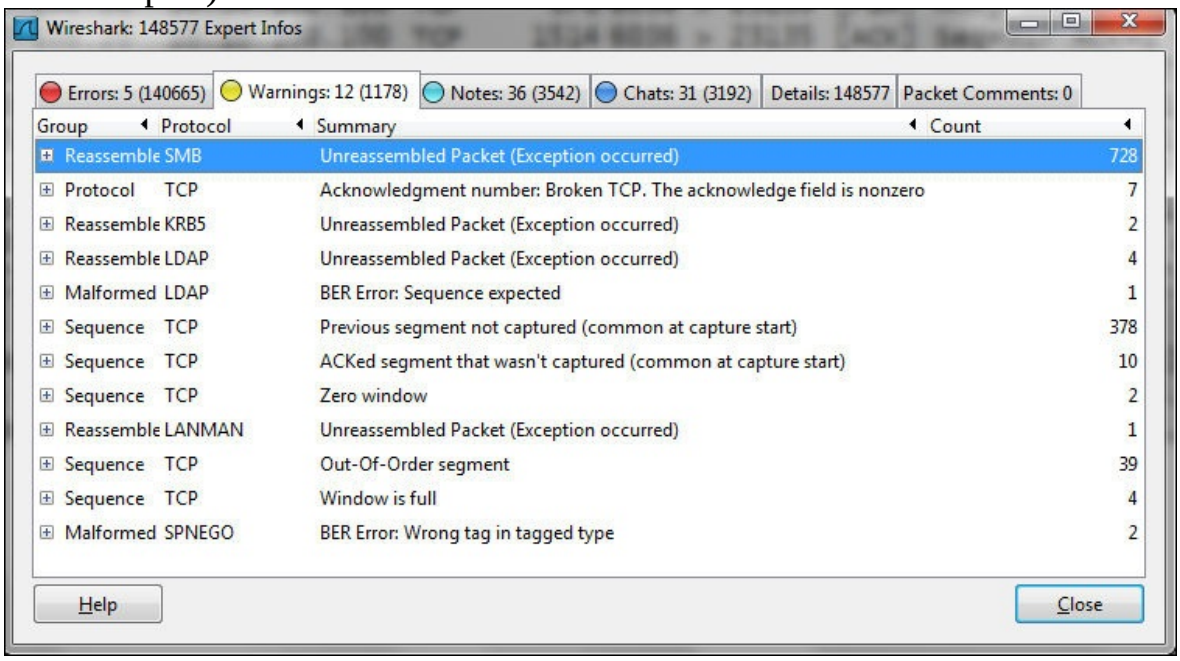
As described earlier, warning events indicate problems in the application or in communication. In this recipe, we will describe the main events in this category.

Getting ready

Start capturing or open an existing file, and start the **Expert Infos** window.

How to do it...

1. From the **Analyze** menu, open **Expert Infos** by clicking on **Expert Info**.
2. Click on the **Warnings:** bar. You will get the following window (all events are examples):



You will see here several event categories:

- **Reassembly problems**: These are mostly un-reassembled packets. These are usually indicated as Wireshark dissector problems.
- **TCP window problems**: These are mostly zero window and window full problems. These usually indicate slow-end devices (servers, PCs, and so on).
- **Segment loss, segments not in order**: These indicate previous segment losses and the ACKed segment that wasn't captured. These are usually TCP problems that are caused by network problems.

How it works...

Wireshark watches the parameters of the monitored packets as follows:

- It watches TCP window sizes and checks if the window size reduced to zero
- It looks for TCP packets (segments) that are out of order, that is, if they were sent before or after the expected time
- It looks for ACKs for TCP packets that were not sent

These parameters, along with many others, provide you with a good starting point to look for network problems. We will go into the details of it in Chapter 9, *UDP/TCP Analysis*.

There's more...

Don't forget that warning events are those that Wireshark refers to as important, but it is not necessarily so. If, for example, you have previous segment not captured, they will be under warnings, but it can be due to capture problems.

See also

- Chapter 9, *UDP/TCP Analysis*

Notes events and understanding them

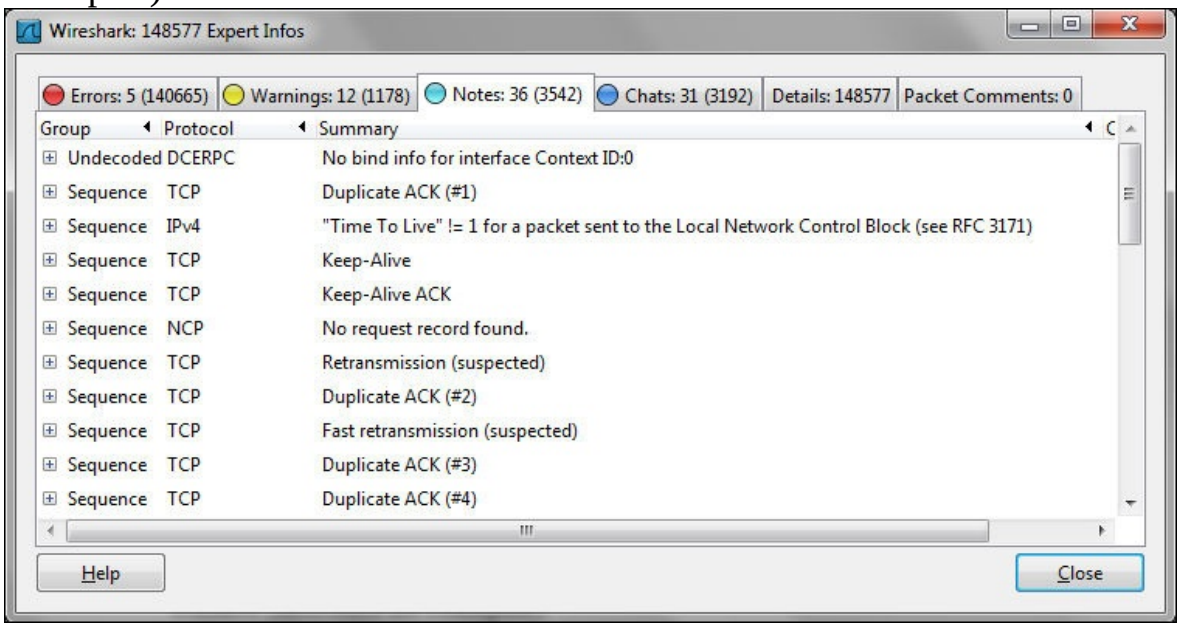
As described earlier, when Wireshark indicates that an event may cause a problem but is still inside the normal behavior of the protocol, it will be under the **Notes** bar. TCP retransmission, for example, will be displayed under the **Notes** bar because even though it is a critical problem that slows down the network, it is still under the normal behavior of TCP.

Getting ready

Start capturing or open an existing file and start the **Expert Infos** window.

How to do it...

1. From the **Analyze** menu, open **Expert Infos** by clicking on **Expert Info**.
2. Click on the **Notes** bar. You will get the following window (all events are examples):



You will see here several event categories:

- Retransmissions, duplicate ACKs, fast retransmissions that usually indicate slow network, packet loss, or very slow end devices or applications
- Keep-alives that indicate TCP or application problems
- Time to live and routing events that in most cases indicate routing problems

Tip

Additional events will be discussed in Chapter 9, *UDP/TCP Analysis*, Chapter 10, *HTTP and DNS*, Chapter 11, *Analyzing Enterprise Applications', Behavior*, and Chapter 12, *SIP, Multimedia, and IP Telephony*.

How it works...

Wireshark watches the parameters of the monitored packets. It watches TCP sequences and acknowledges numbers while checking for retransmissions and other sequencing problems. It looks for IP **Time To Live** (**TTL**) with value of 1 coming from a remote network, and tells you it is a problem. It looks for keep-alives that may be in a normal condition but can also indicate a problem.

These parameters, along with many others, provide you with a good starting point to look for network performance problems.

There's more...

Many symptoms that are seen here can be an indication of several types of problems. For example, a packet can be retransmitted because of an error that caused the packet to be lost, because of bad network conditions (low bandwidth or high delay) that caused the packet not to arrive on time, and it can be also because of a nonresponsive server or client. The Expert Info system will give you the symptom. We will learn later in this book how to solve this problem.

See also

- You can read more on TCP performance issues in Chapter 9, *UDP/TCP Analysis*. It includes TCP retransmissions, fast retransmissions and why they happen, what are ACKs and duplicate ACKs, zero window, window changes and other TCP sliding windows issues, and more.

Chapter 7. Ethernet, LAN Switching, and Wireless LAN

In this chapter we will cover the following topics:

- Discovering broadcast and error storms
- Analyzing Spanning Tree Protocols
- Analyzing VLANs and VLAN tagging issues
- Analyzing wireless (Wi-Fi) problems

Introduction

In this chapter, we will focus on how to find and resolve layer-2-based problems with the focus on Ethernet-based issues such as broadcast events and errors and how to find out where they are coming from. We will also focus on LAN protocols such as Spanning Tree, VLANs, and Wireless LAN.

These issues have to be resolved before we go up to layers 3, 4, and the Application layers, since layer 2 problems will be reflected in the upper layer protocols. For example, packet losses in layer 2 will cause retransmissions in TCP, which is a layer 4 protocol, and these can cause slow application response time in the upper layers.

Discovering broadcast and error storms

One of the most troublesome problems in communication networks is the **broadcast** and **error storms**. These problems can happen because of layer 2 loops, layer-2-based attacks, a problematic network adapter, or a service that sends packets to the network.

In this chapter we will provide some basic recipes on how to find, isolate, and solve these types of problems.

Tip

A broadcast storm is when you get thousands and even tens of thousands of broadcasts per second. In most cases it would lock out the network completely.

Getting ready

In these types of problems, you will usually be called on to solve the *network is very slow* or *network has stopped working* problems.

Several important facts to remember are:

- Broadcasts are not forwarded by routers.
- Broadcasts are not forwarded between VLANs (this is why VLANs are called **broadcast domains**), so every VLAN is a single broadcast domain.
- Error packets are not forwarded by LAN switches (at least not through the good ones).
- Multicasts are forwarded through switches, unless configured otherwise.
- Multicasts are forwarded through routers only if the routers are configured to do so.
- A reasonable number of broadcasts are transmitted in every network. This is how networks work. Too many broadcasts could be a problem.

Tip

There is a difference between too many broadcasts and a broadcast storm. Too many broadcasts (for example, a few hundred per second) can load the network but still, in most cases, users will not notice it. Broadcast storms will lock out the network completely.

How to do it...

To find out where the problem comes from, go through the following steps:

1. Since "slow network" is a problem sensed by users, start with asking the following questions:
    - Is this problem in the HQ?
    - In a single branch?
    - All over the network or a specific VLAN?

    Don't ask the users about VLANs, of course; users are not networking experts. Ask them about applications running on their group, on their department, and so on.

    Tip

    In an organization network, VLAN will usually be configured per department (or several departments) and per geographical area (or several areas) or even per organization functionality; for example, HR VLAN, finance VLAN, users of a specific software VLAN, and so on. By asking if the problem is as per one of these characters, you will be able to narrow the area in which you need to look for the problem.
2. The next question should be a trivial one: "Is the network still working?" In a broadcast storm, the network will become very slow; in most cases, to the point that applications will stop functioning. In this case, you have the following typical problems:
    - Spanning Tree Problems
    - A device that generates broadcasts
    - Routing loops (will be discussed in Chapter 8, *ARP and IP Analysis*)

    Tip

    The question I'm always asked is: "How many broadcasts are too many?" Well, there are, of course, several answers for this. It depends on what the network devices are doing and the protocols that are running on them.

    A reasonable number of broadcasts should be from 1 to 2 up to 4 to 5

per device per minute. For example, if your network is built from 100 devices on a single VLAN, you should expect no more then 5-10 broadcasts per second (5 broadcasts x 100 devices gives 500 broadcasts per minute, that is, around 9-10 per second). More than these is also reasonable, as long as they are not coming in thousands and you know what they are.

Spanning Tree Problems

In Spanning Tree Problem, you will get thousands and even tens of thousands of broadcasts per second (refer to the *How it works...* section in this recipe to know why). In this case, your Wireshark, and probably your laptop, will freeze. Close Wireshark, disconnect cables to isolate the problem, and check the STP configuration in the switches.

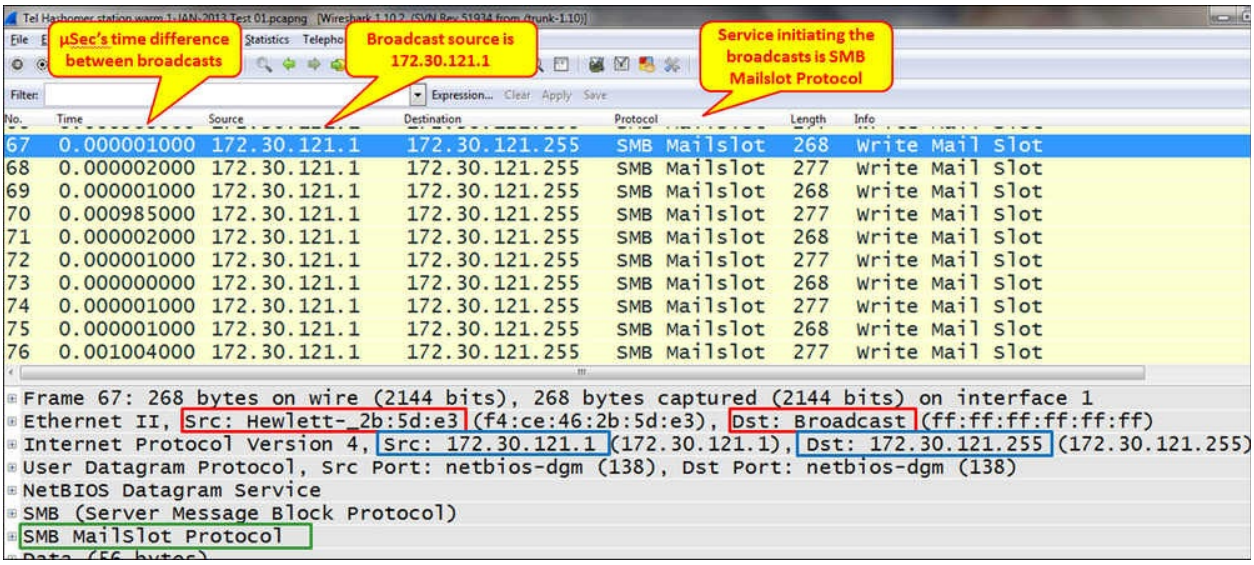
A device that generates Broadcasts

A typical broadcast storm generated from a specific device will have the following characteristics:

- Significant number of broadcasts per second (thousands and more)
- In most cases, the broadcasts would be from a single source; but in case of attacks, they can be from multiple sources
- Usually in constant packet/second rate, that is, with intervals between frames that are nearly equal

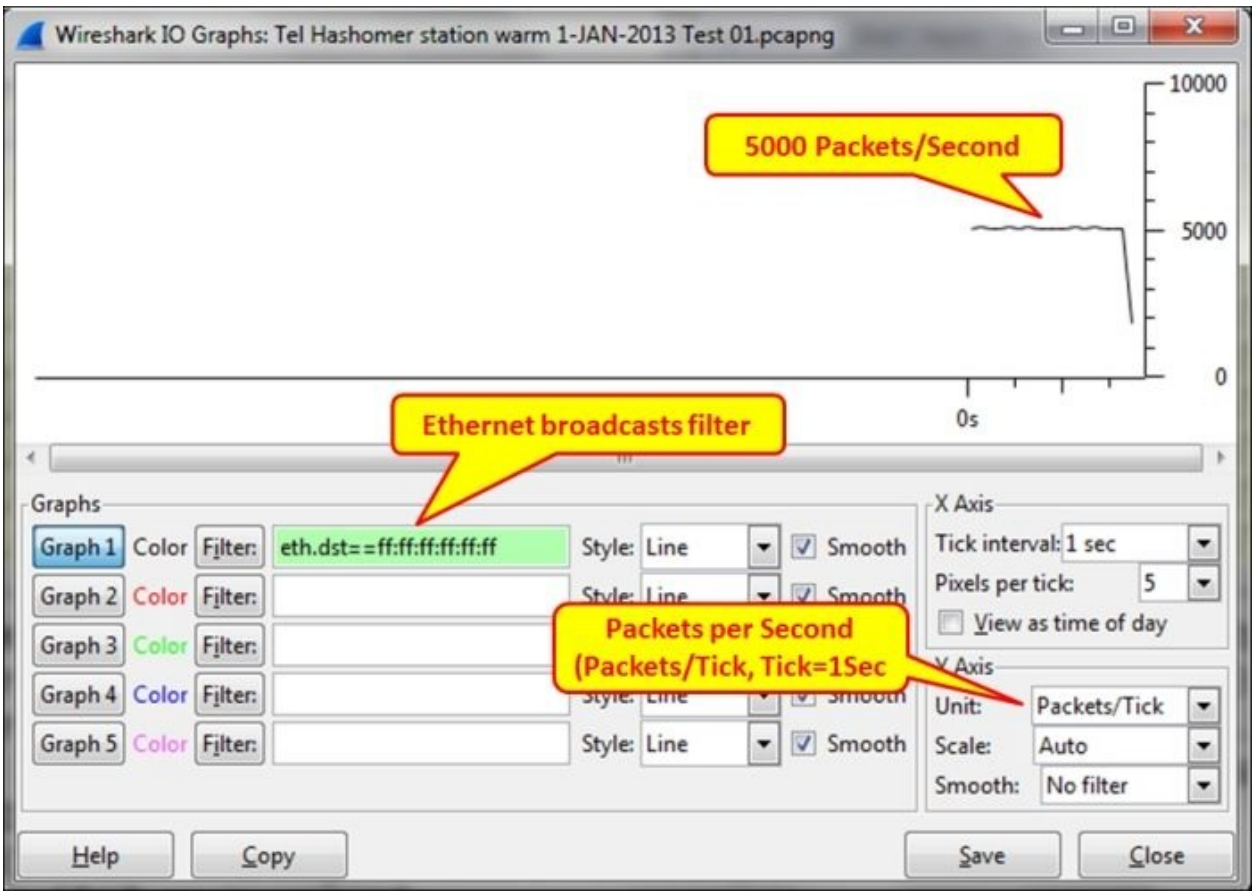
We can see how we find a broadcast storm according to the parameters mentioned in the preceding list in the next three screenshots.

In the following screenshot we see a large number of broadcast packets sent from the source MAC (HP network adapter) to ff:ff:ff:ff:ff:ff:

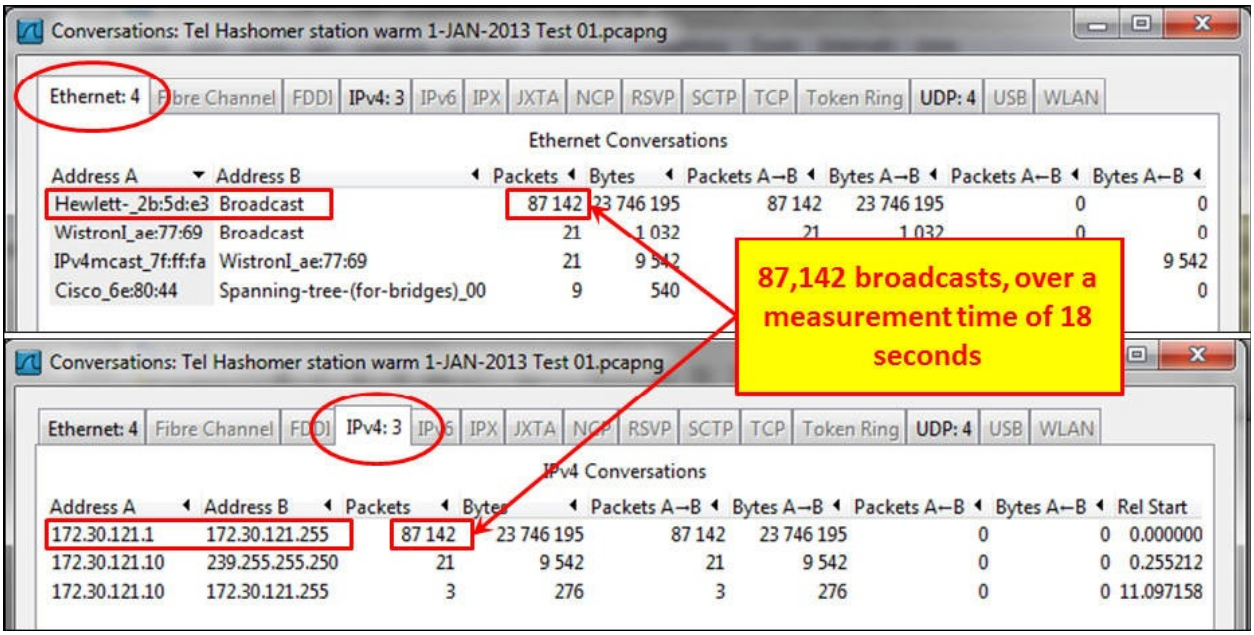


In the preceding screenshot we just saw that the time column is configured in seconds since the previous displayed packet. You can configure it by navigating to **View** | **Time Display Format**.

The following screenshot shows the traffic on an IO Graph; we see that the total number of packets/second is 5,000:



In the following screenshot we see what we will get in the **Conversations** window. Here, we will also get an enormous number of broadcasts that can be viewed in the Ethernet and the IPv4 statistics (I've captured data for 18 seconds).



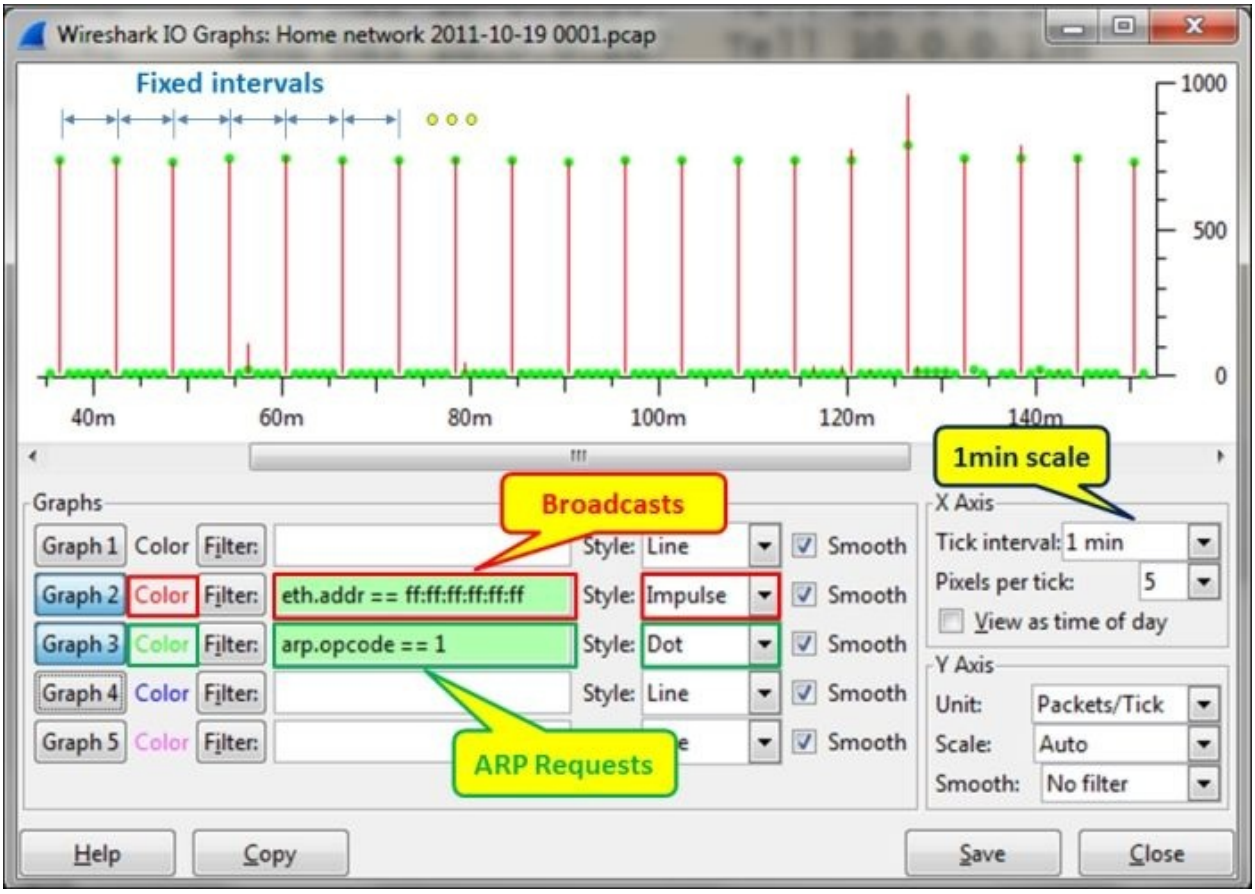
In the preceding case, the problem was a service called **SMB Mailslot Protocol**. Simple trial and error to find what this service is, and disabling it on the station that caused it solved the problem.

Tip

It is important to note that when you disable a service (especially the one that belongs to the operating system), make sure that the system keeps functioning and stays stable over time. Don't leave the site before you have verified it!

Fixed pattern broadcasts

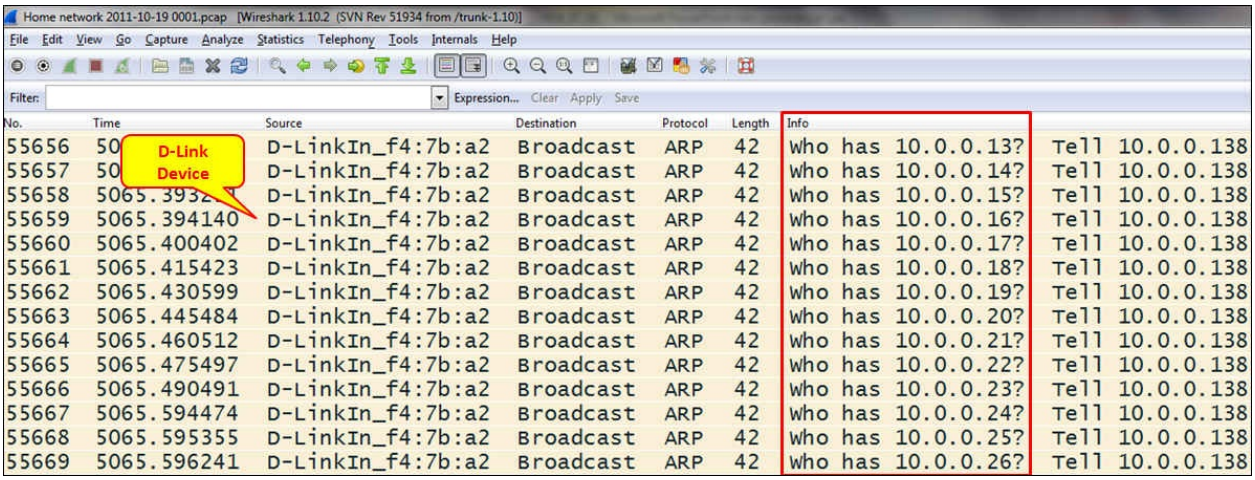
You can also have broadcasts in fixed patterns, for example, every fixed amount of time, as shown in the following screenshot:



The graph is configured for **Tick interval:** (under **X Axis**) of **1 min**, and for the following filters:

- The red filter for all broadcasts in the network (`eth.addr == ff:ff:ff:ff:ff:ff`)
- The green filter for broadcasts that are ARP requests (`arp.opcode ==1`)

What we see here is that around every five minutes, there is a burst of ARP requests (the green dots). If we click on one of the dots in the graph, it will take us to the packet in the capture pane. In the following screenshot, we see the scan pattern that happens every five minutes:



In the preceding screenshot, we can see that it is the D-Link router that scans the internal network. This can be good or bad, and we will get to the details later in Chapter 14, *Understanding Network Security*. In any case, it's good to check what is running in our network.

How it works...

Broadcasts in IPv4 networks are quite common, and these layer 3 broadcasts will be sent over layer 2 broadcasts. Every time a layer 3 device sends a broadcast to the network (an IP address that ends with all 1s, refer to Chapter 8, *ARP and IP Analysis*), it will be converted to layer 2's all fs destination address.

There are several families of broadcasts that you will see in IP-based networks. Some of them are as follows:

- TCP/IP-based network protocols such as ARP requests, DHCP requests, and others
- Network protocols such as **NetBIOS Name Service** (**NBNS**) queries, **NetBIOS Server Message Block** (**SMB**) announcements, **Network Time Protocol** (**NTP**), and others
- Applications that send broadcasts such as Dropbox, Microsoft Network Load Balancing, and others

In IPv6, we don't have broadcasts, but we have unicasts, multicasts, and anycasts. Since the protocol works with multicasts for discovery mechanisms, announcements, and other mechanisms, we will see a lot of them.

There's more…

A problem I come across in many cases is how to use the broadcast and multicast storm control definitions in LAN switches (the `storm-control broadcast level [high level] [lower level]` command in Cisco devices).

The problem is that in many cases, I see configurations that limit the number of broadcasts to 50, 100, or 200 broadcasts per second, and this is not enough. In a network, you could happen to install a software that sends broadcasts or multicasts to the network that crosses these values. Then, according to what you have configured in the switch, it will start sending traps to the management system or even disconnecting ports (the `storm-control action {shutdown | trap}` command in Cisco devices), depending on what you have configured.

The solution for this is simply to configure high levels of broadcasts as the threshold. When a broadcast storm happens, you will get thousands of broadcasts; so configuring a threshold level of 1,000 to 2,000 broadcasts or multicasts per second provides you with the same protection level, without any disturbances to the regular network operation.

See also

- For more information about IPv4 refer to Chapter 8, *ARP and IP Analysis*

Analyzing Spanning Tree Protocols

All of us have worked with, or at least heard about, **STP** (**Spanning Tree Protocol**). The reason I call this recipe *Analyzing Spanning Tree Protocols* is because there are three major versions of it as follows:

- **Spanning Tree Protocol (STP)**: This is an IEEE 802.1D standard from 1998 called 802.1D-1998
- **Rapid Spanning Tree Protocol (RSTP)**: This is an IEEE 802.1W standard from 2001, later added to 802.1D, called 802.1D-2004
- **Multiple Spanning Tree (MST)**: This was originally defined in IEEE 802.1S and later merged into IEEE 802.1Q

There are also some proprietary versions from Cisco and other vendors. In this recipe we will focus on the standard versions, and learn how to troubleshoot problems that might occur during STP/RSTP/MST operations.

Getting ready

The best way to find out STP problems is to log in to the LAN switches and use the vendor's commands (for example, Cisco IOS or Juniper JUNOS CLI) to find and fix the problem. If you have properly configured SNMP on your network device, you will get all the messages on the management console.

The purpose of this recipe is to see how to use Wireshark for this purpose, even though we still recommend to use it as a second line tool for this purpose.

So just open your laptop, start Wireshark, and start capturing data on the LAN.

How to do it...

There are several things to notice in a network regarding STP:

- Which STP version is running on the network?
- Are there any topology changes?

Which STP version is running on the network?

Wireshark will provide you with the version of the STP type (STP, RSTP, or MST) running on the network by looking at the **Bridge Protocol Data Units** (**BPDUs**). BPDUs are the update frames that are multicast between switches.

The protocol versions are:

- For STP, protocol version ID equals 0
- For RSTP/MST, the protocol version ID equals 3

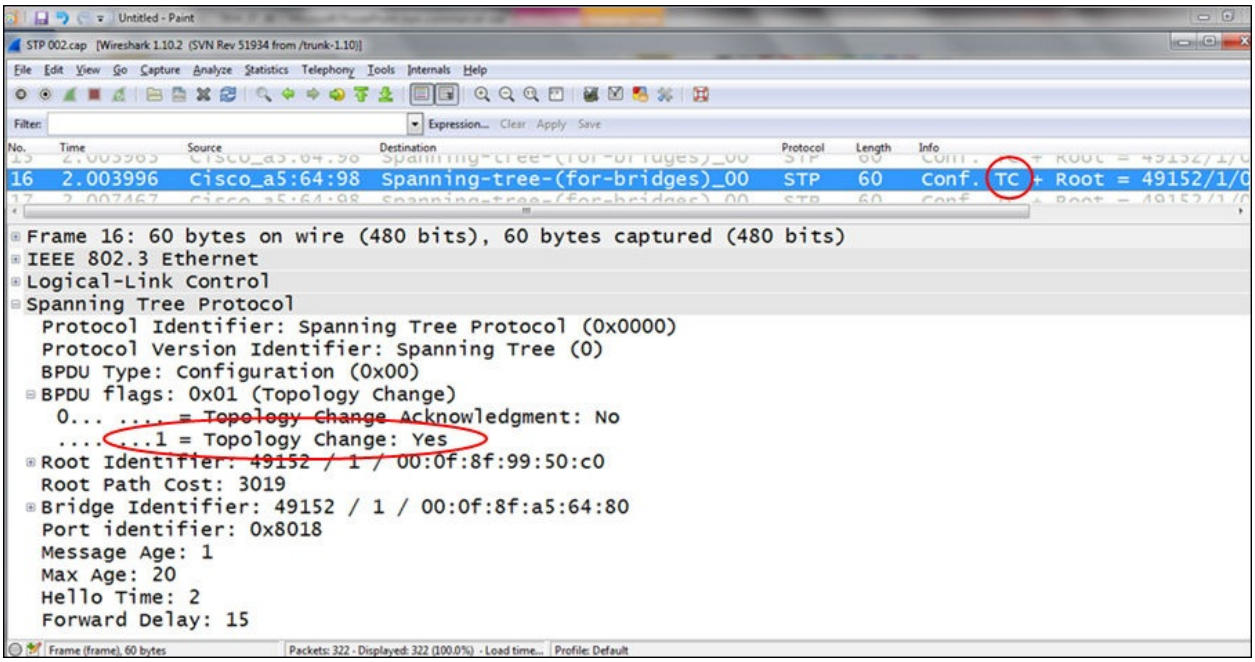
Tip

In the standards you will not find the word "switch"; it will always be "bridge" or "multiport bridge". In this book, we will use the terms bridge and switch alternatively.

Are there too many topology changes?

When you monitor STP operations, you may be concerned when you see many topology changes. Topology changes are normal in STP, but too many of them can have an impact on network performances.

A topology change happens when a new device is connected to the network. You can see a topology change in the following screenshot:



When you see too many topology changes, configure the LAN switch ports that are connected to hosts, which do not support STP, (typically, end stations that users frequently power on and off) with the portfast feature (applied for Cisco switches; for other vendors, check the vendor's manual).

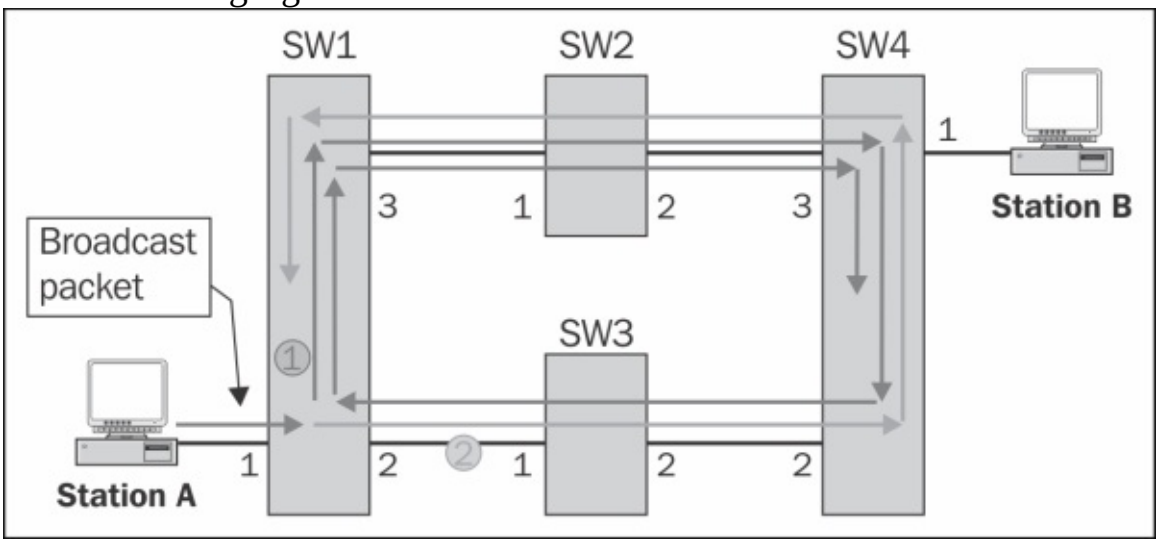
Tip

In the old STP (IEEE 802.1d), after connecting a device to a switch port, it takes the switch around a minute to start and forward packets. This can be a problem when a client tries to log in to the network servers during this period of time. The portfast feature forces the port to start forwarding within a few seconds (usually 8 to 10), in order to prevent these kinds of problems.

If topology changes continue, check what can be the problem and who is causing it.

How it works...

Spanning Tree Protocol prevents a loop in the local area networks. A loop can happen if you connect two or more switches with multiple connections as shown in the following figure:



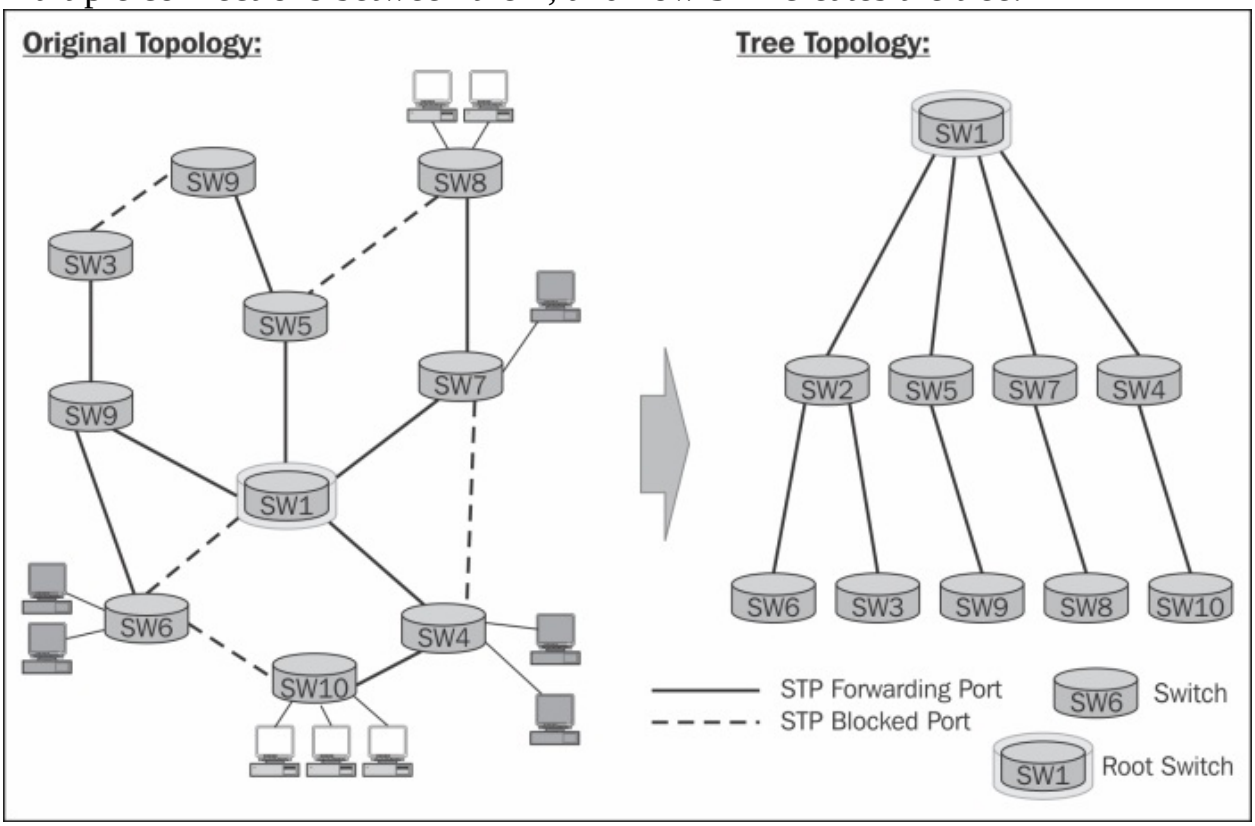
Let's see how a loop is created:

1. **Station A** sends a broadcast to the network. A broadcast can be an ARP, NetBIOS, or any other packet with all ffs in the destination MAC address.
2. Since broadcasts are forwarded to all ports of the switch, **SW 1** receives the broadcast from port **1** and forwards it to ports **2** and **3**.
3. **SW 1** and **SW 3** will forward the packets to their other ports, which will get them to ports **2** and **3** of **SW 4**.
4. **SW 4** will forward the packet from port **2** to port **3**, and the packet coming from port **3** to port **2**.
5. We will get two packets circling endlessly: the one that has been forwarded to port **3** (the red arrows), and the one that has been forwarded to port **2** (the green arrows) of **SW 1**.
6. Depending on the switch forwarding speed, we will get up to tens of thousands of packets that will block the network completely.

The Spanning Tree Protocol prevents this from happening by simply building a

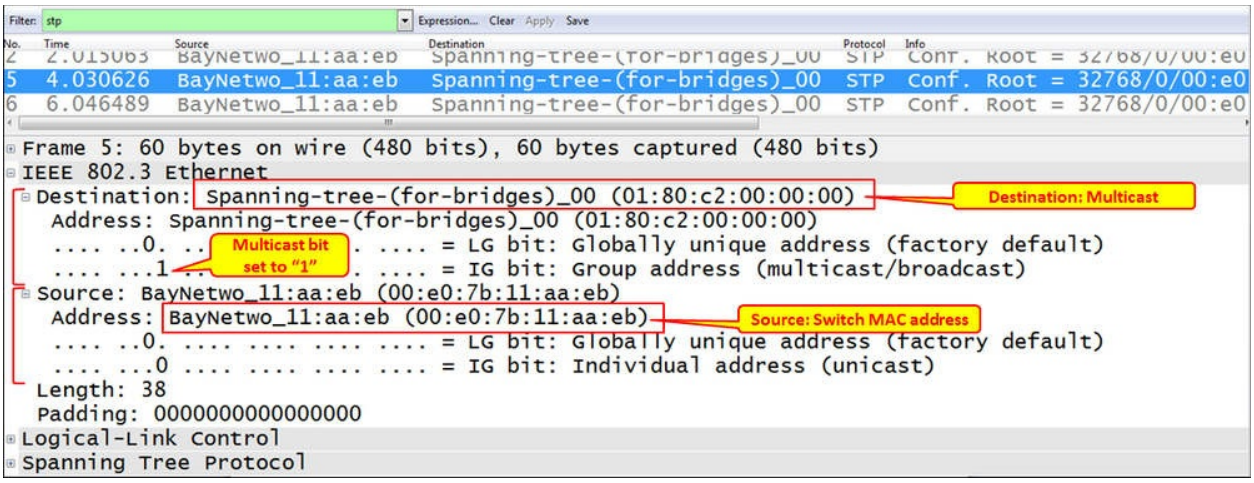
tree topology, that is, by defining a loop-free topology. Links are disconnected and brought back to service in the case of a failure.

In the following figure, we see how we initially connect all switches with multiple connections between them, and how STP creates the tree:

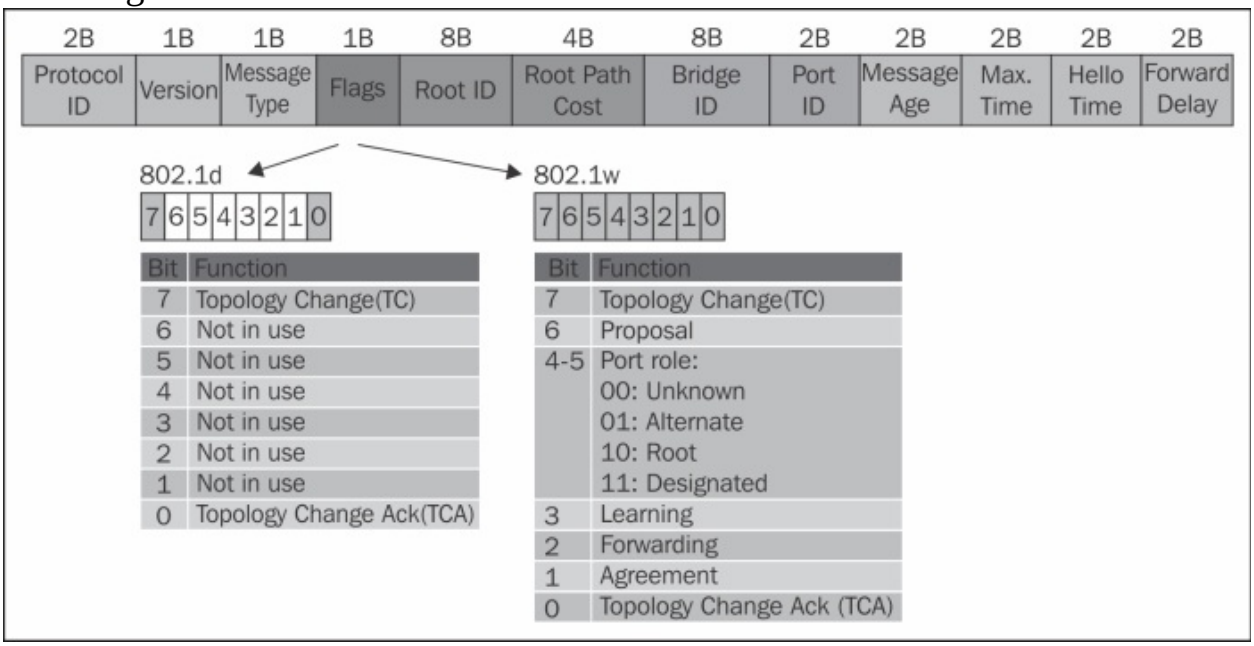


BPDUs are update frames that are sent by multicast between the LAN switches.

First, on the Ethernet level as we see in the following screenshot, the packet will be multicast from the source MAC of the switch sending the update:



The BPDU is carried by Ethernet 802.3 frame has the format as shown in the next diagram:



In the following table, you can see the fields in the STP frame:

Field	Bytes	What is it	Values	Display filter
Protocol	2	The protocol identifier.	Always 0	`stp.protocol`

ID				
Version	1	The protocol version.	For STP = 0 For RSTP = 2 For MST = 3	`stp.version`
Message Type	1	The BPDU type.	For STP = 0 For RSTP = 2 For MST = 2	`stp.type`
Flags	1	The protocol flags.	In the previous illustration.	`stp.flags`

Root ID	8	The root identifier (Root ID), that is, the bridge priority concatenated with the bridge hardware address (MAC).	The MAC address of the root bridge.	`stp.root.prio`
				`stp.root.ext`
				`stp.root.hw`

Root Path Cost	4	The path cost to the root.	Path cost as calculated by Spanning Tree. In case this is the root, path cost will be zero.	`stp.root.cost`

Bridge ID	8	The bridge identifier (Bridge ID), that is, the bridge priority concatenated with the bridge hardware address (MAC).	The bridge MAC address.	`stp.bridge.prio`
				`stp.bridge.ext`
				`stp.bridge.hw`

Port ID	2	The port identifier.	The identifier of the port from which the update was sent.	`stp.port`
Message Age	2	The **Message Age** field indicates the amount of time that has elapsed since a bridge sent the configuration message on which the current configuration message is based.	For every BPDU, the bridge that sends the frame sends a value of 0, incremented by 1 for every bridge that forwards it.	`stp.msg_age`
Max. Time	2	The maximum age, which is the maximum time (practically the number	Usually 20	`stp.max_age`

		of bridges) that the frame can stay in the network.		
Hello Time	2	Time between BPDUs.	Usually 2 seconds	`stp.hello`
Forward Delay	2	The **Forward Delay** field indicates the length of time that bridges should wait before transitioning to a new state after a topology change.	Usually 15 seconds	`stp.forward`

Note that in the case of MST, an additional header will be added for the MST parameters.

Port states

In STP, the port states are as follows:

- **Disabled**: In this state no frames are forwarded and no BPDUs are heard
- **Blocking**: In this state no frames are forwarded, but BPDUs are heard
- **Listening**: In this state no frames are forwarded, but the port listens for frames
- **Learning**: In this state no frames are forwarded, but MAC addresses are captured
- **Forwarding**: In this state frames are forwarded, and MAC addresses are captured

The moment you connect a device to the LAN switch, the port goes through these stages and the time it takes is as follows:

- From Blocking to Listening takes 20 seconds
- From Listening to Learning takes 15 seconds
- From Learning to Forwarding takes 15 seconds

In RSTP and MST, the port states are as follows:

- **Discarding**: In this state frames are discarded
- **Learning**: In this frame no frames are forwarded, and MAC addresses are captured
- **Forwarding**: In this state frames are forwarded, and MAC addressesare captured

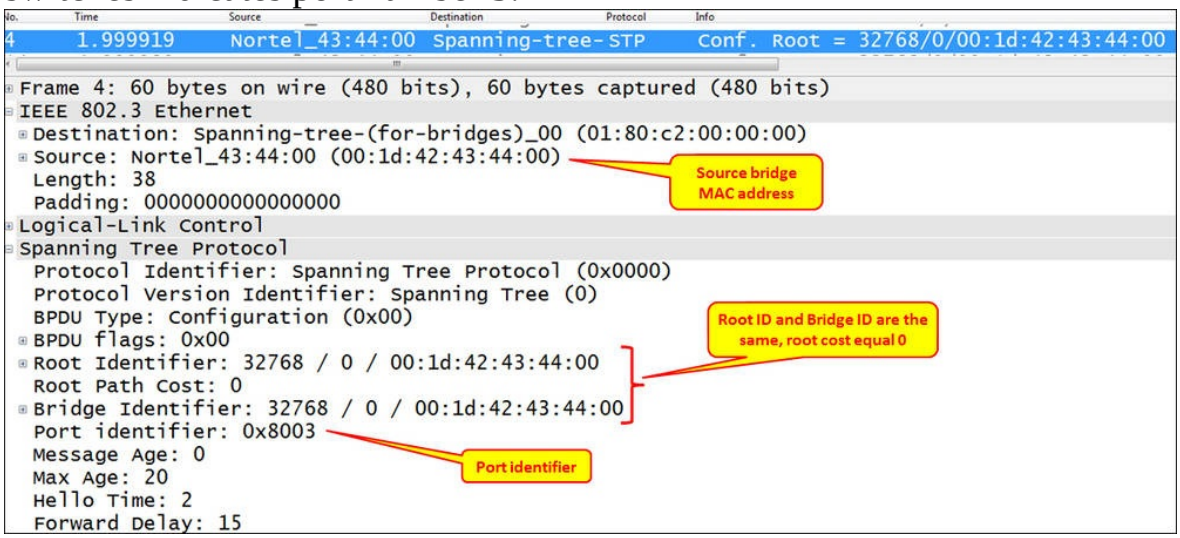
The entire port state transition from Discarding to Forwarding should take a few seconds, depending on the network topology and complexity.

There's more…

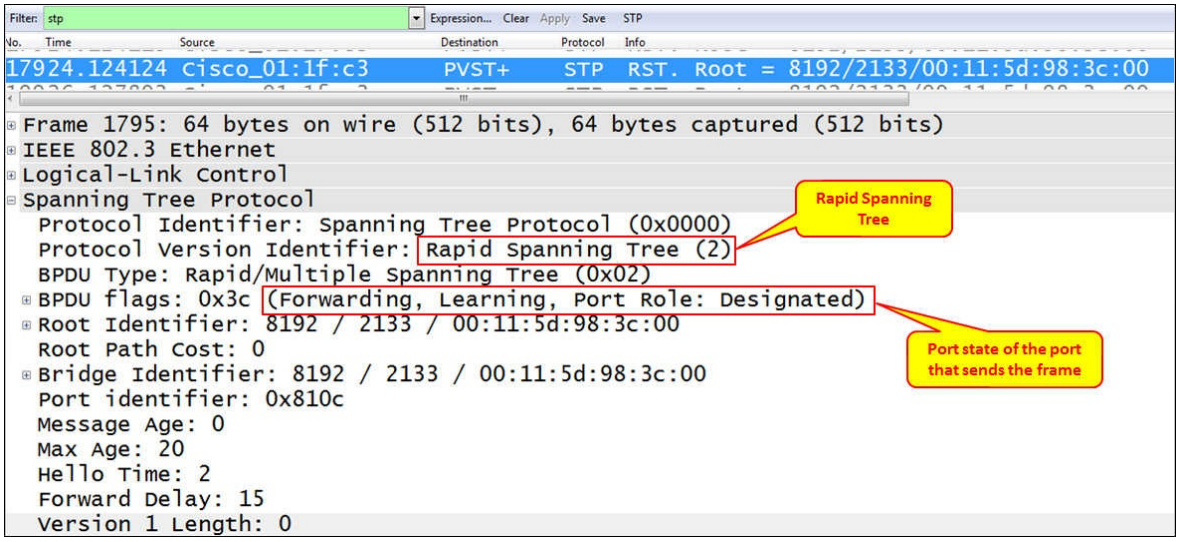
For Spanning Tree debugging, the best thing is to get the data from a direct connection to the LAN switches. A well-configured SNMP trap to a management system can also assist in this task.

Some examples of STP packets are as follows:

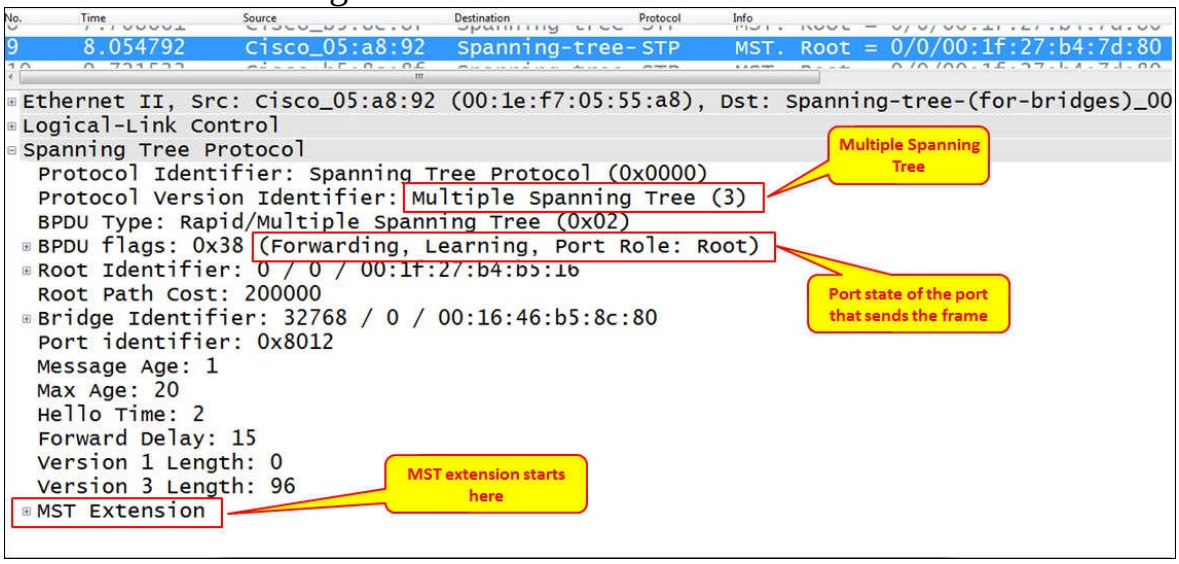
- In the following screenshot you see an STP frame. You can see that the source MAC address is a Nortel address, and in the BPDU itself you see that the root and the bridge identifiers are equal; this is because the bridge that sends the packet is the root. The port ID is 8003, which in Nortel switches indicates port number 3.



- In the following screenshot, you can see a Rapid STP BPDU. You can see here the protocol identifier that equals 2 and the port state that is designated.



- In the previous screenshot, you can see an example for MST. Here we see the MST extension right after the standard STP frame.



Analyzing VLANs and VLAN tagging issues

VLAN, or **Virtual LAN**, is a mechanism that divides a LAN into separate LANs without any connectivity between them, and this is where the name virtual comes from. In this section we will have a look at recipes to monitor VLAN traffic.

The purpose of this recipe is to give the reader a general description of how to use Wireshark for VLAN issues. An easier way to solve related problems is to use the vendor's CLI (Cisco IOS, Juniper JUNOS, and so on) for this purpose.

Getting ready

We will discuss two issues in this recipe:

- How to monitor traffic inside a VLAN
- How to view tagged frames going through a VLAN-tagged port

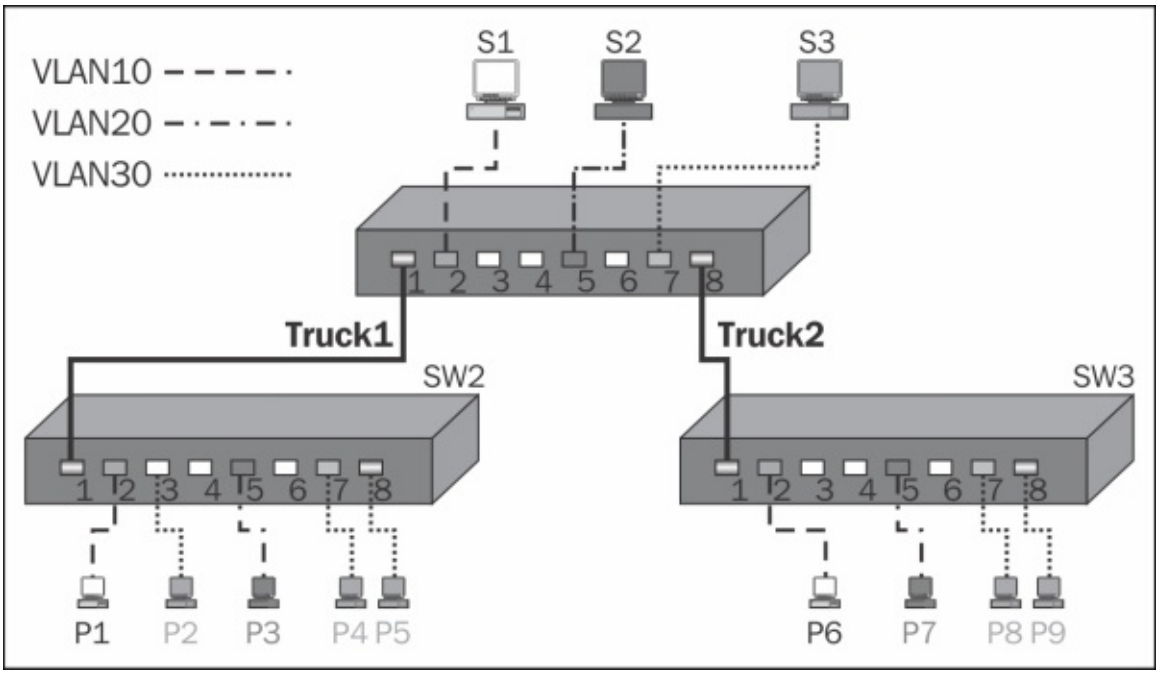
In the first case, a simple configuration is required. In the second case, there are some points to take care of.

While capturing on a VLAN, you won't necessarily see the VLAN tags in packets. The question of whether you will see the VLAN tags actually depends on the operating system you are running, and if your **Network Interface Card** (**NIC**) and the NIC driver supports this feature.

Note

The question of whether your OS and NIC supports VLAN tagging entirely depends on the OS and the NIC vendor. Go to the vendor's manuals or Goggle it to find out.

In the following figure you can see a typical topology with VLANs. The upper switch is connected by two trunks (these are ports that tag the Ethernet frames) to the lower switches. On this network you have VLANs 10, 20, and 30, while PCs connected to each of the VLANs will not be able to see PCs from other VLANs.


VLAN10
VLAN20
VLAN30
S1
S2
S3
1 2 3 4 5 6 7 8
Truck1
Truck2
SW2
SW3
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
P1
P2
P3
P4 P5
P6
P7
P8 P9

How to do it...

Connect Wireshark to the switch you want to monitor. Let's look at the preceding configuration (shown in the preceding figure).

Monitoring traffic inside a VLAN

In order to monitor traffic on an entire VLAN

1. Connect your laptop to the central switch and to one of the ports.
2. Configure the port mirror from the monitored VLAN to the port you are connected to. For example, if you connect your laptop to SW1 port 4, and you want to monitor traffic from VLAN10, the commands will be (in Cisco):

```
Switch(config)#monitor session 1 source vlan 10
Switch(config)#monitor session 1 destination interface
fastethernet 0/4
```

   This will show you traffic from VLAN10 that is forwarded through the central switch, SW1.

Tip

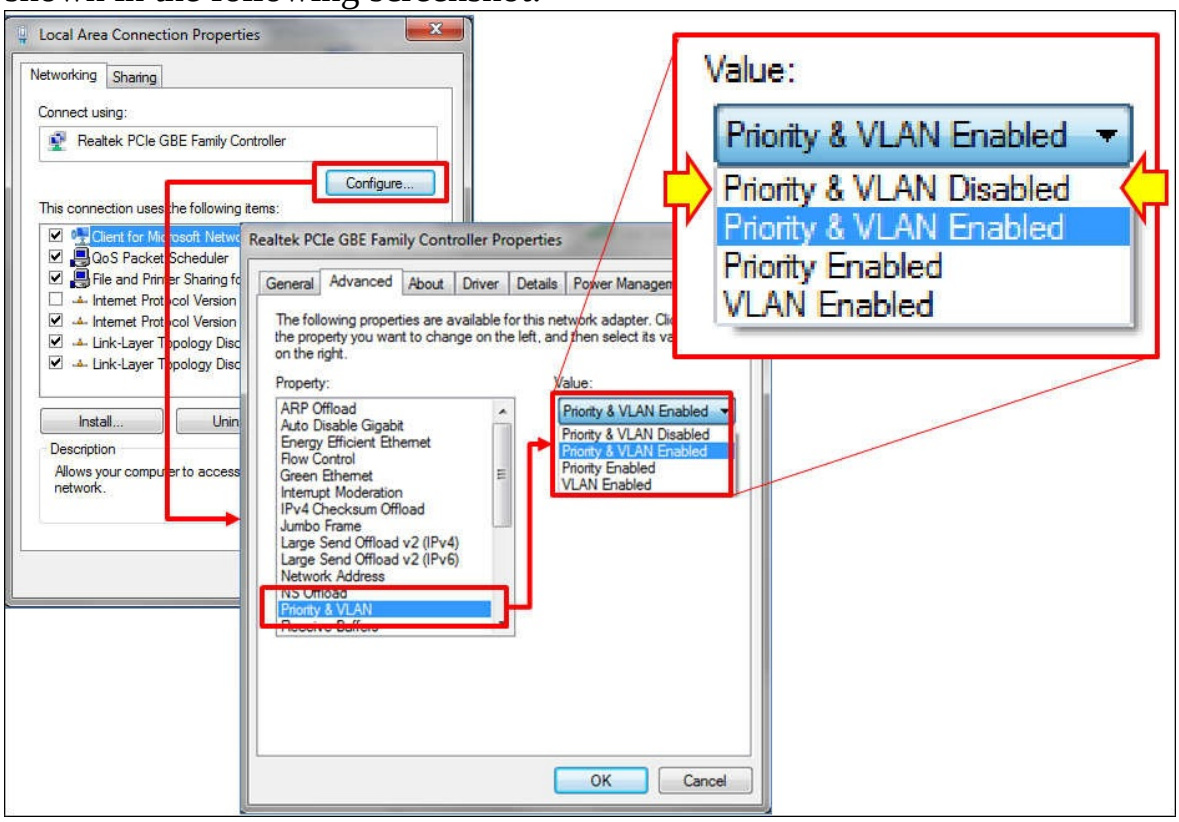
For further information on how to configure port mirroring on various vendor websites, search for `SPAN` (in Cisco), `port mirror`, or `mirroring` (HP, DELL, Juniper, and others). While monitoring traffic in a blade center, usually, you can only monitor traffic on a physical port; however, there are applications that enable you to monitor traffic on only a specific server on a blade (for example, Cisco Nexus 1000V).

Viewing tagged frames going through a VLAN tagged port

Monitoring tagged traffic is not a straightforward mission. The issues of whether you see VLAN tags while capturing data with Wireshark or not will depend on the network adapter you have, the driver that runs over it, and what they do with VLAN tags.

The simplest way to verify that your laptop can capture tagged frames is as follows:

1. Start capturing the tagged port with the port mirror. If you see tags, continue with your work.
2. If you don't see any tags, go to the adapter configuration. In Windows 7, you will get there by clicking on Start and then navigating to **Control Panel | Network and Internet | View Network Status and Tasks | Change Adapter Settings | Local Area Connection**. Next, perform the steps as shown in the following screenshot:



3. Configure the adapter with **Priority & VLAN Disabled**. This will move the tags for the WinPcap driver and for the Wireshark.

Note

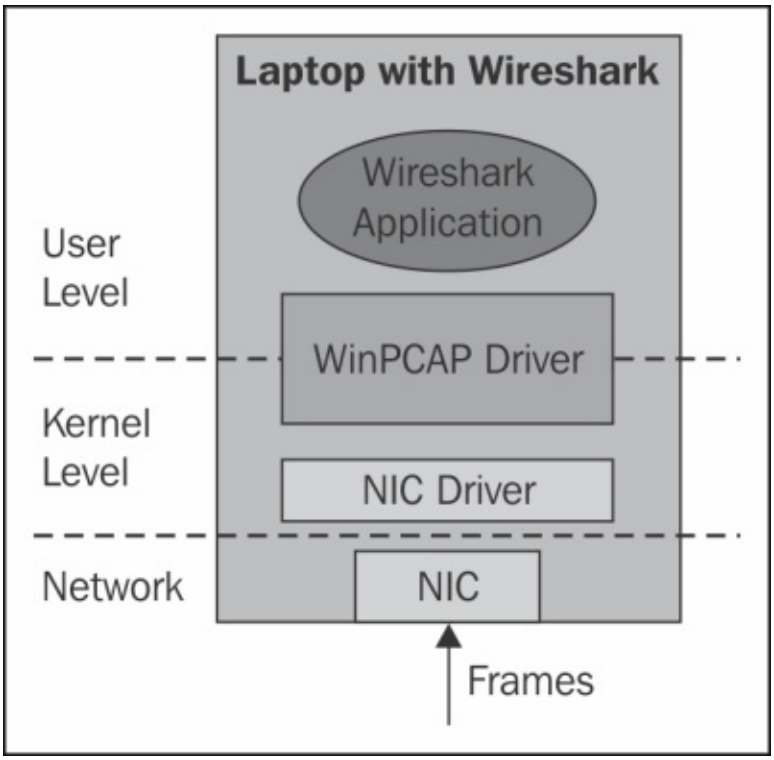
In the previous screenshot we see an example of a Lenovo laptop with Realtek NIC. The illustration gives an example on a popular device, but it can be different on other laptops or servers. The principle should be the same; disable the adapter by extracting the VLAN tag, so it will be forwarded to the WinPcap driver and presented on Wireshark.

How it works...

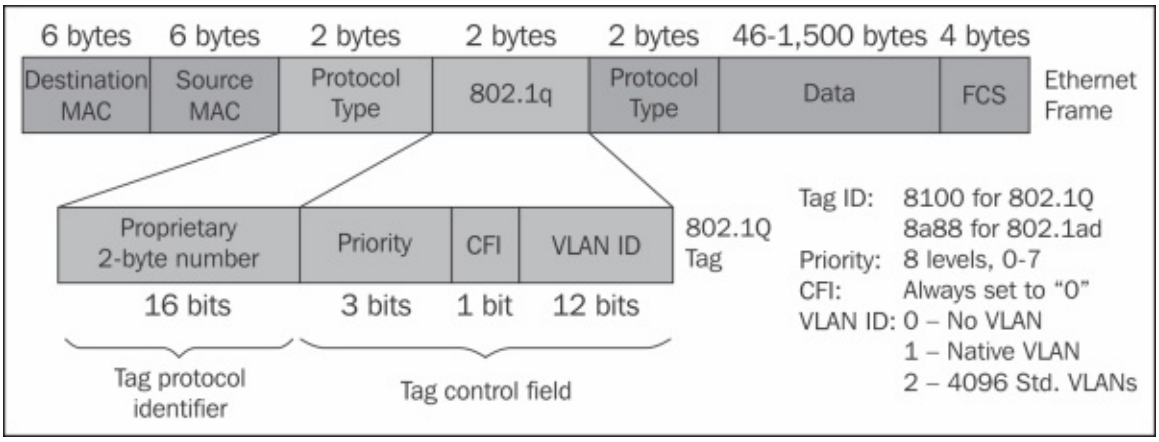
Tags are small pieces of data added to a packet in order to add VLAN information to it. The tag is a 4-byte long string (32 bits), as presented in one of the following diagrams.

Most network adapters and their drivers will simply pass VLAN tags to the upper layer to handle them. In these cases, Wireshark will see VLAN tags and present them.

In more sophisticated adapters and drivers, the VLAN tag will be handled in the adapter itself. This includes some of the most common adapters with Intel and Broadcom Gigabit chipsets. In these cases, you will have to disable the VLAN feature.

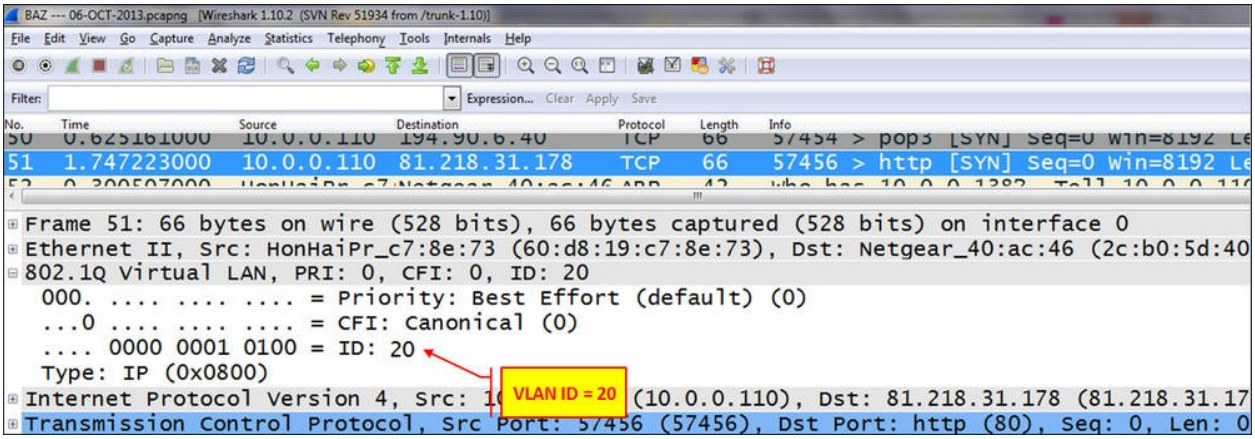


When configuring the NIC driver in order to ensure that it will not handle VLAN tags, packets will simply be forwarded to the WinPcap driver and presented by Wireshark.



VLAN tagging

In the following screenshot you see an example for a tagged frame; the frame is tagged with VLAN ID = 20.



There's more…

Wireshark will also capture double tags, just like in the 802.1ad standard. These tags are what's called service tags and are added at the service provider edge, in order to divide between the provider and the customer tags. The provider tag is called **S-Tag** (802.1ad), and the customer tag is called **C-Tag** (802.1Q). It is also referred to as a QinQ mechanism.

See also

- For more information about WinPcap, go to the Winpcap home page at http://www.winpcap.org/
- For more information on the UNIX/Linux library, refer to the tcpdump home page at http://www.tcpdump.org/

Analyzing wireless (Wi-Fi) problems

Wireless LAN (**Wi-Fi**) became very popular in the last decade, starting from the old 802.11b through 802.11g and to the latest 802.11n standard for high-bandwidth wireless communications.

There are also the emerging standards such as IEEE 802.11ac with products coming in to the market, along with the 802.11ad, which is still under development.

In this recipe we will learn how to resolve Wi-Fi problems, and how to use Wireshark to capture Wi-Fi frames and for basic traffic analysis.

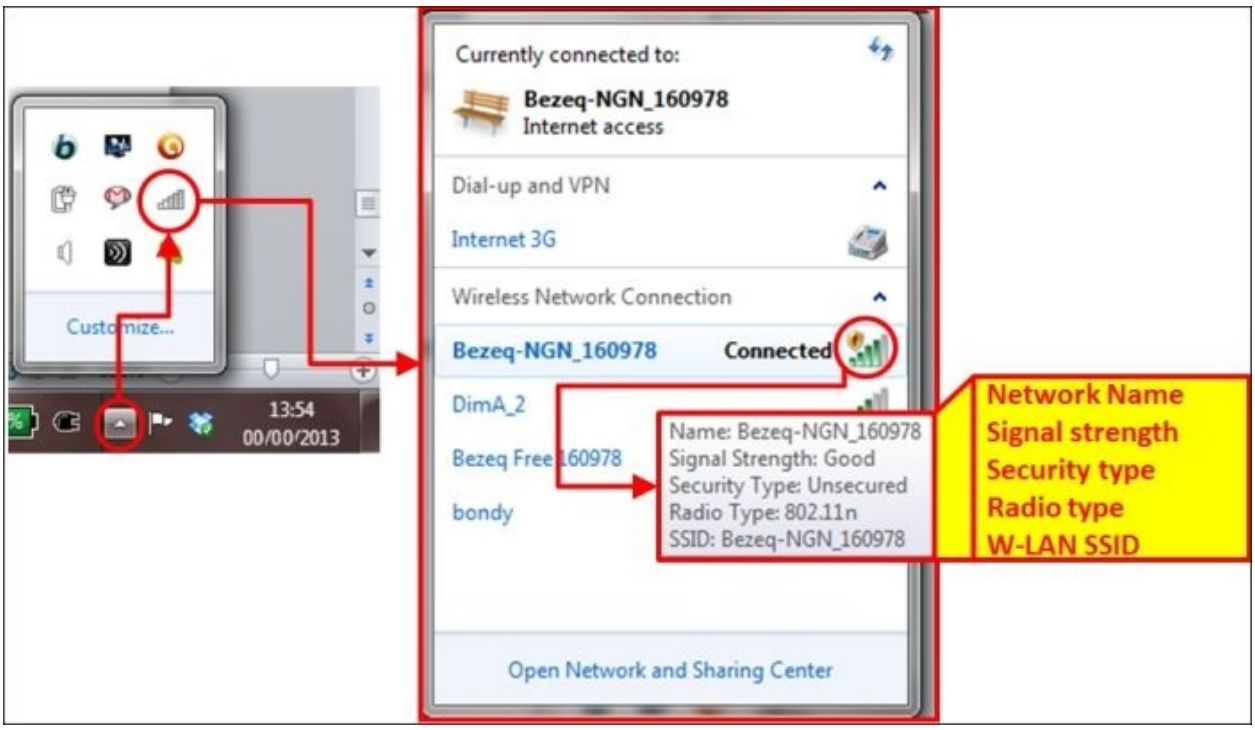
Getting ready

When users complain about bad performance when they connect through a Wi-Fi connection, go as close as you can with your laptop to the user location and verify that you have your Wi-Fi adapter enabled.

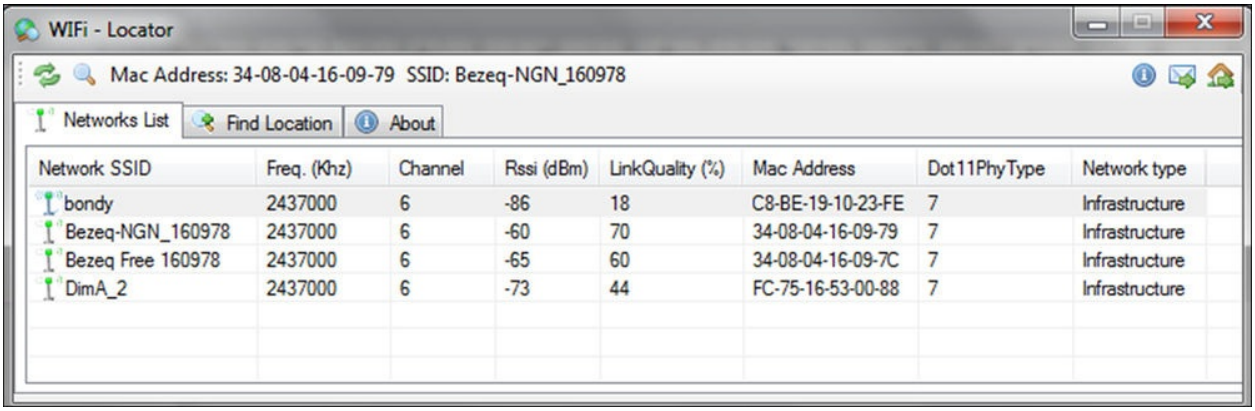
How to do it…

The basic tool is right in the laptop (as we can see in the following screenshot) where, you have the first indication for:

- The signal strength, that is the **Received Signal Strength Indicator** (**RSSI**). In some cases, you will see only the quality of signal; in other cases, you will also see the dBm number
- The access point ID, that is the **Service Set Identification** (**SSID**)
- The security protocol that is used
- Radio type (802.11n as shown in the following screenshot)



You can also use dedicated software, many of them being freeware, to discover available Wi-Fi networks and channels (some of them from the laptop vendors, and some from others). In the following screenshot you can see a list of wireless networks discovered by a software name WIFi Locator (http://tcpmonitor.altervista.org/); however, there are many other software with basic discovery features:



RSSI levels indicate that the higher the number is, the lower is the strength:

- **-60dBm and better**: This indicates a good signal level
- **-80dBm to -60dBm**: This indicates a reasonable signal level
- **-80 dBm to -90dBm**: This indicates a weak signal level
- **-90 dBm and lower**: This indicates a very weak signal

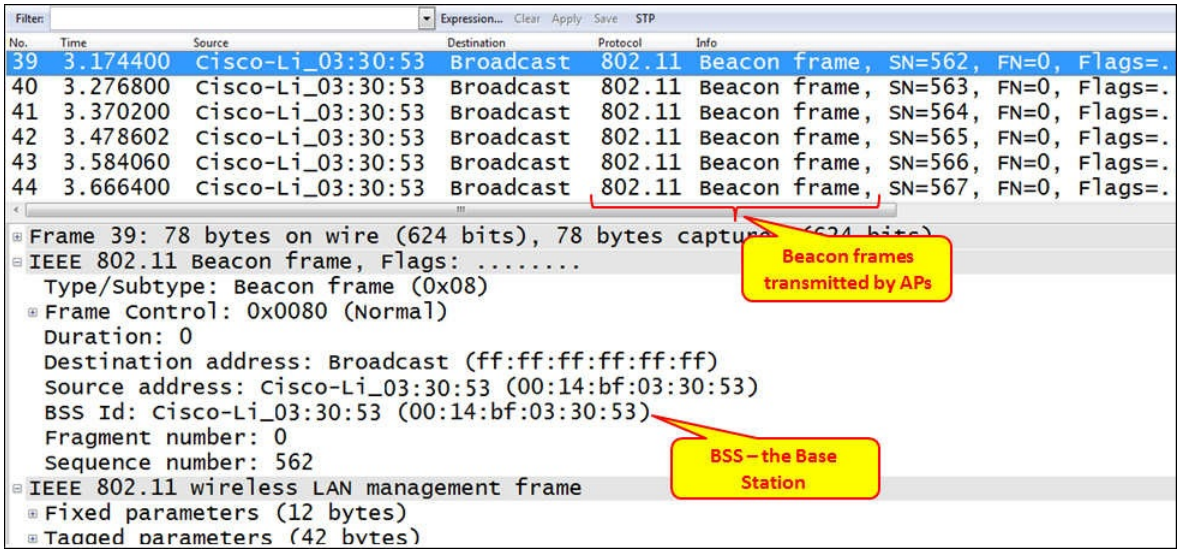
If you have RSSI in the reasonable range and above, the received level is usually enough, and you should look for frequency disturbances and other radio problems.

Note

A rule of thumb that I usually apply for wireless network design is that for standard enterprise applications, I require 75dBm and better, and for wireless networks that should also be used for VoIP, I require -65dBm or better.

If you want to check if there are any disturbances, you can use software that will discover RSSI over time, and will give you a more accurate picture of your network. In the following screenshot you see such a software, called **inSSIDer**; it gives you a more accurate picture about which access points are working and their details.

1. Try to find out the following problems:
    - Different access points (APs) working on the same channel in the same area
    - Low RSSI values (indicated in RSSI numbers lower than -90dBm)
2. The next step is to use spectrum analyzers to check which frequencies are used in your area. You can expect frequency disturbances in areas such as airports, seaports, and military. Spectrum analyzers are available from various vendors such as Fluke Networks, Agilent, and Anritsu.
3. Wireshark can be used to analyze Wi-Fi control frames. The first thing to look for is whether the APs are sending beacon frames. In the following screenshot, you can see these frames:



The stations can send beacon probe request frames to find a nearby access point.

Tip

A probe request frame is sent by a station when it wants to obtain information from another Wi-Fi device, for example, to determine the access points within the range.

The station can also acknowledge the beacon frames coming from the access point in order to register to the AP.

Tip

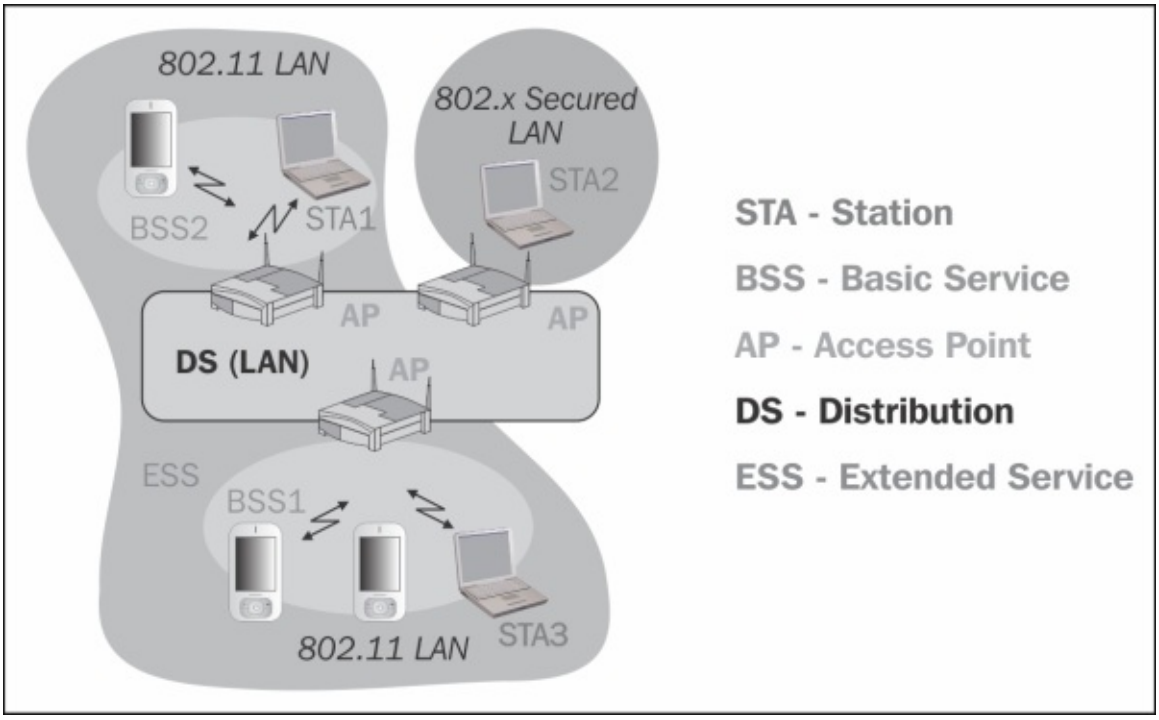
The access point periodically sends beacon frames to announce its presence and relay information. This information includes timestamps, SSIDs, and other parameters with regard to the access point. Radio wireless NICs continually scan all 802.11 radio channels and listen to beacons, in order to choose the best access point to associate with.

4. After accepting and acknowledging the beacon frame, a standard DHCP process will start, as described in Chapter 8, *ARP and IP Analysis*.

How it works…

The Wireless LAN standards are based on the IEEE 802.11 committee. The standards started with 802.11a, 802.11b, 802.11g, 802.11n, and lately 802.11ac and 802.11ad for higher bandwidth.

As seen in the following figure, Wireless LAN networks are based on access points (APs), and wireless clients connect to them.



The most common wireless standard today is the 802.11n, which uses the advanced modulation **Multiple Input Multiple Output** (**MIMO**), to work with up to four antennas and some additional technologies, to increase bandwidth.

Chapter 8. ARP and IP Analysis

In this chapter we will cover the following issues:

- Analyzing connectivity problems with ARP
- Using IP traffic analysis tools
- Using GeoIP to look up physical locations of the IP address
- Finding fragmentation problems
- Analyzing routing problems
- Finding duplicate IPs
- Analyzing DHCP problems

Introduction

In this chapter we will learn how to analyze Layer 3 (IP) and Layer 3 to Layer 2 resolution (ARP). We will discuss the **Internet Protocol** (**IP**), **Address Resolution Protocol** (**ARP**), **Dynamic Host Configuration Protocol** (**DHCP**), routing issues and others, and the problems that you might face while troubleshooting these protocols.

We will start with presenting the protocol's normal behavior for the various protocols and continue with showing what can go wrong and how to solve it.

In general, when we analyze a network problem, we will go bottom up: if you cannot get connectivity, look for the problem in the following order:

1. **Layer 1**: Check if the cable is connected and the link LED on the switch and your PC is turned on. This step is to be executed manually.
2. **Layer 2**: Check if ARP has discovered the MAC address of the destination. In case of a remote location, check for connectivity between every two nodes/routers on the way. This can be executed with the command line (look at the ARP recipe following this).
3. **Layer 3**: Check by using the `ping` command to the destination, and if you do not receive a response, trace route to it. This can be executed with the command line and in some cases with Wireshark.
4. **Layer 4**: Check if the process/server on the other side is answering. This step is to be executed with Wireshark.
5. **Layers 5-7**: Check for application problems. This step is to be executed with Wireshark.

In Chapter 7, *Ethernet, LAN Switching, and Wireless LAN*, we talked about Layers 1 and 2. In this chapter we will talk about Layer 3, that is IP, and the resolving process between Layer 3 and Layer 2, that is ARP. In Chapter 9 , *UDP/TCP Analysis*, we will talk about Layer 4, that is TCP and UDP, and in the rest of the book about application layers.

Analyzing connectivity problems with ARP

ARP is used by IP to resolve the destination MAC address out of the IP address of the device that we wish to communicate with. When we send packets to a destination, the first packet is the ARP request to find the MAC address of the destination. We get it from the destination and then send the other packets destined to it.

Tip

ARP operation is only local, that means the ARP request, which is a broadcast, will be sent only on the LAN. In case you send a packet to a device on your IP network (with the same IP network and mask), ARP will try to find its address. When you send a packet to someone out of your network, ARP will be sent to find out the default gateway MAC address.

Getting ready

We will use three methods to find the basic connectivity problems:

- The standard command line (In MS-Windows go to **Start** and in the command window type `run`. In Linux use any available Shell)
- Wireshark
- Connecting to a LAN switch or router directly and getting information from there

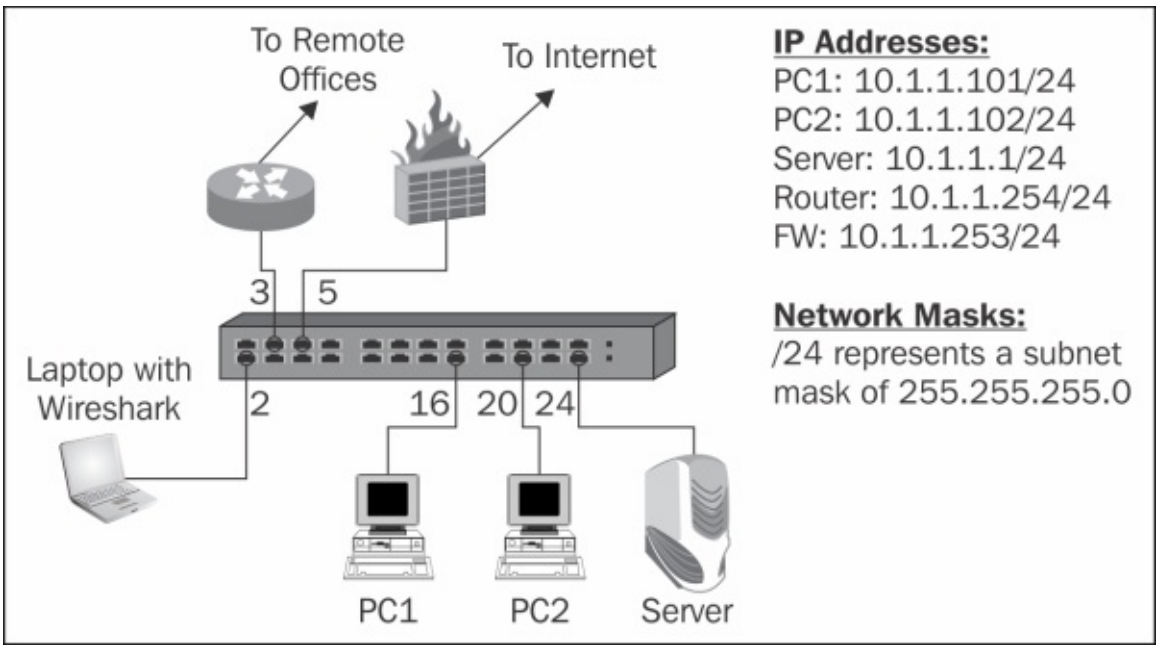
  Tip

  To connect to the communication devices (routers, switches, and so on), you first connect with a console cable (in Cisco, this is the light-blue flat cable); you configure the device with an IP address, and then you can access the device via Telnet, HTTP, or SNMP software. With all of these methods you can read the device counters that provide you with information of the traffic, errors, CPU utilization, and more.

In each one of the following recipes, we will see exactly what to use and where.

How to do it...

In this recipe, we will see several connectivity issues and how to deal with them. Let's look at the very simple network in the following figure. Here, we can see a typical network and discuss some of the connectivity problems that can happen on it.



In the network, we can see two PCs, PC1 and PC2, that are connected to switch ports 16 and 20 respectively, a server connected to switch port 24, a router that connects us to the remote offices on switch port 3, and a firewall that connects us to the Internet on switch port 5. Our laptop with Wireshark installed on it is connected to port 2.

Let's see some of the problems that might occur here and how to solve them.

Let's consider case 1 when there is no connectivity from the PCs to the server:

1. Ping the server from PC1 and PC2.
2. If there is no answer, type `ARP -a` on the command line. In the ARP table, you should see the IP address of the server and its MAC address.
3. If you see the MAC address of the server, probably there is a firewall

running on the server that blocks ICMP requests. For the test, disable the firewall and test again.

Tip

A firewall, VPN client, or an antivirus software that comes with some firewall features can block ICMP requests. Don't forget it while testing network connectivity issues.

4. If you have pinged, but your application still doesn't work, go to Chapter 9, *UDP/TCP Analysis*, and continue from there.

   You've gone through steps 1 to 3 in case 1, and still don't get a ping response (you get request time out).
5. Connect to the LAN switch and get the list of MAC addresses that the switch has learned.

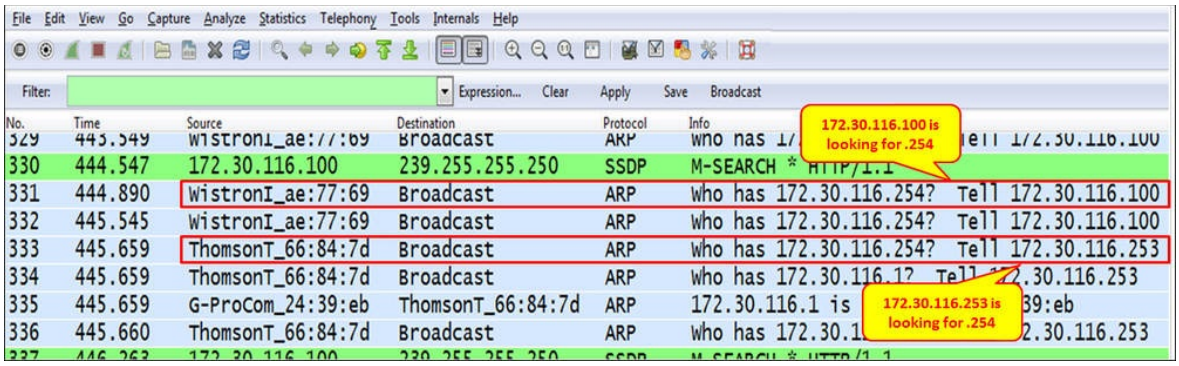
Tip

In every managed switch, you will be able to see the list of MAC addresses that the switch has learned and on which ports the switch has learned them.

In Cisco, the command for this will be `show mac-address-table` or `show mac address table` (depends on the IOS version).

6. If you don't see the addresses of your PC and the server, check for physical problems such as bad cable, adapter problem, and switch port problem. For doing so, you can simply switch cables or replace the switch for the test, replace ports on the switch and so on.

   Now let's see how to use Wireshark to resolve this problem.
7. When clients complain about connectivity problems to a specific device (server, printer, and so on), you connect Wireshark to the port (with port mirror), and you see many ARP requests and no answer. You can see an example of this in the following screenshot:



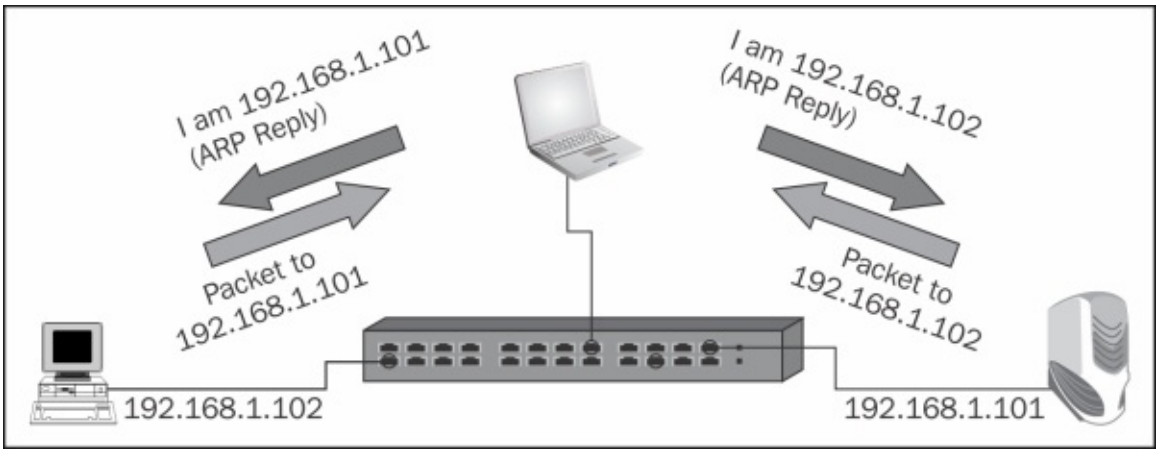
8. In this example, we see that both **172.30.116.100** and **172.30.116.253** are looking for the MAC address of **172.30.116.254**, but there is no reply.
9. In this case, check in the switch if the MAC address has been learned, and if not, check for physical problems.

ARP poisoning and Man-in-the-Middle attacks

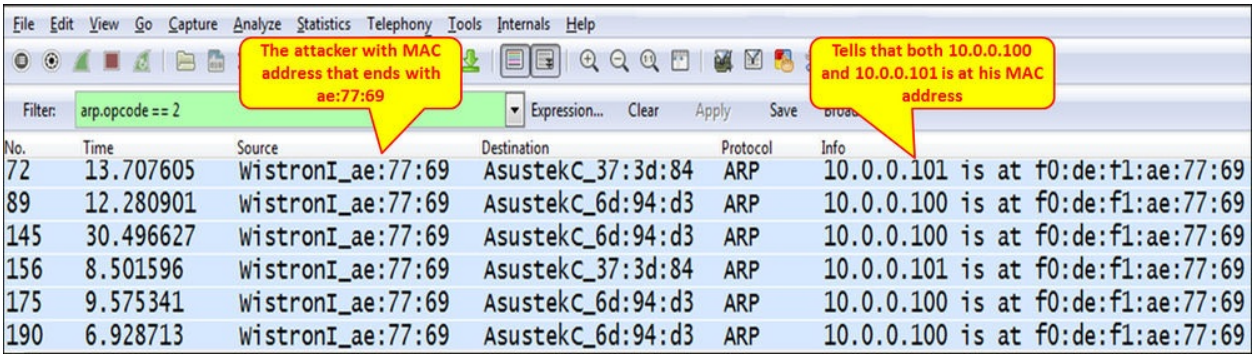
One of the types of a Man-in-the-Middle attack is when an attacker poisons the ARP cache of the devices that he wants to listen to with the MAC address of his Ethernet NIC. Once the ARP cache has been successfully poisoned, each of the victim devices sends all their packets to the attacker while communicating with the other device. The attacker, of course, will resend it to them after reading the data.

This is called Man-in-the-Middle attack since it puts the attacker in the middle of the communication path between the two victim devices. It is also called ARP Poisoning since the attacker actually poisons the victim's ARP cache with wrong information.

In the following figure, we see an example of a Man-in-the-Middle attack:

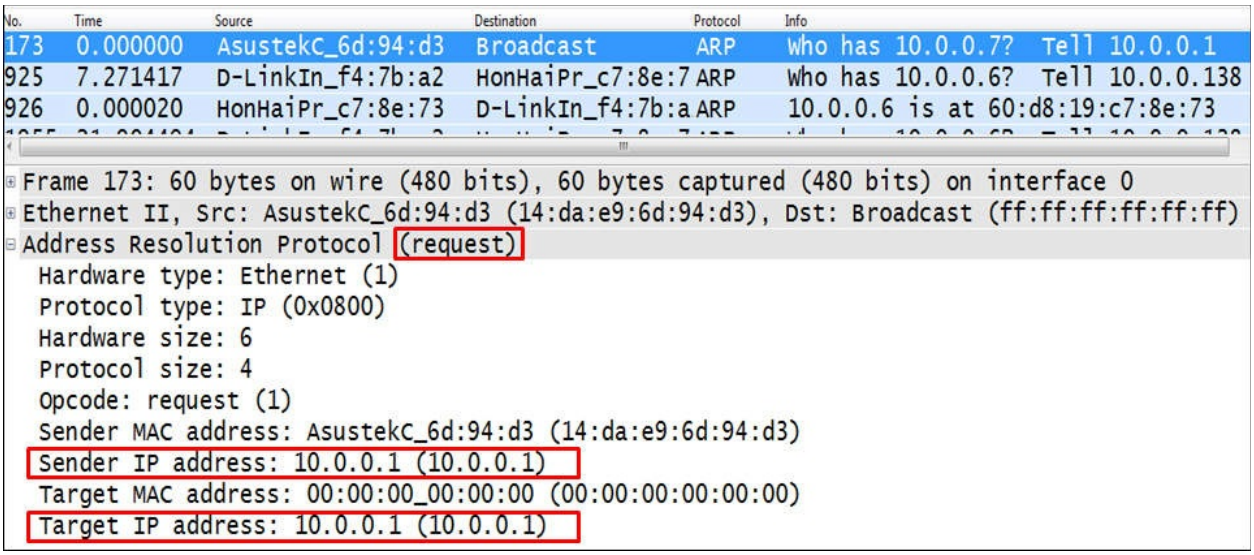


The following screenshot shows Wireshark:



Gratuitous ARP

A gratuitous ARP takes place when a device wants to verify if some other device has its own IP address. In this case, you will see an ARP with the same source and destination address fields. There is nothing wrong with gratuitous ARPs. There are devices that work this way; for example, home routers that scan for attached devices.



ARP sweeps

ARP sweep is used when, for some reason, we see a device that scans the network with ARPs, requests, or response in order to get information or attack the network.

While watching an ARP sweep, just check for the following:

- Is it requests or replies, and who is the sender?
- How many ARPs (per second)?

Requests or replies, and who is the sender

ARP requests and replies are a part of the regular network operation. Here are some rules of thumb to make sure they are actually so:

- Requests from different sources—no problem, this is how a network works (as long as there are not too many of them!)
- Many requests from a singles device—look at the source address and verify who is the device actually sending the requests to:
    - It can be a management system that auto-discovers the network
    - It can be a router that scans to see who is on the local network (see the previous screenshot)
    - If you don't identify the source, it might be a problem, like a worm or

ARP poisoning. Get into the details!

- If you see replies that are not to specific requests, it might be a problem. Get intothe details!

How many ARPs

An ARP is sent when a device wishes to send data to a destination for the first time (see the *How it works...* section). It is difficult to estimate the exact number of ARPs per minute on a network, but this thumb rule gives you a general idea.

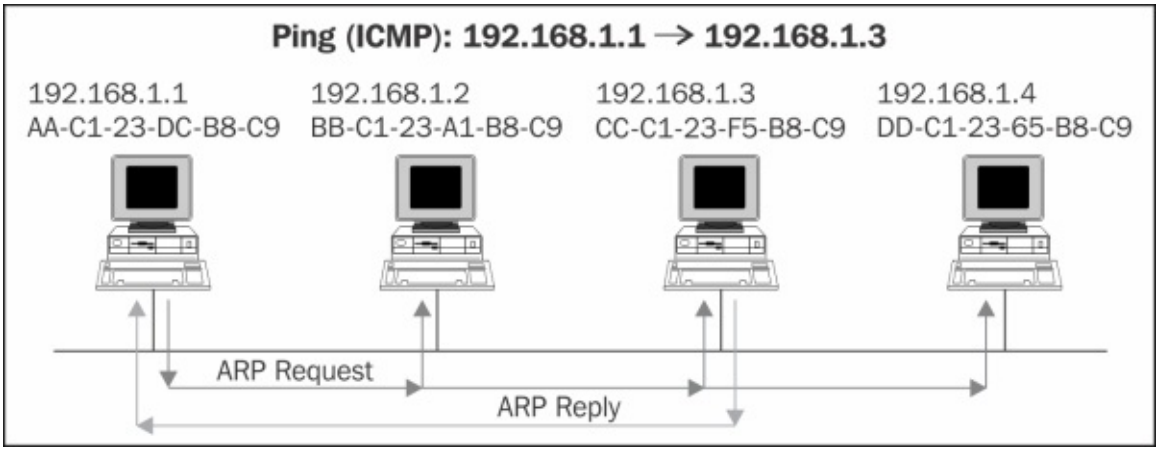
- Around 1-2 ARPs per device per minute is ok. If, for example, you have 100 devices (PCs, servers, printers, and so on) on your network, up to 200 ARPs per minute or 2-3 ARPs per second are still ok.
- Even if you have more than this, for example, 5 and even 10 per second for the network mentioned earlier, it is not necessarily a problem; it will be a good idea to just look for a suspicious pattern.

Tip

The number of ARPs on the network depends on what the devices on the network actually do. If all devices are connecting only to a single device (for example, to a router that connects them to the world), you will see a small number of ARPs. However, if devices, for example, send periodic messages to all their neighbors, you will see many ARPs. Don't forget, God is in the details!

How it works...

When we send a packet from our IP address to a destination IP address through the LAN, we send the data in an IP packet that is encapsulated in an Ethernet frame.



Let's say for example, we send a Ping (ICMP request) from **192.168.1.1** to **192.168.1.3** on the same LAN. To send the packet, we need the IP and the MAC addresses of the destination, but what we have is only its IP address. We know it because this is our destination.

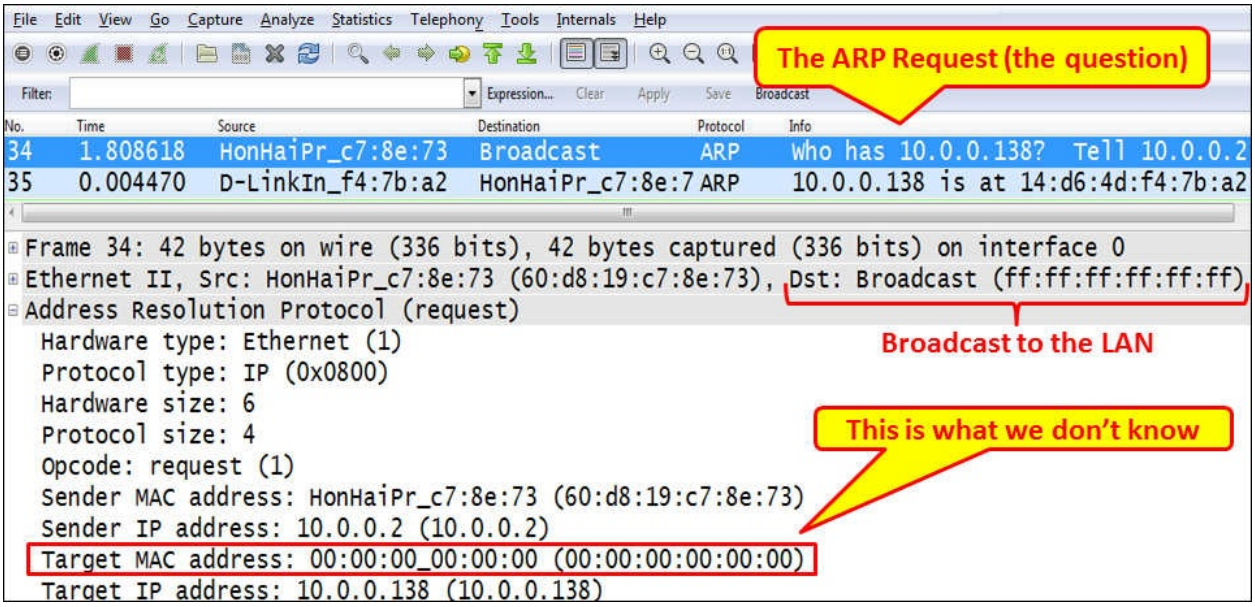
In order to find out what is the MAC address of the destination, we simply send an ARP broadcast to the LAN, asking all devices attached to it: who has the IP address **192.168.1.3**? If the destination station is alive, it sends an ARP response with the MAC address of the destination.

From this moment, the source station holds the data in a cache, the **ARP Cache**, and the next time it wants to transmit, the station transmits the data directly to the destination address.

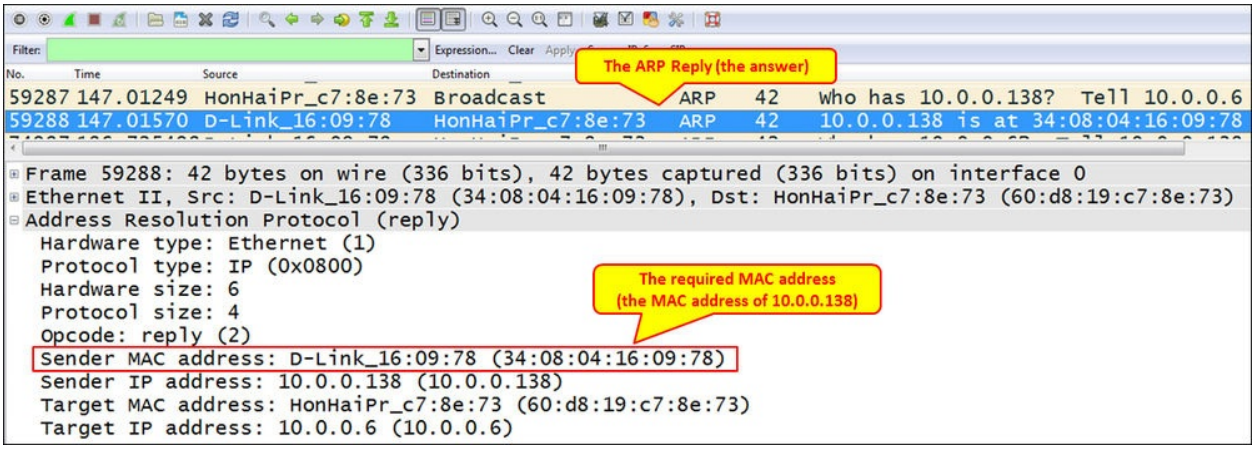
The ARP Cache remains in the host buffer for the next several minutes (how many minutes depends on the operating system). The ARP entry is flushed a few minutes after the last packet is sent to the destination.

The ARP request will be a broadcast sent to the LAN, as you see in the

following screenshot:



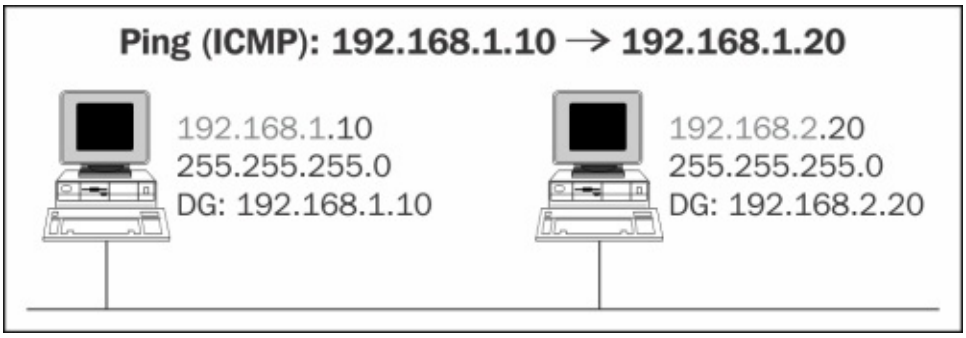
The ARP reply will be a packet sent from the destination that we looked for to the source with the required MAC address, as you see in the following screenshot:



Anything that doesn't look like the standard and is a known exception (for example, gratuitous ARP, ARP sweep by router) should be checked!

There's more...

Here is a short example to understand the principle of ARP operation. In the following diagram, we see an interesting case: two devices attached to the same LAN with different subnets with a default gateway configured to their own IP address. The question is: will a Ping (or any communication) work between them?



Well, intuitively, you will say no because these are two devices on different subnets and therefore require a router between them. But let's look at what actually happens over the wire:

- **192.168.1.10** wants to send a packet to **192.168.2.20**. Since the default gateway is configured to its own address, it thinks the destination is on the same LAN and sends an ARP request.
- Since we are on the same LAN as the destination, the destination receives the ARP request (because it is a broadcast) and answers to **192.168.1.10** with its MAC address.
- Now, the source has the MAC address of the destination. It sends the packet to it, and although not on the same IP subnet, the destination receives the packet.

Another important issue is Proxy ARP. While a proxy in communications is a device that performs an operation on behalf of someone else, Proxy ARP is the technique in which one host, usually a router, answers the ARP requests intended for another machine.

The proxy ARP concept is implemented in various cases, for example:

- While placing a device in front of a router, for example, WAN acceleration/optimization device. When you configure this device in bridge (or transparent) mode, it will answer to the ARP requests intended for the router.
- Firewalls, web filters, and other devices that work in the transparent mode and are located in front of a server.
- In case of a software that requires an IP address in addition to the IP of the server it is installed on.

Using IP traffic analysis tools

IP is the network protocol in the TCP/IP protocol stack that carries all upper layer information. Whether it is HTTP, Video, IP Telephony, or other application, IP will be the Layer-3 protocol for all of them. In this section, we will look at some tools that will help us with the analyses of IP traffic.

Getting ready

Just open Wireshark, connect it to the network, configure port mirror to the device you want to test, and start it.

How to do it...

There are several tools and configurations that will help you with the analysis of IP traffic. Among them are:

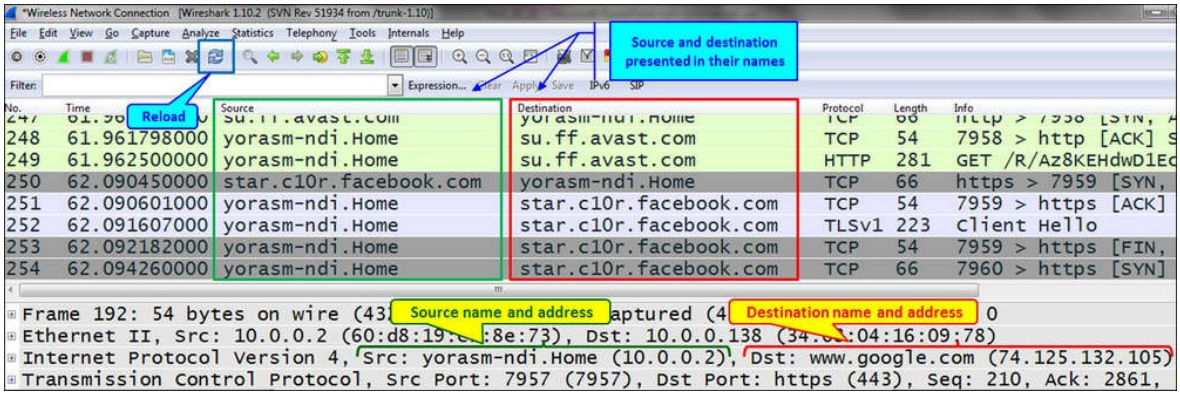
- IP statistics
- IP name resolution

IP statistics tools

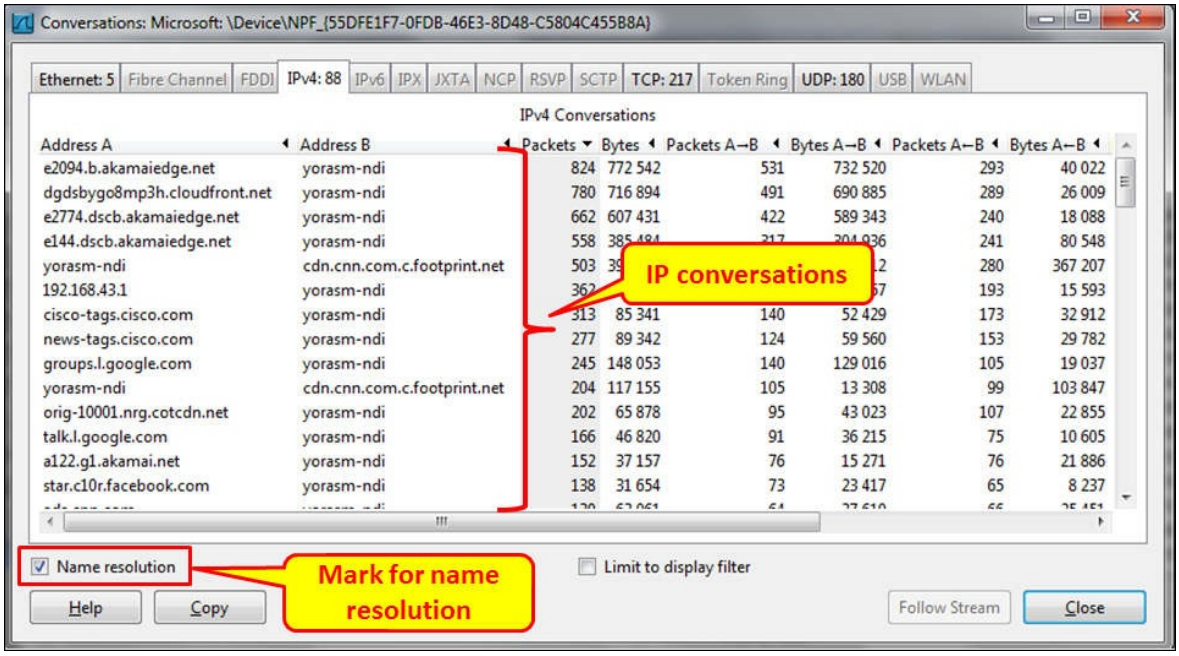
When you monitor a communication line, connectivity to a server, traffic to the Internet, or any other type of traffic, there are several tools for monitoring the source and destination IPs.

Following are the steps for seeing the source and destination IPs:

1. From the menu, choose **View** | **Name Resolution** and mark **Enable** for the Network layer. If you are watching an existing file, after you make the change, click on the **Reload** icon. The capture screen will be presented with DNS names in addition to IP addresses.



2. In order to see the statistics, choose from the **Statistics** | **Conversations** menu and mark **Name resolution** at the bottom-left corner of the window, as illustrated in the next screenshot:


Conversations: Microsoft: \Device\NPF_{55DFE1F7-0FDB-46E3-8D48-C5804C455B8A}
Ethernet: 5
Fibre Channel
FDDI
IPv4: 88
IPv6
IPX
JXTA
NCP
RSVP
SCTP
TCP: 217
Token Ring
UDP: 180
USB
WLAN
IPv4 Conversations
Address A
Address B
Packets
Bytes
Packets A→B
Bytes A→B
Packets A←B
Bytes A←B
e2094.b.akamaiedge.net yorasm-ndi 824 772 542 531 732 520 293 40 022
dgdsbygo8mp3h.cloudfront.net yorasm-ndi 780 716 894 491 690 885 289 26 009
e2774.dscb.akamaiedge.net yorasm-ndi 662 607 431 422 589 343 240 18 088
e144.dscb.akamaiedge.net yorasm-ndi 558 385 484 317 304 936 241 80 548
yorasm-ndi cdn.cnn.com.c.footprint.net 503 39 280 367 207
192.168.43.1 yorasm-ndi 362 193 15 593
cisco-tags.cisco.com yorasm-ndi 313 85 341 140 52 429 173 32 912
news-tags.cisco.com yorasm-ndi 277 89 342 124 59 560 153 29 782
groups.l.google.com yorasm-ndi 245 148 053 140 129 016 105 19 037
yorasm-ndi cdn.cnn.com.c.footprint.net 204 117 155 105 13 308 99 103 847
orig-10001.nrg.cotcdn.net yorasm-ndi 202 65 878 95 43 023 107 22 855
talk.l.google.com yorasm-ndi 166 46 820 91 36 215 75 10 605
a122.g1.akamai.net yorasm-ndi 152 37 157 76 15 271 76 21 886
star.c10r.facebook.com yorasm-ndi 138 31 654 73 23 417 65 8 237
IP conversations
Name resolution
Mark for name resolution
Limit to display filter
Help
Copy
Follow Stream
Close

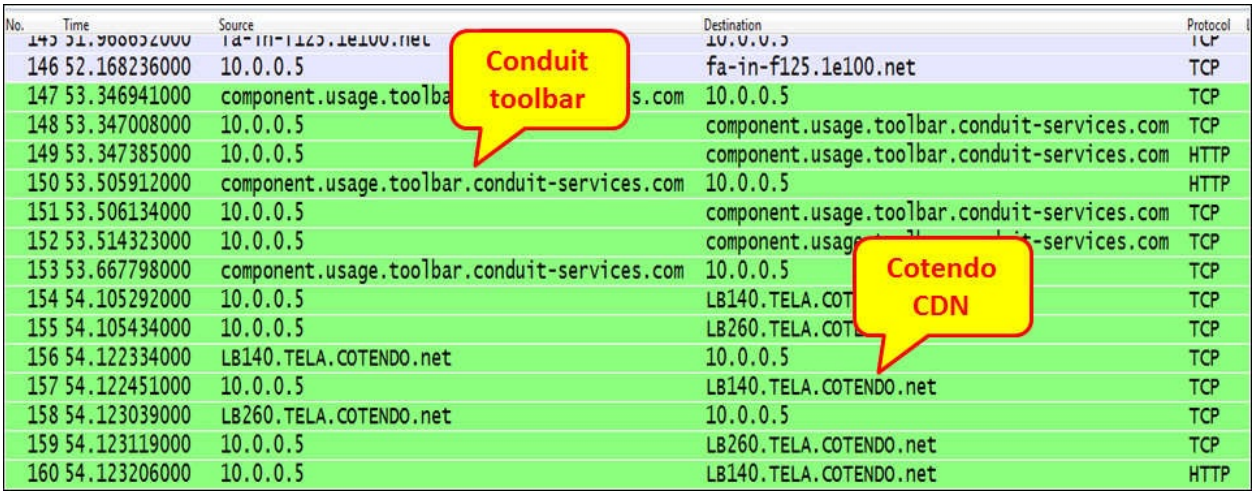
How it works...

This is very simple. Wireshark uses the DNS server configured on your laptop in order to translate the IP addresses to names. In some cases, it can be very helpful to find out problematic traffic patterns. These can be, for example:

- Traffic to websites that is not allowed according to company policy.
- Automatic software updates, for example, Anti-virus websites and Microsoft updates. The solution to this is the central servers that download the software while all company PCs get the software and updates from this server.
- Toolbar traffic can cause a huge amount of traffic if installed on organization devices (think about 50-100 opened connections on every device in your company in addition to regular traffic).

There's more...

You can see, for example, a browser configured with the Conduit toolbar. The moment you run it, you will see many connections to the websites that you know, and to the websites that you don't. Here, for example, you see connections to the Conduit website, and also to a **Content Delivery Network** (**CDN**) vendor.



No.	Time	Source	Destination	Protocol
146	52.168236000	10.0.0.5	fa-in-f125.1e100.net	TCP
147	53.346941000	component.usage.toolba	10.0.0.5	TCP
148	53.347008000	10.0.0.5	component.usage.toolbar.conduit-services.com	TCP
149	53.347385000	10.0.0.5	component.usage.toolbar.conduit-services.com	HTTP
150	53.505912000	component.usage.toolbar.conduit-services.com	10.0.0.5	HTTP
151	53.506134000	10.0.0.5	component.usage.toolbar.conduit-services.com	TCP
152	53.514323000	10.0.0.5	component.usage	TCP
153	53.667798000	component.usage.toolbar.conduit-services.com	10.0.0.5	TCP
154	54.105292000	10.0.0.5	LB140.TELA.COT	TCP
155	54.105434000	10.0.0.5	LB260.TELA.COT	TCP
156	54.122334000	LB140.TELA.COTENDO.net	10.0.0.5	TCP
157	54.122451000	10.0.0.5	LB140.TELA.COTENDO.net	TCP
158	54.123039000	LB260.TELA.COTENDO.net	10.0.0.5	TCP
159	54.123119000	10.0.0.5	LB260.TELA.COTENDO.net	TCP
160	54.123206000	10.0.0.5	LB140.TELA.COTENDO.net	HTTP

To see the exact website and pages, you can, of course, select **Statistics** | **HTTP** and choose the relevant feature (with IP configured as filter).

Some rules for efficient usage of toolbars:

- Have a policy about what to use and what not, and block users from installing toolbars that are not allowed
- Monitor your line to the Internet, and make sure where the traffic is going

Using GeoIP to look up physical locations of the IP address

Wireshark 1.1.2 and the higher versions can use GeoIP (commercial version) and GeoLite (free version) databases to look up the city, country, AS number, and other information for an IP address discovered by Wireshark.

Getting ready

1. Go to the following website: http://dev.maxmind.com/geoip/geolite.
2. For IPv4, download the following files (the binaries):
    - GeoLite Country
    - GeoLite City
    - GeoLite ASN

- For IPv6, download the following files:
    - GeoLite Country (IPv6)
    - GeoLite City (IPv6)
    - GeoLite ASN (IPv6)

      Tip

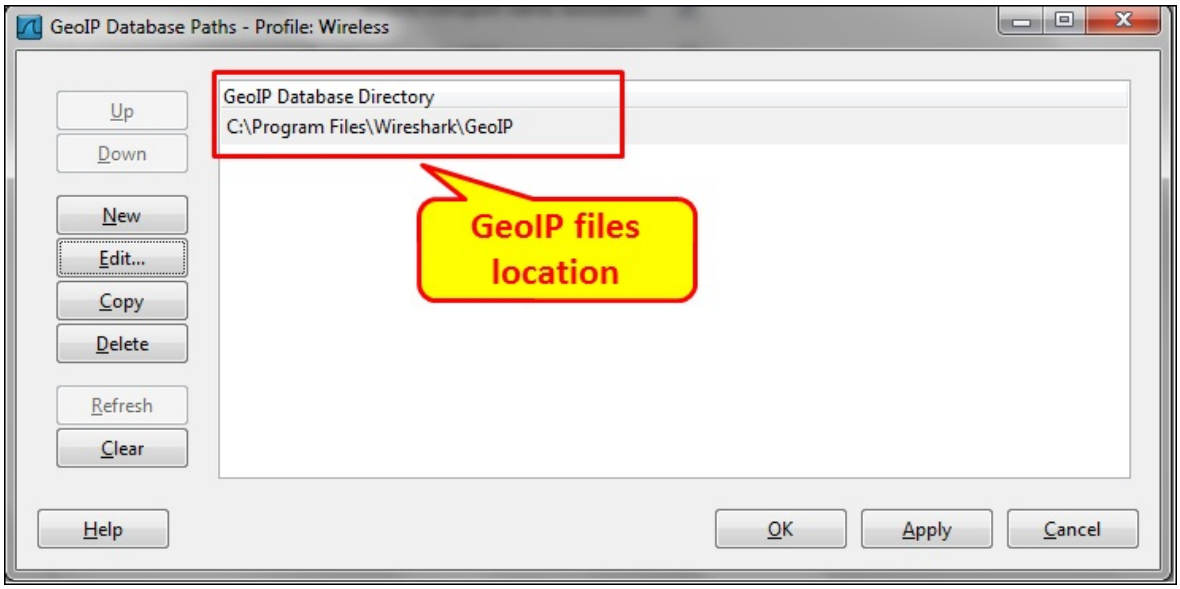
      Autonomous System (**AS**) is a term used in **Exterior Gateway Protocols** (**EGPs**), for identifying all routers under the control of the same network operator. When you connect to the Internet through two different **Internet Service Providers** (**ISPs**), you will get your own AS, while the two ISPs have their ASe While configuring connectivity to the Internet with two different **Internet Service Providers** (**ISPs**), ASs are configured along with an EGP routing protocol. The market standard for EGP protocol is **Border Gateway Protocol version 4** (**BGPv4**).

You will get the binary files with the country, city, and Autonomous System (AS) numbers.

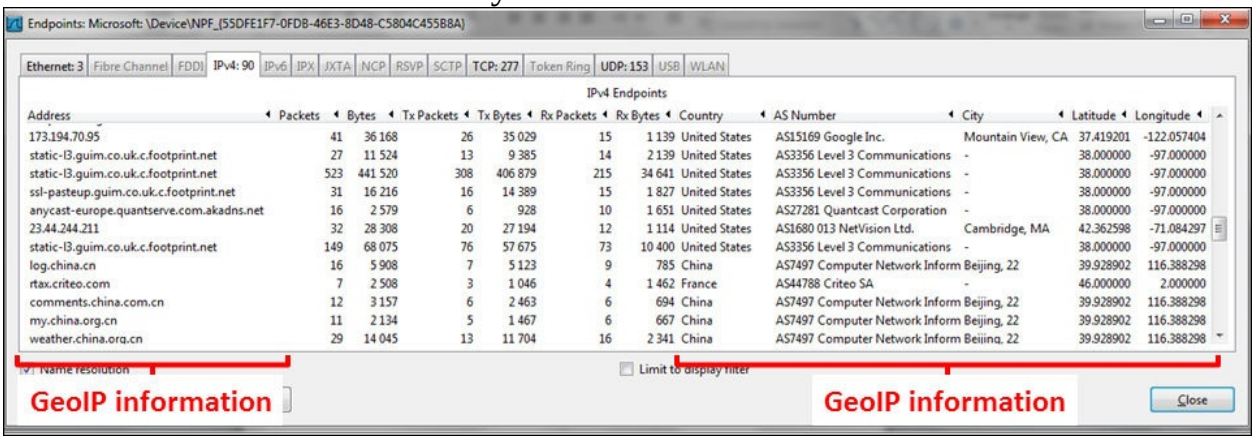
How to do it...

After you have downloaded the files, follow these steps:

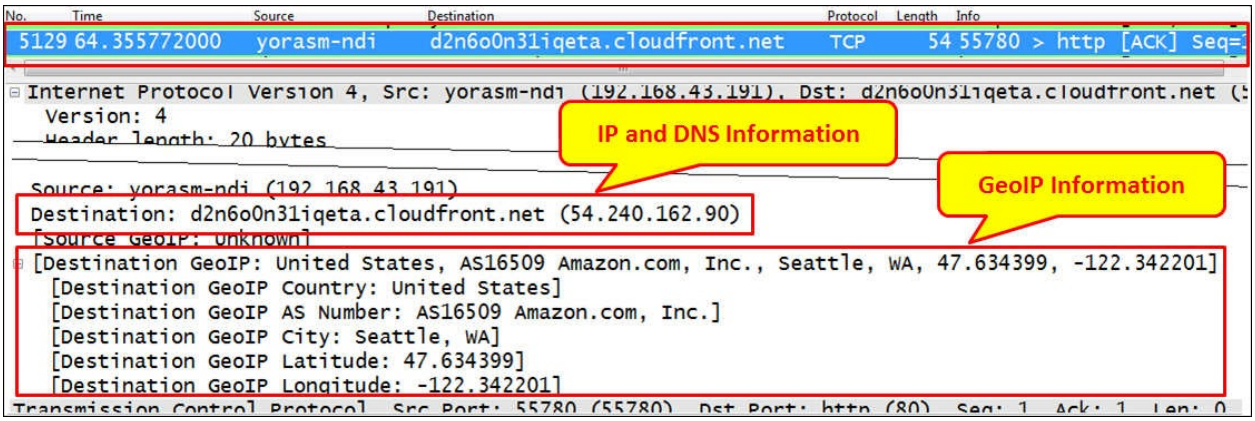
1. Put all of the files in the same directory (you can also put them in different directories, but it will be less convenient).
2. Now, you must tell Wireshark where the files are. Go to **Edit** | **Preferences** | **Name Resolution** and select **GeoIP database directories**.
3. Add the full path of the GeoIP directory, as shown in the following screenshot:



- Click on **Apply** and close the window and restart Wireshark.
- Now, start Wireshark (or open saved file), select **Statistics** | **Endpoints**, and see the GeoIP information in any of the tabs that contains the IP addresses:



- You can also see the GeoIP data in the IP packet detail tree. To enable this, go to **Edit** | **Preferences** | **Protocols** | **IP** and make sure that **Enable GeoIP lookup** is checked.



How it works...

The IP addresses are provided by **Internet Assigned Numbers Authority** (**IANA**), a suborganization of the Internet Standard Organization (ISO), to regional organizations called Regional Internet Registrars (RIPE-NCC, APNIC, AFRINIC, LACNIC, and ARIN), who then allocate them to national ISPs, and national ISPs allocate them to individual customers. GeoIP simply is a database of these locations, so it resolves the IP addresses that Wireshark captures according to this database.

The GeoLite files are free IP geographical location databases that are updated monthly. It can be found at http://dev.maxmind.com/geoip/geolite#IP_Geolocation-1.

There's more...

The GeoIP can be used for several reasons:

- To view the sites (websites, FTP servers, and so on), that people in your organization are connecting to
- To resolve source IP addresses of connections that are coming from the world to your organization
- For fun

Finding fragmentation problems

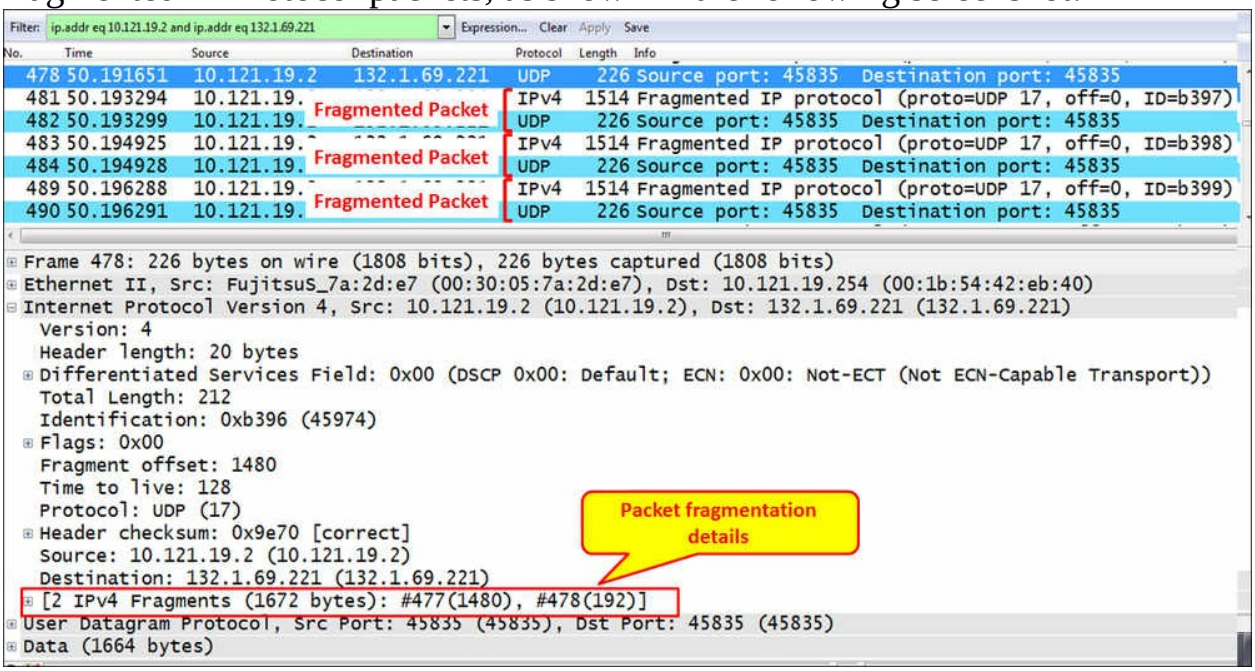
Fragmentation is a common mechanism in IP that takes a large IP packet and divides it into smaller-size packets that will fit in the Layer-2 Ethernet frames. In most of the cases, there shouldn't be any problems with the mechanism, but there might be performance issues due to this mechanism.

Getting ready

Just open Wireshark, connect it to the network, configure port mirror to the device that you want to test, and start it. Fragmentation will mostly influence interactive applications such as databases, and these are the places where we should look for problems.

How to do it...

When fragmentation takes place, you will see UDP or TCP packets along with fragmented IP Protocol packets, as shown in the following screenshot:



While suspecting performance problems, for example, a database client that experiences slow connectivity with the server, follow these steps to see if the problem is due to fragmentation:

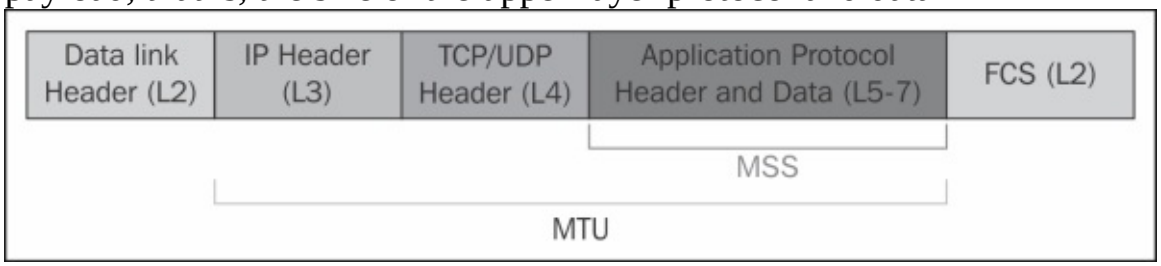
1. Test the connectivity between clients and the server to verify that there are no other problems.
2. Look for fragmentation between the client and the server. Fragments will be shown as in the previous screenshot (IPv4 fragments).
3. In the case that you suspect fragmentation to be the reason for the problem, contact a good **Database Administrator** (**DBA**) that will tune the database to send out packets that do not cause fragmentation to the network.
4. The recommended packet size in Ethernet is not greater than 1460 bytes minus the TCP header size. Thus, the segments coming out of the interface should have a size of 1420-1440 bytes.

In cases where we need more bytes for the header, for example, when we use tunneling mechanisms and TCP options, the DBA will have to reduce this size even more. The best way will be simply to reduce it to such a size that you will not see any fragments.

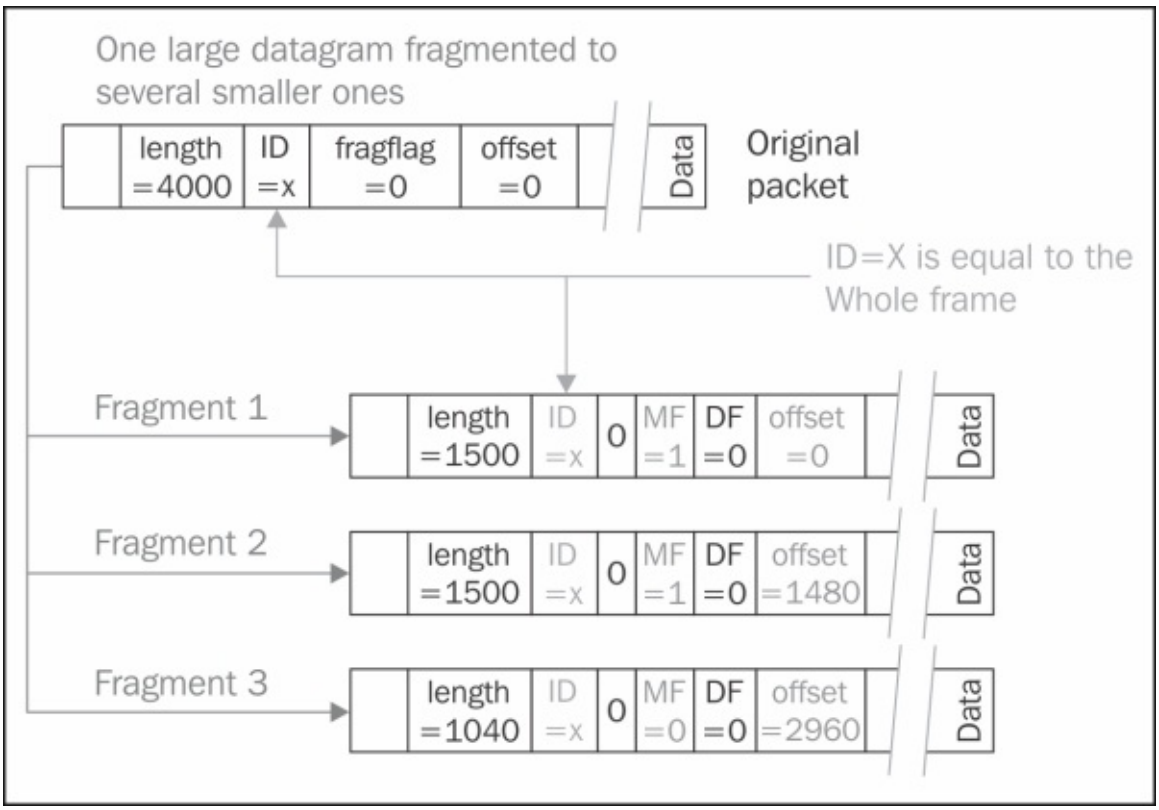
How it works...

It is important to understand two terms that define the size of the data units that are sent over the network, as you see in the following diagram:

- **Maximum Transfer** (or **Transmission**) **Unit (MTU)**: This is the size of the IP packet including the header and the data
- **Maximum Segment Size (MSS)**: This is the maximum size of the TCP payload, that is, the size of the upper-layer protocol and data



The fragmentation mechanism that is used in IPv4 works as shown in the following illustration:



1. An original large packet enters the NIC or the router with a packet size that needs to be fragmented.
2. The packet is fragmented into several parts depending on the original size.
3. For the fragmentation, we have these fields:
    - **ID**: This is identical to the ID of the original packet
    - **Bit 0**: Always 0
    - **Bit 1 (DF Bit)**: 0 = May Fragment, 1 = Don't Fragment.
    - **Bit 2 (MF Bit)**: Don't Fragment: 0 = Last Fragment, 1 = More Fragments
    - **Fragment offset**: This indicates the number of bytes from the beginning of the original packet

In IPv4, the NIC itself can fragment the packet along with every router on the way to the destination.

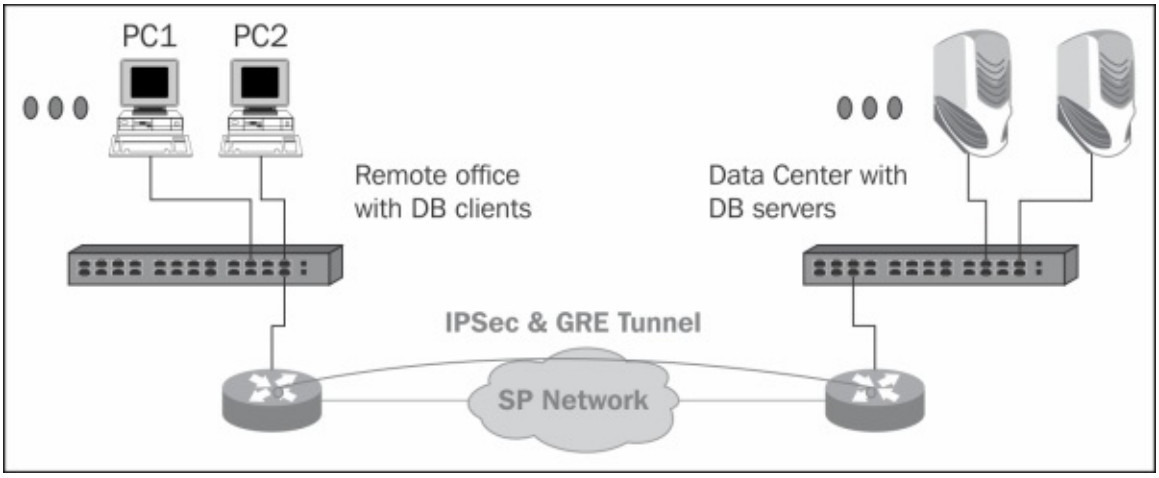
In IPv6, fragmentation can be done only by the sender and not by the routers to

the destination. In IPv6, fragmentation is implemented by the extension headers.

There's more...

A packet can be fragmented several times on the way to the destination, while in any case, it will be reassembled by the end device only.

In the example in the following illustration, we see a part of a large network in which the customer has several hundred remote offices connection to a central data center through a **service provider** (**SP**) network.



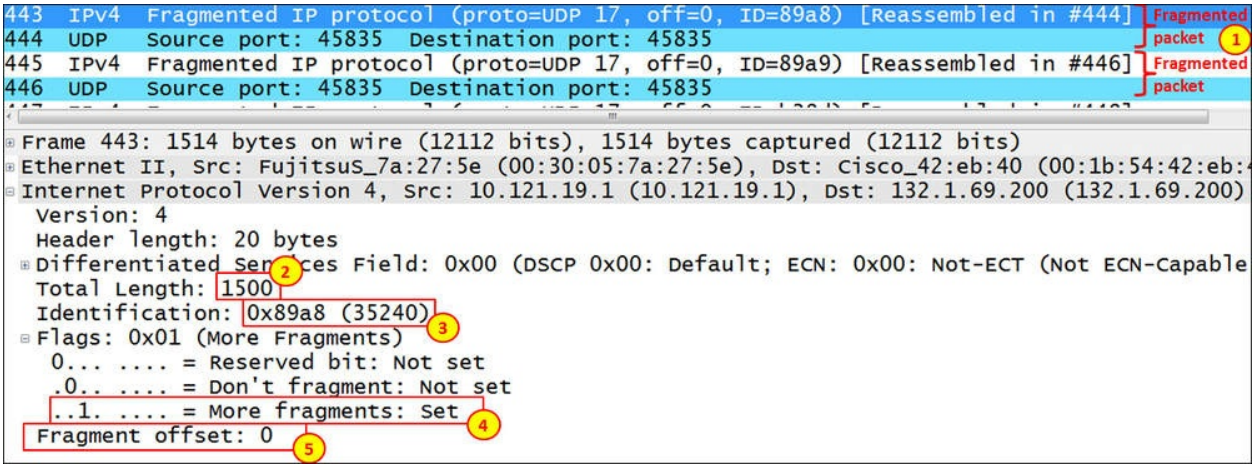
In the remote offices, there were 5 to 10 PCs with DB clients connecting to the DB servers in the central data center. IPSec and GRE tunnels were used for encrypting the data through the SP network.

The problem was that in some of the database applications, the database created frames of 1800 bytes that were fragmented:

- First, it was created in the NIC and sent out of it in two fragments: 1500 bytes and 300 bytes
- Second, it was created in the router because the tunnel required some bytes for itself that divided the 1500 bytes frame to 1420 bytes and 80 bytes frame

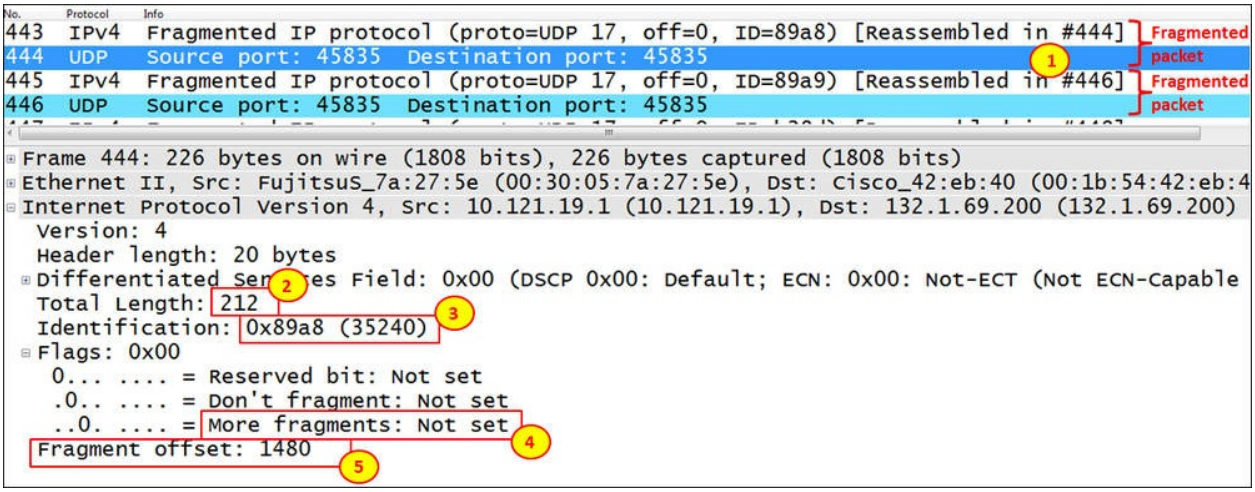
The bottom line is that for every packet sent by the PC the servers received 3 packets, and since the customer had several thousand clients and quite an old server, the whole thing worked very slowly.

In the next screenshot, you see the packets when they leave the client. In the first packet, which is the first fragment leaving the NIC, we see:



In the preceding screenshot, we see that packets 443 and 444 (1) are both fragments of the original packet. In packet 443, we see that the total length is 1500 bytes (2), the ID is 0x89a8 (3), more fragments flag is set (4), meaning that there are more fragments to follow, and the fragment offset is 0 (5), meaning that this is the first fragment in the stream.

In the next screenshot, we see the next fragment, that is, packet 444 in the capture file:



In packet 444, we see that the total length is 212 bytes (2), the ID is 0x89a8 (3), which is the same as in packet 443, more fragments flag is not set (4) meaning that this is the last fragment from the original packet, and the fragment offset is 1480 (5), meaning that this is the second fragment from the original packet.

Analyzing routing problems

One of the most critical issues in networks is routing. These include routing loops, no route to destination, and many more. Most of the routing problems will not require using Wireshark in order to solve them. In most of the cases, some knowledge of routing principles and protocols along with common sense will do the job. In this recipe, we will try to provide some basic tips along with some basic issues such that Wireshark can be of assistance.

Getting ready

First, make sure you are familiar with the very basic commands, `Ping` and `Tracert` (or `Traceroute`). In most of the cases, these commands along with logging in to the routers will help you with solving the problems.

In this recipe, we will show some important things on the captured file that can indicate a routing problem.

How to do it...

In this section, we will not give a recipe of what to do, like we usually do, but rather mention things to watch and notice.

Among the things you should notice, the crucial ones are:

- The first and most important, **Time To Live** (**TTL**) messages. A TTL value of 0 should raise an alert since the meaning of it in most of the cases is a loop. Wireshark will not tell you where the loop is coming from, but seeing these messages is an alert to something that went wrong. A typical message will be: TTL expired in transit.
- The following ICMP message should indicate a configuration problem in a router or in several routers:
    - **Destination network unreachable**: It usually indicates a missing route in one of the network routers
    - **Destination host unreachable**: It usually indicates a device (for example, a PC) on the destination network that is not connected to the network or a default gateway is not configured on it
- Another issue that should raise a flag is when you see packets going from a source IP address to the destination, back to the source, back to the destination, and so on, while the TTL value is reduced by one in every packet, which is a clear indication of a loop.

How it works...

The TTL is an 8-bit field in the IP header that is implemented in the following way:

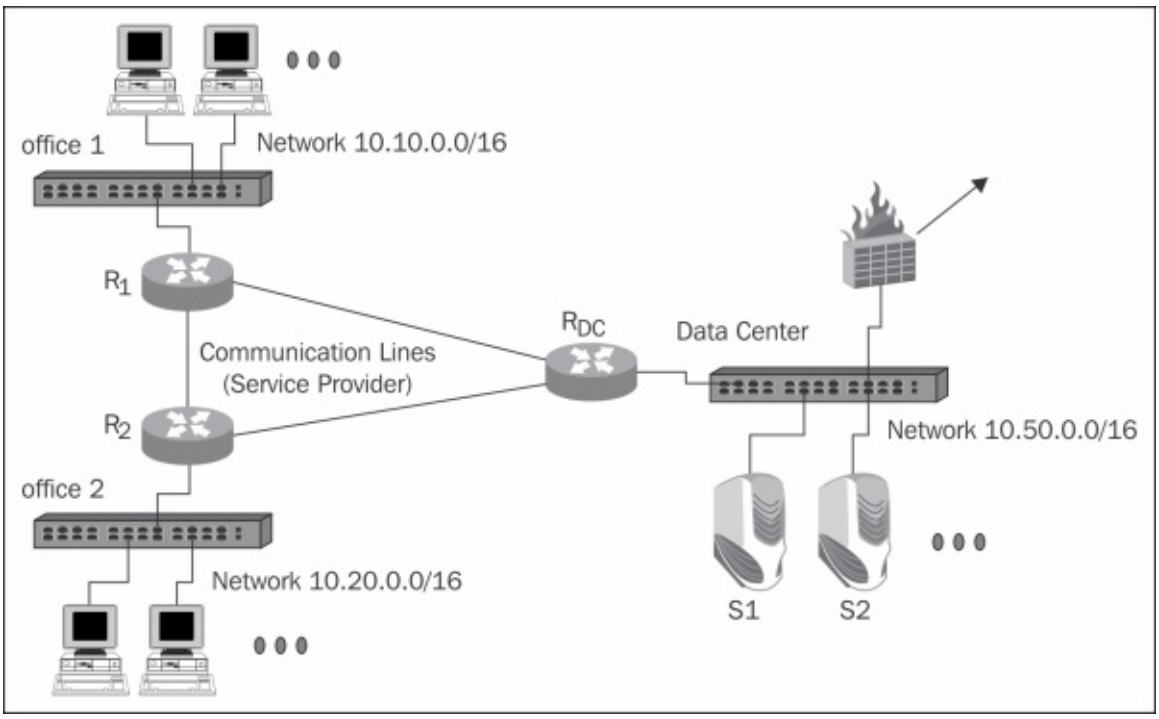
- The sender inserts a number to it. The number value is usually 64, 128, or 256, depending on the operating system that sends the packet.
- Each router decrements this value by one. If, for example, a packet is sent with a value of 128 and crosses 10 routers, the TTL value will be 118.
- A router that will see a value of 1 in the TTL field will decrement it to 0 and drop the packet, as well as send an ICMP error message to the source address from which it has received the packet.

Tip

The TTL field in the IP packet can tell us how many routers the packet has crossed on the way from the source to us. This is due to two assumptions: first, while sending a packet from end-to-end through the Internet, it will not cross more than 30 hops (routers); this is the way the Internet is planned, and second, the sender inserts a value of 64, 128, or 256 in the TTL field. If, for example, we see a TTL value of 110, the meaning is that it has crossed 18 routers on the way to us (128-110) because it cannot be that it has crossed 146 routers (256-110).

There's more...

A typical routing problem can be seen in the following illustration. In this network, we have a central data center with two remote offices. The network was built this way in order to provide redundancy from the remote offices to the central data center.



EIGRP is running in the routers, in addition to static routes to the Internet. A partial (and relevant to the case) routing table is presented in the following figure (only routes that are relevant to the example are resent):

Router	Destination Network	Next Hop	Admin Cost	Discovery Protocol
R_{DC}	10.10.0.0/16	R_1	5	EIGRP
	10.20.0.0/16	R_2	5	EIGRP
	0.0.0.0/0	FW	1	Static
R_1	10.50.0.0/16	R_{DC}	5	EIGRP
	0.0.0.0/0	R_{DC}	1	Static
R_2	10.50.0.0/16	R_{DC}	5	EIGRP
	0.0.0.0/0	R_{DC}	1	Static
FW	10.10.0.0/16	RDC	1	Static
	10.20.0.0/16	RDC	1	Static
	0.0.0.0/0	ISP	1	Static

The problem was that we've had a very rare case in which both lines to the center, R1 to RDC, and R2 to RDC were disconnected (a tractor that cut both in the last mile).

Of course, both offices were disconnected immediately. The question was why the central office that had several hundred PCs in addition to the data center became very slow, especially on the Internet.

When I connected Wireshark to the central switch with port mirror to the whole switch (port mirror to VLAN1), I saw the loop. Packets were traveling between the servers and firewall with TTL decrement in every packet. This is a loop.

What happened?

1. The moment the two lines were disconnected, EIGRP in router RDC stopped seeing R1 and R2.
2. When a server sends a packet to networks 10.10.0.0/16 or 10.20.0.0/16, the server sends it to its default gateway; that is, RDC.
3. When the packet arrived to it, RDC sent it to the firewall. This is the route to 0.0.0.0 that takes place if EIGRP becomes inactive.

4. The firewall gets the packet and sends it back to RDC. This is what he has in his routing table.
5. All packets from servers in the data center that are sent to the remote locations start to ping-pong between the servers and the firewall, and that is enough traffic to slow down the servers and access to the Internet.

Finding duplicate IPs

One of the most annoying problems in IP networks is duplicate IP addresses. The funny thing is that if you are familiar with the problem, what causes it, and how to find it, it becomes one of the most simple ones to solve.

Getting ready

When you suspect a duplicate address in the network, the first thing to do will be to use the simple CLI commands—ARP and Ping. If you don't locate the problem, connect Wireshark to the switch and in a large network to every VLAN in the network and move step-by-step until you find the problem.

How to do it...

We start with the phenomena, such as slow access to a server or to another device, slow access to the Internet, and all the pings that you don't get replies to.

1. When you get slow access to a network device, one of the problems that might arise is that the IP address of this device collides with another address. To verify this, ping the IP address.

   Tip

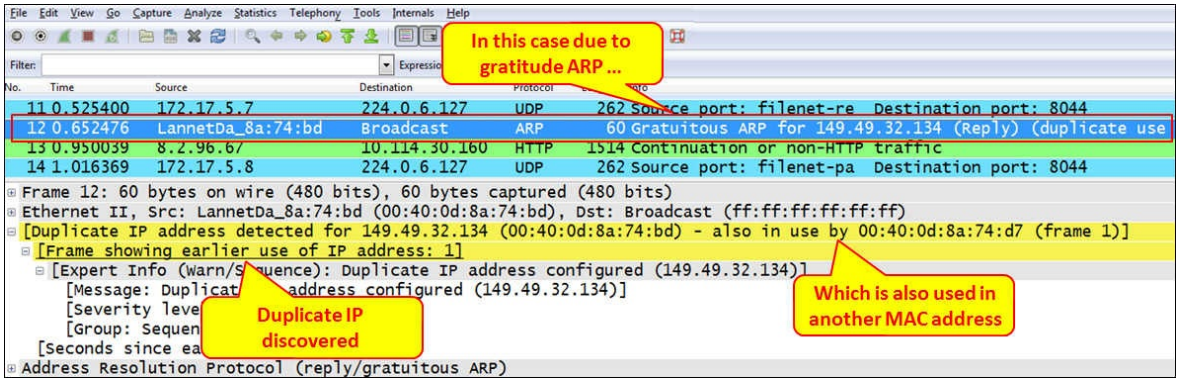
   In some devices, when their address collides with an identical address, the driver will simply be turned off (the little symbol at the bottom-left corner of the screen in the Windows operating system). In other devices, you will not get any notification for a conflict, and this is the place where problems will arise.

2. Type **arp –a** in the **Command Line Interface** (**CLI)**. Use the command `cmd` in Windows (or any shell in Linux). If you get two lines for the IP address you've pinged with different MAC addresses, this is a duplicate.
3. Google the MAC addresses of the two devices, and the first part of the address will tell you who the vendor is. This will lead you to the trouble maker.
4. If you need the location of the device, log in to your LAN switch (when you have a managed switch, of course), and from the switch MAC address table, you will see the switch port that you are connected to.

   Tip

   There is a software that shows you the list of devices that are connected to every switch along with their MAC address, IP address, DNS names, and more. Google for switch port mapper or switch port mapping tools and you will find lots of them.

5. If you don't get anything with Ping and ARP, simply start Wireshark and port mirror the network VLANs. Wireshark will show you a duplicate address error with the relevant details.
6. The error message that you will get will be as shown in the following screenshot:


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
In this case due to gratitude ARP ...
Filter:
No. Time Source Destination Protocol Info
11 0.525400 172.17.5.7 224.0.6.127 UDP 262 Source port: filenet-re Destination port: 8044
12 0.652476 LannetDa_8a:74:bd Broadcast ARP 60 Gratuitous ARP for 149.49.32.134 (Reply) (duplicate use
13 0.950039 8.2.96.67 10.114.30.160 HTTP 1514 Continuation or non-HTTP traffic
14 1.016369 172.17.5.8 224.0.6.127 UDP 262 Source port: filenet-pa Destination port: 8044
Frame 12: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Ethernet II, Src: LannetDa_8a:74:bd (00:40:0d:8a:74:bd), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
[Duplicate IP address detected for 149.49.32.134 (00:40:0d:8a:74:bd) - also in use by 00:40:0d:8a:74:d7 (frame 1)]
[Frame showing earlier use of IP address: 1]
[Expert Info (Warn/Sequence): Duplicate IP address configured (149.49.32.134)]
[Message: Duplicat address configured (149.49.32.134)]
[Severity leve
[Group: Sequen
[Seconds since ea
Address Resolution Protocol (reply/gratuitous ARP)
Duplicate IP discovered
Which is also used in another MAC address

How it works...

When you ping an IP address that appears twice on your local network, the two devices (or more) that have the same IP address will answer to the ARP request that you sent, and your ARP cache will have two entries for the same IP address.

In many cases, your device will indicate it by closing its IP driver and notify you by a pop-up window or any other type of notification that you will be aware of.

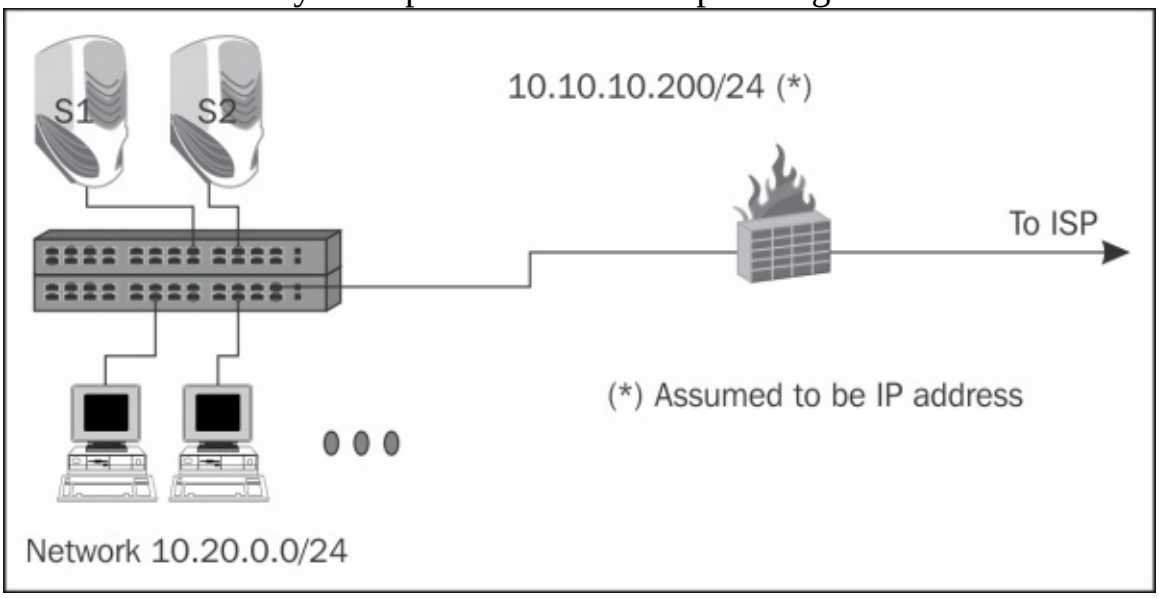
In other cases, the colliding devices will not notify the conflict, and then you will find a problem only with Ping and ARP, as described before.

In any case, when you connect Wireshark to the network and see duplicate IP messages, don't ignore it.

There's more...

Duplicate IP usually happens when there are two identical addresses in the network, but it becomes even more interesting when you have three identical addresses.

You can see a funny example for this in the upcoming screenshot:



In this customer network, they've internal network of around 150 devices with connectivity to the Internet through a firewall. The problem was a very slow connection to the Internet.

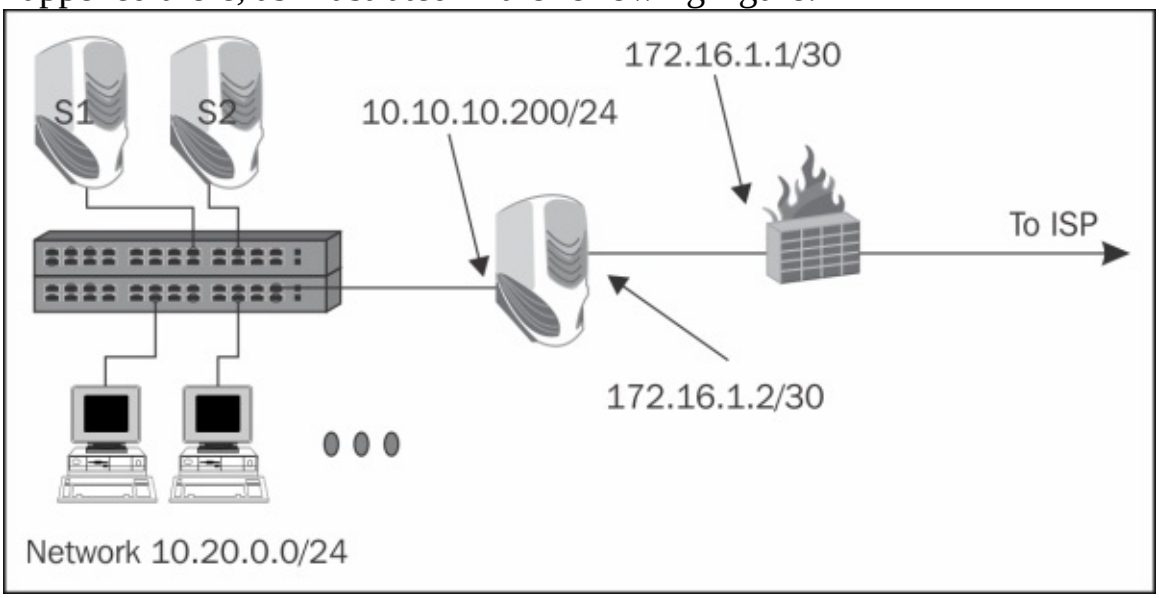
When they did a ping to a server on the Internet (any server), they got the following responses:

```
Reply from 173.194.35.148: bytes=32 time=98ms TTL=51
Request timed out.
Reply from 173.194.35.148: bytes=32 time=124ms TTL=51
Request timed out.
Reply from 173.194.35.148: bytes=32 time=134ms TTL=51
Request timed out.
Reply from 173.194.35.148: bytes=32 time=582ms TTL=51
Request timed out.
```

The customer made some changes to the network, the network became even slower, and pinging the same server on the Internet got them the following response:

```
Reply from 173.194.35.148: bytes=32 time=98ms TTL=51
Request timed out.
Request timed out.
Reply from 173.194.35.148: bytes=32 time=124ms TTL=51
Request timed out.
Request timed out.
Reply from 173.194.35.148: bytes=32 time=134ms TTL=51
Request timed out
Request timed out….
```

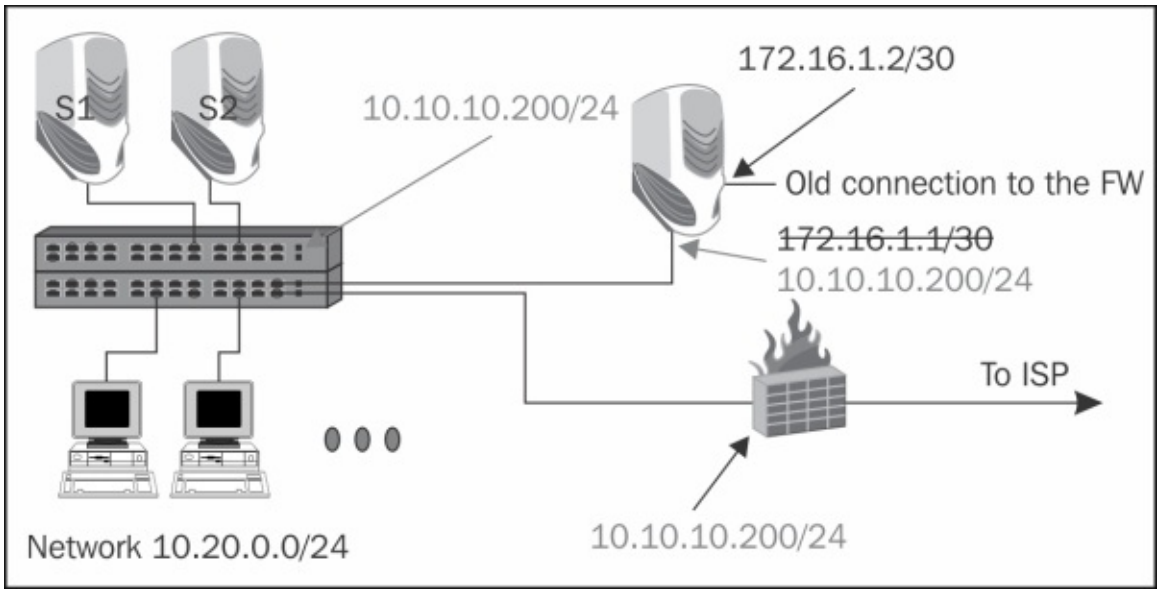
When I came into the picture, the first thing I did was to ping the server on the Internet and type `ARP –a` to see what I got. And what I saw was the IP address 10.10.10.200 with three different MAC addresses. Of course, it was a three-time duplicate address, and digging into the problem showed me what actually happened there, as illustrated in the following figure:



What happened was that the network default gateway to the Internet was not actually the firewall, but a web-filtering device that was located between the network and the firewall with the address 10.10.10.200, while the network between it and the firewall was **172.16.1.2/30**.

What actually happened is explained as follows:

1. In the first place, they configured the DHCP server on the network to exclude addresses 10.10.10.201-254, so the FW address was not excluded.
2. Then they connected a new LAN switch to the stack. The LAN switch was configured by default to receive the IP address by DHCP, so it received the address 10.10.10.200 and that was the first duplicate.
3. And the funniest thing was that the customer suspected a problem of connecting to the Internet, so they disconnected the web-filter server. The stupid problem was that they disconnected the external interface of the web-filter server and connected the internal interface to the switch while changing its address to the address of the firewall that was still connected to the network.
4. What they got is presented in the following illustration, that is, triple 10.10.10.200 addresses.



The conclusion from this case and from many other cases I've experienced is that one of the most important conclusions, is: *Always have an updated drawing of your network!!!*

Analyzing DHCP problems

Dynamic Host Configuration Protocol (**DHCP**) is the protocol that provides you with an IP address automatically while connecting to the network. In this recipe, we will learn how to locate some of the common DHCP problems.

Getting ready

When you have a DHCP server on your network, and PCs are not able to receive IP addresses automatically, just connect Wireshark with port mirror to the device that doesn't receive the address, connect and disconnect the device from the network, or simply use the `ipconfig /release` and `ipconfig /renew` commands. Now, we will have a look at what can go wrong.

How to do it...

Have a look at the DHCP procedure described in the *How it works …* section. Anything that is not going according to this procedure is wrong, so check for the following:

1. Did the client send the `DHCP Discover` packet?
2. If it did, the client works fine.
3. If it didn't:
    - Something is wrong with the client. Check if the client is configured with DHCP (obtain an IP address automatically as marked in the TCP/IP configuration window).
    - It can be that the client is physically not connected to the network. It happens a lot with wireless communications (WiFi), where the client does not have connectivity to the network and therefore, does not send the `DHCP Discover` packet since it doesn't have a network to send it over.

- The client sends `DHCP Discover` and receives `DHCP Offer` from a single server. This is ok; continue watching the wire.
- The client sends `DHCP Discover` and receives `DHCP Offer` from two or more servers. This is a problem. You have more than one DHCP server on your LAN, and you might get different address allocations to clients on the LAN. Turn off one of the servers (at least the DHCP service on it).
- You receive DHCP Discover and send DHCP Request; this is fine.
    - If you immediately receive `DHCP Ack` with the IP parameters, everything is fine.
    - If you don't receive anything, and you send another `DHCP Request`, it can be a slow or non-responsive server. Check it.
- If you receive a DHCP Decline message, it is the server that has refused your request.

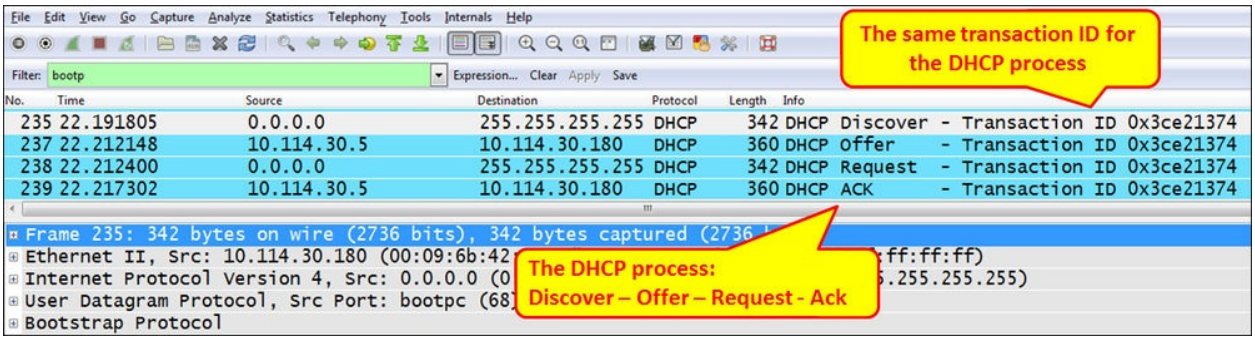
1. It can be that the server does not have available addresses. In this case, extend your address range.
2. It can be also that the server has allocated your previous IP address to someone else. This is a server configuration issue; so if you need this feature, configure the server to save IP addresses per clients.

How it works...

DHCP is considered to be a simple protocol, but actually it is very complex. When you connect a client to the network, it will go through the following steps:

1. `DHCP Discover`: The client initializes a limited version of TCP/IP and broadcastsa request looking for a DHCP server. The request is sent from UDP port 68 to UDP port 67.
2. `DHCP Offer`: DHCP servers listen on UDP port 67, and if a server receives the request, it answers with a DHCP offer, that is offering to provide the service of address assignment.
3. `DHCP Request`: The client receives the DHCP offer and sends back a request to receive information. The request will be, for example, the IP address that we requested before (because we had it before), for our MAC address so that the server will recognize us as a prior client with a saved IP address and other parameters.
4. `DHCP Ack`: Here the server sends the requested information, including the IP address, subnet mask, default gateway, DNS servers, and other parameters that are configured on the server.

In the next screenshot, we see a standard procedure of DHCP that works properly:



There's more...

A very common problem is when you connect a device to your network, you receive an IP address and you don't have any idea where it came from. Usually, this is because someone has connected a DHCP server to your LAN without telling you. In most of the cases, it will be a small Internet router. This is very simple to find out:

1. If you type `ipconfig` and get an address that you don't know, it might be a problem.
2. Since the router we suspect is connected to the network, assign your IP address, subnet mask, and a default gateway. When you ping your default gateway, you actually ping the router, which is likely to be the troublemaker.
3. Type `ARP -a` to give you the troublemaker's MAC address. This will tell you two things:
    - Who is the vendor? When you know who is the vendor is (D-Link, Edimax, Netgear, and many others), you can simply go and look for it.
    - By logging into the LAN switch, the MAC address will also tell you which port it is connected to. Go to your communications room and disconnect it.

- Of course, while listening to the port with Wireshark, you will see the vendor MAC address easily.

Chapter 9. UDP/TCP Analysis

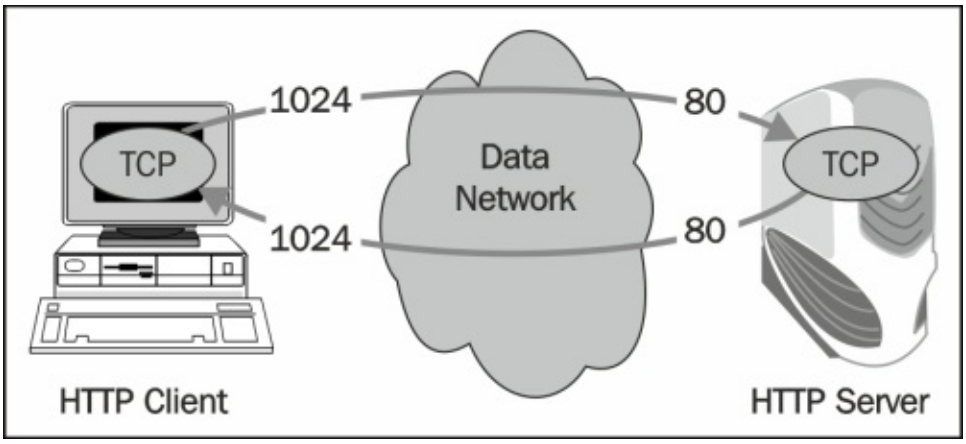
This chapter contains the following recipes:

- Configuring TCP and UDP preferences for troubleshooting
- TCP connection problems
- TCP retransmissions – where they come from and why
- Duplicate ACKs and fast retransmissions
- TCP out-of-order packet events
- TCP Zero Window, Window Full, Window Change, and other Window indicators
- TCP resets and why they happen

Introduction

The goal of **Transmission Control Protocol** (**TCP**) and **User Datagram Protocol** (**UDP**) is to pass information between end applications, for example, from a web client to a web server, mail client to a mail server, and so on. This is done by providing identification to end applications and forwarding packets between them. These identifications are called port numbers, and a port number with its IP address is called a socket. In the following diagram you can see what happens when you open a connection from your browser to a web server. The web server listens on port 80 and you will open a connection, for example, from port 1024.

So, the server is listening to requests on port 80 and will send responses to you on port 1024.



While TCP is a reliable, connection-oriented protocol, UDP does not support connectivity and reliability, but simply transfers datagrams between two end processes.

Tip

There is an additional layer-4 protocol, which is called SCTP (Stream Control Transmission Protocol). This protocol can be considered as an improved version of TCP, and mostly used in a service provider's networks. SCTP is not included in the scope of this book.

In this chapter, we will focus on TCP, its behavior, various problems, and how to use Wireshark in order to isolate and solve them.

Configuring TCP and UDP preferences for troubleshooting

In most cases you can use the default Wireshark parameters for TCP and UDP network analysis, but there are also some changes that can be configured. The changes will be configured in the **Preferences** window.

Getting ready

For TCP or UDP configuration:

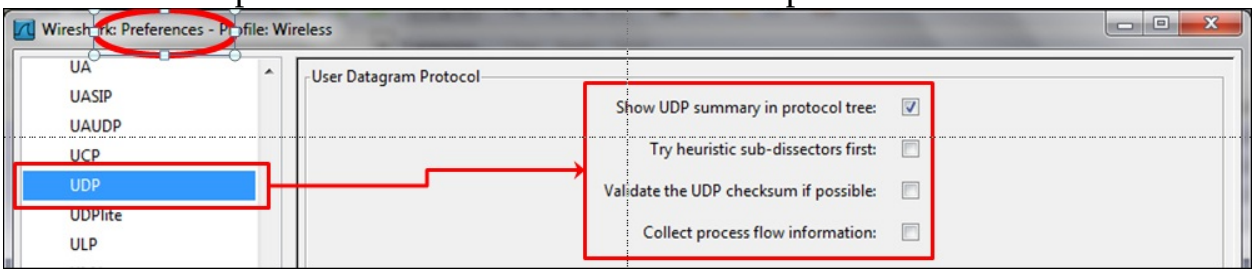
1. Start Wireshark, and from the **Edit** menu, choose **Statistics**.
2. Under **Protocols**, choose **TCP** or **UDP**.

How to do it...

In this section we will see how to configure TCP and UDP preferences.

UDP parameters

Let's see some parameters that can influence the capture of UDP:

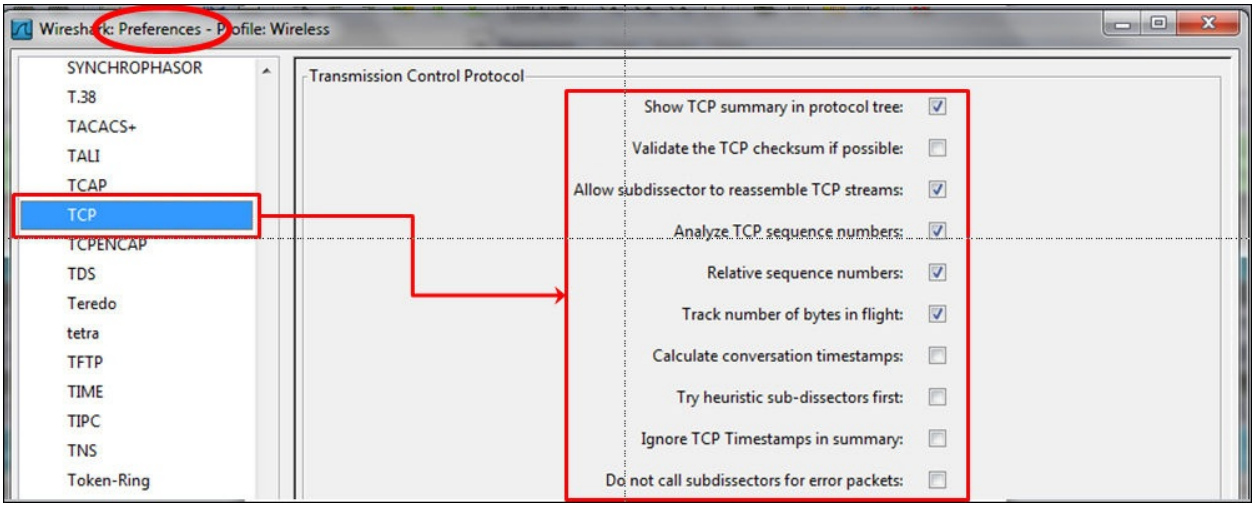


You can configure the following parameters in UDP:

- **Show UDP summary in protocol tree**: Mark this button if you want the UDP summary line to be shown in the protocol tree (set by default)
- **Try heuristic sub-dissectors first**: Try to decode a packet using the heuristic method before using a sub-dissector registered to the specific port
- **Validate the UDP checksum if possible**: Validates the UDP checksum
- **Collect process flow information**: Collects process flow information

By default only the first parameter is set. In most cases it is enough.

TCP parameters



You can configure the following parameters in TCP:

- **Show TCP summary in protocol tree**: Mark this button if you want the TCP summary line to be shown in the protocol tree (set by default).
- **Validate the TCP checksum if possible**: This feature can slow down performance. In most cases it is not required.
- **Allow subdissector to reassemble TCP streams**: This option is for stream analysis (set by default).
- **Analyze TCP sequence numbers**: When this is set, Wireshark analyzes sequence numbers and track phenomena such as retransmission, duplicate ACKs, and so on, which is one of the important features of Wireshark.
- **Relative sequence numbers**: When this is set, Wireshark will show you every TCP connection that starts from `Seq=0`.
- **Track number of bytes in flight**: This setting enables Wireshark to track the number of unacknowledged bytes flowing on the network (set by default).
- **Calculate conversation timestamps**: This feature enables the calculations of TCP timestamps option.
- **Try heuristic sub-dissectors first**: Try to decode a packet using heuristic method before using a sub-dissector registered to the specific port.
- **Ignore TCP Timestamps in summary**: Ignore the timestamp option in the TCP header.
- **Do not call subdissector for error packets**: This option does not analyze

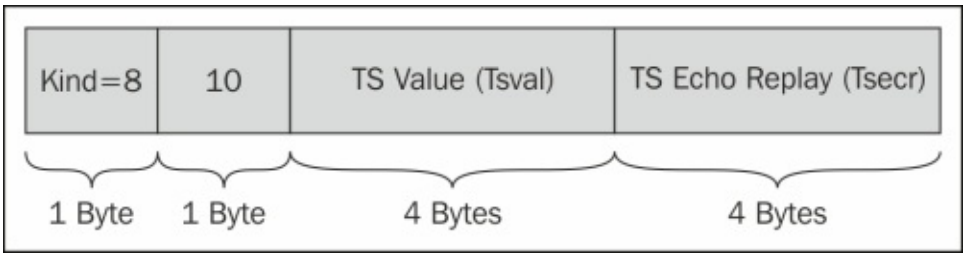
erroneous TCP packets.

How it works...

There are some parameters in the TCP preferences that I would like to say a few words about.

Referring to **relative sequence numbers**, when you look at a TCP connection you see that it always starts with sequence numbers equal to zero. These are the relative numbers that are normalized to zero by Wireshark. The real numbers are numbers between 0 and 232, picked by the TCP process, which are difficult to follow. The TCP standard does not set any rule for picking this number.

The **calculating conversations timestamps** refers to the timestamp option of the TCP packet. The TCP timestamps option carries two 4-byte timestamp fields, as seen in the next diagram:



The problem that the timestamp option comes to solve is the sensitivity of TCP to delay variations. The solution, and written in RFC 1323, is to use TCP options in the following ways (for every TCP connection):

- The sender places a timestamp in each data segment that it sends (the Tsval field)
- The receiver reflects these timestamps in ACK segments (the Tsecr field)

Then, a single subtraction gives the sender an accurate RTT measurement for every ACK segment (which will correspond to every other data segment, with a sensible receiver). This mechanism is called **Round Trip Time Measurement** (**RTTM**).

There's more…

UDP is a very simple protocol with a very simple header that includes only four fields: source port, destination port, packet length, and checksum. Checksum is used by the receiver to check whether to accept the packet or drop it. In case of a packet drop, there is no recovery mechanism. In some cases, the application will recover it (for example, DNS that sends the request again), and in some cases it won't.

TCP is more sophisticated. It is a connection-oriented, reliable protocol, with sequencing mechanism, flow, and congestion control. These subjects will be discussed later in this chapter.

SCTP is a reliable, connection-oriented protocol that allows the transfer of multiple streams per connection, optional bundling of multiple user messages into a single SCTP packet, support for cookies, multi-homing, and other mechanisms. It was initially developed for carrying signaling messages in cellular networks, and later implemented with other application protocols.

TCP connection problems

When two TCP processes wish to communicate, they open the connection, send the data, and then close the connection. This happens when you open a browser to the Internet, connect from your mail client to the mail server, or connect with Telnet to your router or any other application that works over TCP.

When TCP opens the connection, it sends a request for open connection from the source port to destination port.

Some problems can occur during the establishment or closing of the application. Using Wireshark to locate and solve these problems is the goal of this recipe.

Getting ready

If you experience one of the following problems, use Wireshark in order to find out what is the reason for it.

These problems can be of many types. Of these:

- You try to run an application and it does not work. You try to browse the Internet and you don't get any response.
- You try to use your mail but you don't have a connection to the mail server.
- Problems can be due to simple reasons, such as the server being down, the application is not running on the server, or the network is down somewhere on the way to the server.
- Problems can be also due to more complicated reasons, such as DNS problems, insufficient memory on the server that does not enable you to connect (due to high memory consumption by an application, for example), duplicate IPs, and many others.

In this recipe we focus on these GO/NO-GO problems that are usually quite easy to solve.

How to do it...

Here you will see some indicators and what you can see when you use Wireshark for debugging TCP connectivity problems. Usually these problems result in trying to run an application and getting no results.

When you try to run an application, for example, a database client, a mail client, watching cameras servers, and so on, and you don't get any output, follow these steps:

1. Verify that the server and applications are running.
2. Verify if your client is running, you have an IP address configured (manually or by DHCP), and you are connected to the network.
3. Ping the server and verify you have connectivity to it.

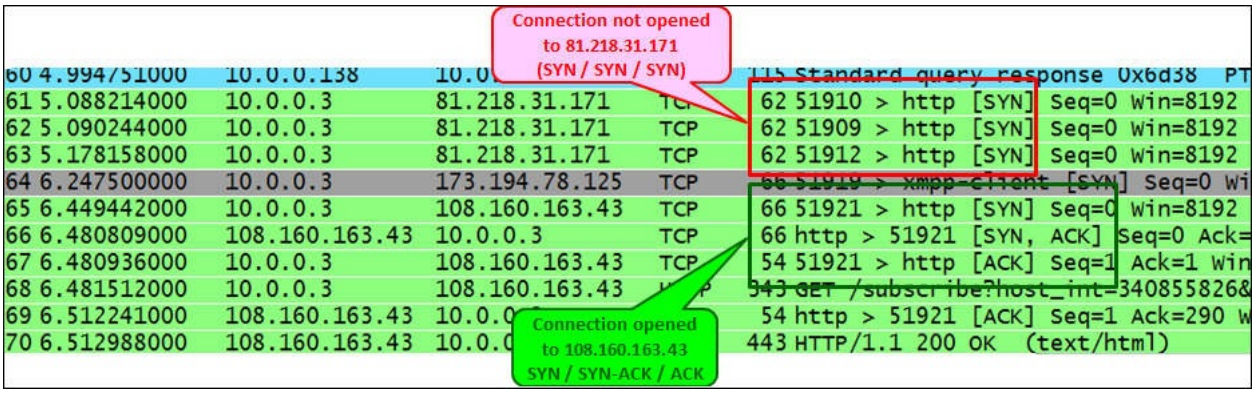
   Tip

   In some cases, you will not have Ping to the server, but still have connectivity to the application. This can happen because a firewall is blocking the ICMP messages, so if you don't have Ping to a destination it doesn't necessarily mean that something is wrong. The firewall can be a dedicated device in the network or a windows (or Linux/UNIX) firewall installed on the end device.

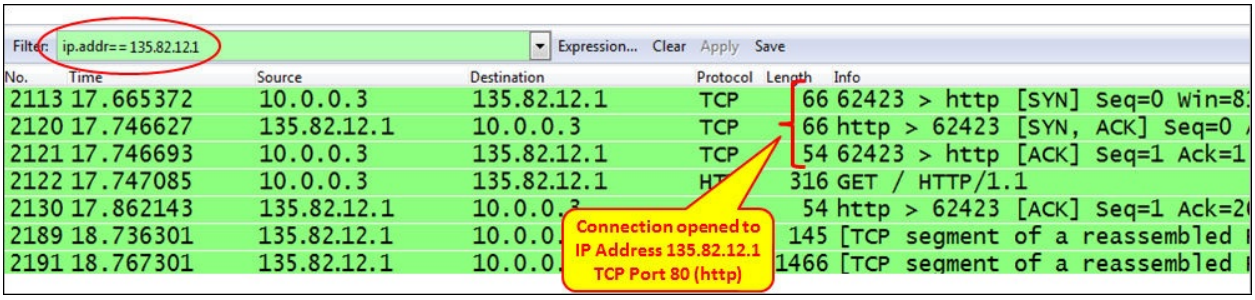
4. In the capture file, look for one of the following patterns:
   - Triple SYN messages with no response (in the following screenshot)
   - SYN messages with a reset (RST) response

In both cases it can be that a firewall is blocking the specific application or the application is not running.

In the following screenshot, we see a simple case in which we simply don't get access to web server `81.218.31.171` (packets `61`, `62`, and `63`). It can be because it is not permitted by a firewall, or simply because there is a problem with the server. We can also see that we have a connection to another website (`108.160.163.43`, packets `65`, `66`, and `67`), so the connection problem is only to `81.218.31.171`.



In the next screenshot we see a slightly more complex case of the same situation. In this case, we've had a cameras server that the customer wanted to log in to and watch the cameras on a remote site. The camera's server had the IP address `135.82.12.1` and the problem was that the customer was able to get the main web page of the server with the login window, but couldn't log into the system. In the following screenshot, we can see that we open a connection to the IP address `135.82.12.1`. We see that a TCP connection is opened to the HTTP server, and at first it looks like there are no connectivity problems:



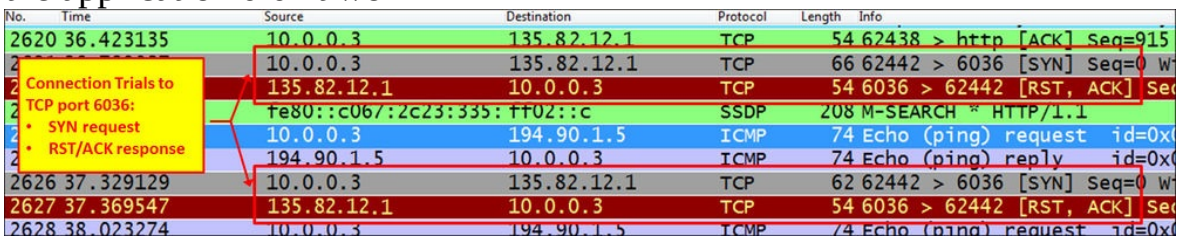
The problems arise when we filter all traffic to the IP address `135.82.12.1`, that is, the cameras server.

Here we see that when we try to connect to TCP port `6036`, we get an RST/ACK response, which can be:

- A firewall that blocks port `6036` (that was the case here)
- When port address translation (PAT) is configured, and we translate only

port `80` and not `6036`

- The authentication of the username and password were done on TCP port `6036`, the firewall allowed only port `80`, the authentication was blocked, and the application didn't work



No.	Time	Source	Destination	Protocol	Length	Info
2620	36.423135	10.0.0.3	135.82.12.1	TCP	54	62438 > http [ACK] Seq=915
		10.0.0.3	135.82.12.1	TCP	66	62442 > 6036 [SYN] Seq=0 W
		135.82.12.1	10.0.0.3	TCP	54	6036 > 62442 [RST, ACK] Se
		fe80::c067:2c23:335:	ff02::c	SSDP	208	M-SEARCH * HTTP/1.1
		10.0.0.3	194.90.1.5	ICMP	74	Echo (ping) request id=0x
		194.90.1.5	10.0.0.3	ICMP	74	Echo (ping) reply id=0x
2626	37.329129	10.0.0.3	135.82.12.1	TCP	62	62442 > 6036 [SYN] Seq=0 W
2627	37.369547	135.82.12.1	10.0.0.3	TCP	54	6036 > 62442 [RST, ACK] Se
2628	38.023274	10.0.0.3	194.90.1.5	ICMP	74	Echo (ping) request id=0x

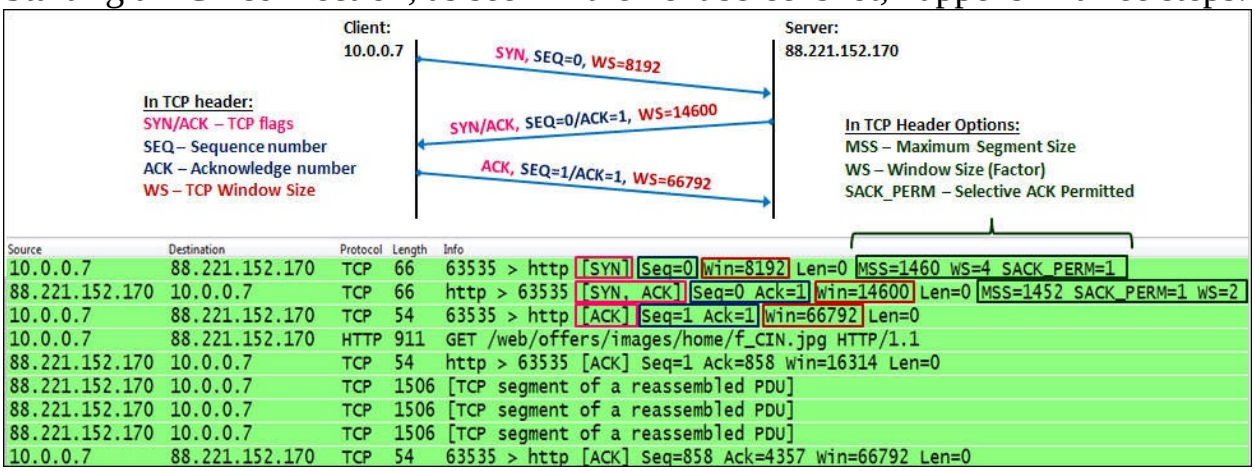
To summarize, when you don't have connectivity to a server, check the server and the client if all TCP/UDP ports are forwarded throughout the network, and if you have any ports that you don't know about.

Tip

In some cases when you install new applications in your network, it is good to connect Wireshark on the client and the server, and check what is actually running between them. The software house will not always tell you what they are actually transferring over the network (sometimes this is because they are not aware of it!), and firewalls can block information that you are not aware of.

How it works...

Starting a TCP connection, as seen in the next screenshot, happens in three steps:



1. The TCP process on the client side sends an SYN packet. This is a packet with the SYN flag set to `1`. In this packet the client:
    - Specifies its initial sequence number. This is the number of the first byte that the client sends to the server.
    - Specifies its window size. This is the buffer the clients allocate to the process (the place in the client's RAM).
    - Sets the options that will be used by it: MSS, Selective ACK, and so on.

- When the server receives the request to establish a connection, the server:
    - Sends an SYN/ACK packet to the client, confirming the acceptance of the SYN request.
    - Specifies the server's initial sequence number. This is the number of the first byte that the server sends to the client.
    - Specifies the server's window size. This is the buffer size that the server allocates to the process (the place in the server's RAM).
    - Responds to the options requested and sets the options on the server side.
- When receiving the server's SYN/ACK, the client:
    - Sends an ACK packet to the server, confirming the acceptance of the SYN/ACK packet from the server.
    - Specifies the client's window size. This is the buffer size that the client allocates to the process. Although this parameter was defined in the first

packet (the SYN packet), the server will refer to this one since it is the latest window size received by the server.

In the options field of the TCP header, we have the following main options:

- **Maximum Segment Size (MSS)**: This is the maximum size of the TCP datagram, that is, the number of bytes from the beginning of the TCP header to the end of the entire packet.
- **Windows Size (WSopt)**: This factor is multiplied with the Window Size field in the TCP header to notify the receiver on a larger size buffer. Since the maximum window size in the header is 64 KB, a factor of 4 gives us 64 KB multiplied by 4, that is, a 256 KB window size.
- **SACK**: Selective ACK is an option that enables the two parties of a connection to acknowledge specific packets, so when a single packet is lost, only this packet will be sent again. Both parties of the connection have to agree on SACK in the connection establishment.
- **Timestamps options (TSopt)**: This parameter was explained earlier in this chapter, and refers to measurement of the delay between client and the server.

By this stage, both sides:

- Agree to establish a connection
- Know the other side's initial sequence number
- Know the other side's window size

Tip

Anything but a full three-way handshake while establishing a connection should be considered as a problem. This includes SYN without a response, SYN and then SYN/ACK and no last ACK, SYN which is answered with a reset (RST flag equal 1), and so on.

There's more…

Some rules of thumb are as follows:

- In case an SYN packet is answered with RST, look for the firewall that blocks the port numbers.
- Triple SYN without any answer occurs either due to an application that didn't respond, or a firewall that blocks the request on a specific port.
- Always verify if you have Network Address Translation (NAT), port forwarding, and mechanisms that play with TCP or UDP ports. These mechanisms can interrupt with the standard operation of TCP.

TCP retransmission – where do they come from and why

When TCP sends a packet or a group of packets (refer to the *How it works…* section later in this recipe), it waits for acknowledgment to confirm the acceptance of these packets. Retransmissions, obviously, happen due to a packet that has not arrived, or acknowledgment that has not arrived on time. There can be various reasons for this, and finding the reason is the goal of this recipe.

Getting ready

When you see that the network becomes slow, one of the reasons for this can be retransmissions. Connect Wireshark in the port mirror to the suspicious client or server, and watch the results.

In this recipe, we will see some common problems that we encounter with Wireshark, and what they indicate.

How to do it...

Let's get started:

1. Start capturing data on the relevant interface.
2. Go to the **Analyze** | **Expert Info** menu.
3. Under **Notes**, look for **Retransmissions**.
4. You can click on the (+) sign and a list of retransmissions will open. A single mouse click on every line will bring you the retransmission in the packet capture pane.
5. Now comes the important question: how to locate the problem.

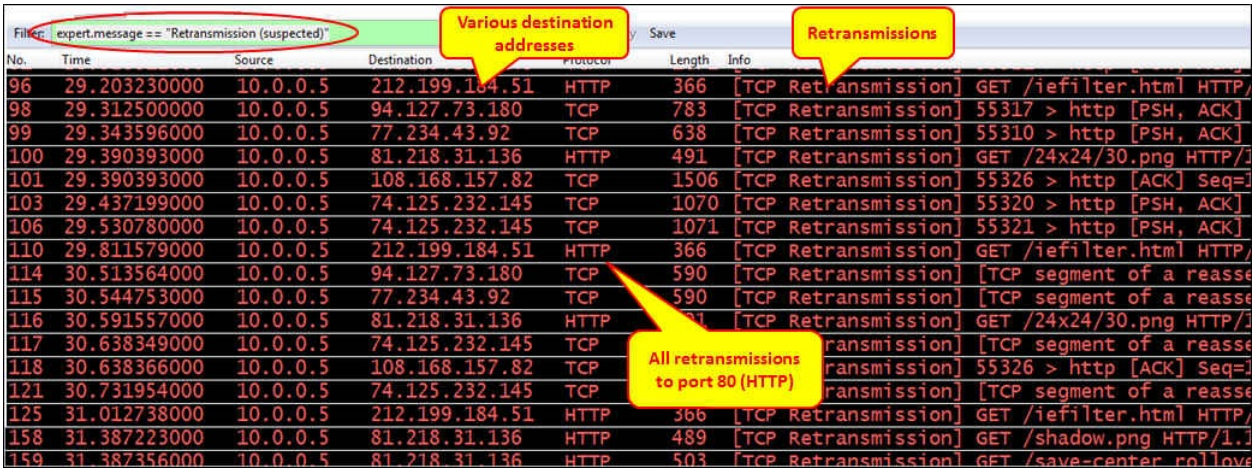
   Tip

   When you capture packets over a communication line, server interface, link to the Internet, or any other line, you can have traffic from many IP addresses, many applications, and even specific procedures on every application, for example, accessing a specific table in a database application. The important thing here is to locate the TCP connections on which the retransmissions happen.

6. You can see where the retransmissions come from by:
   - Moving packet-by-packet in the **Expert Info** window, and looking for what packets does it take you in the packet capture pane (good for experienced users)
   - In the packet pane, configure the display filter `expert.message == "Retransmission (suspected)"`, and you will get all retransmissions in the capture file
   - By applying the filter, and then checking the **Limit to display filter** section to the right-bottom corner of the window in the **Statistics à Conversations** window

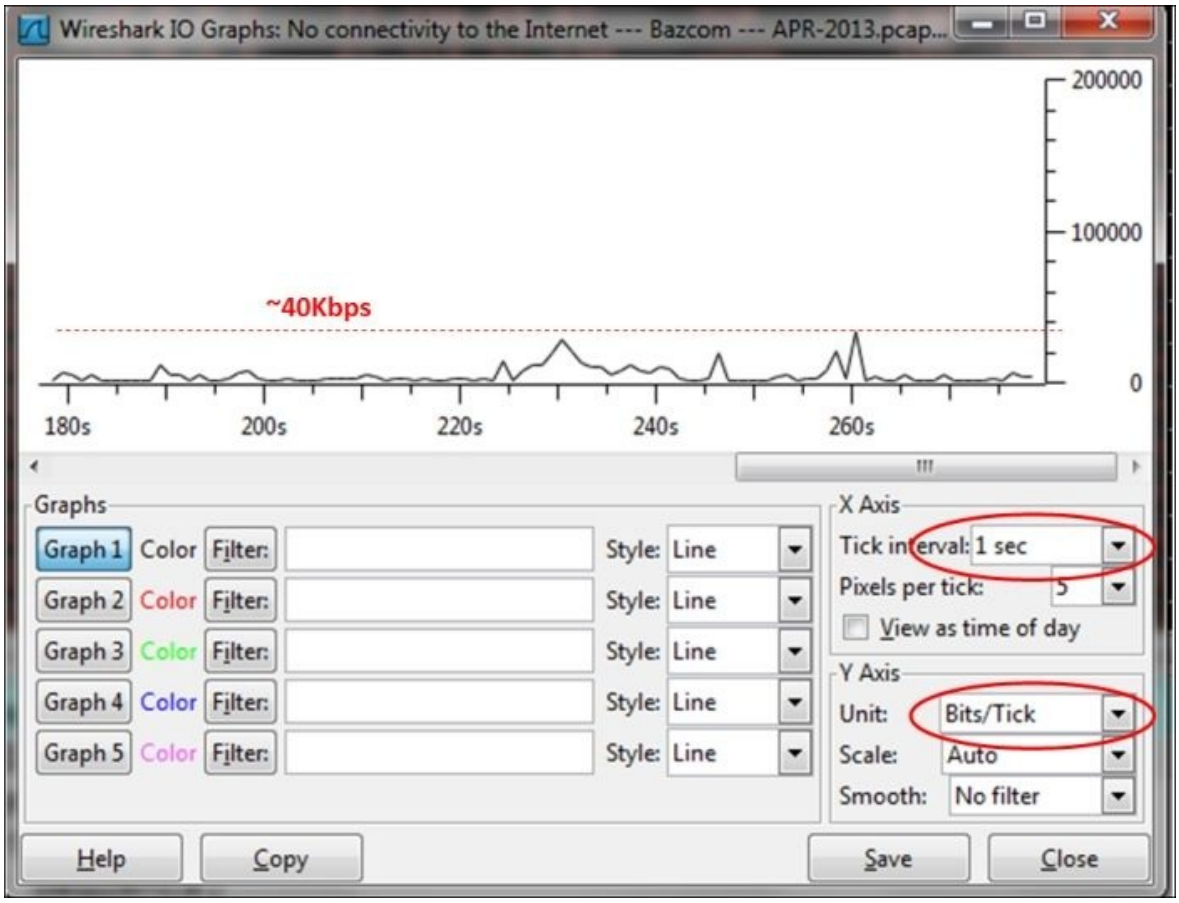
Case 1 – retransmissions to many destinations

In the following screenshot, you see that we've got many retransmissions, spread between many servers, with destination ports `80` (HTTP). What we can also see from here is the `10.0.0.5` port sends the retransmission, so packets were lost on the way to the Internet, or acknowledgement was not sent back on time from the web servers.



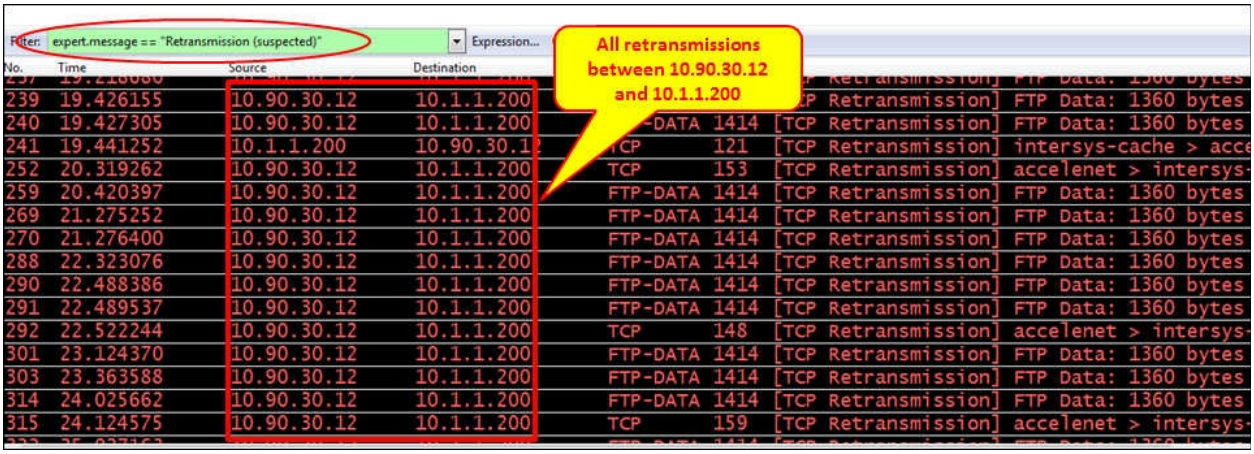
Well, obviously something is wrong on the line to the Internet. How can we know what it is?

1. From the **Statistics** menu, open **IO Graph**.
2. In this case (case 1), we see that the line is nearly empty. Probably it is an error, or another loaded line on the way to the Internet.
3. You can check packet losses and errors that cause them by logging into the communications equipment or by any SNMP browser (when the SNMP agent is configured on the equipment). Check the following screenshot for reference:



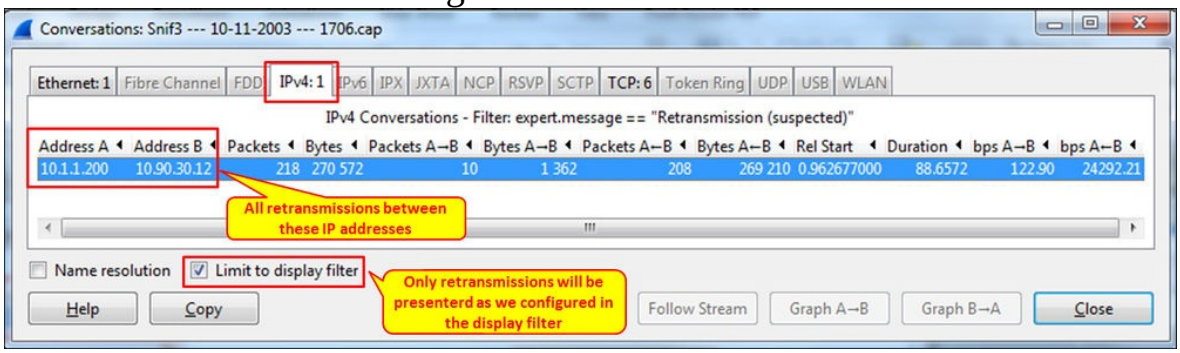
Case 2 – retransmissions on a single connection

If all retransmissions will be on a single IP, with a single TCP port number, it will be a slow application. We can see this in the following screenshot:

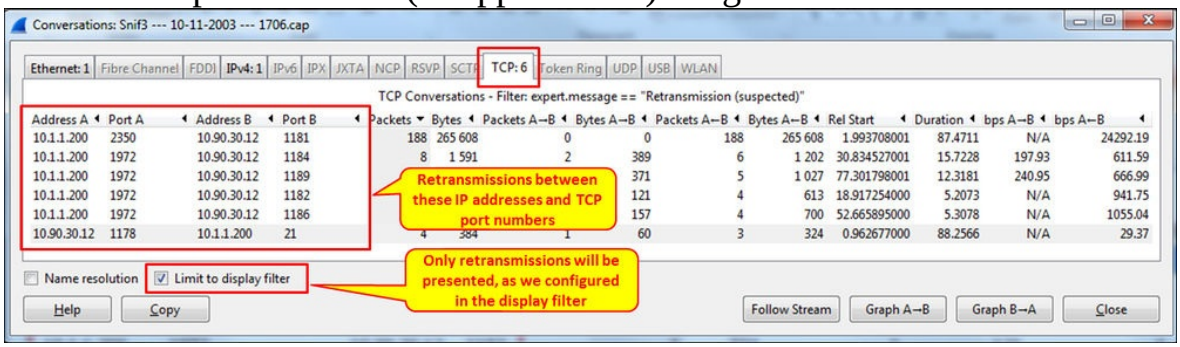


For retransmissions on a single connection, perform the following steps:

1. We can also verify this by opening **Conversations** from the **Statistics** menu, and by selecting the **Limit to display filter** checkbox, we will get all the conversations that have retransmissions, in this case, a single conversation.
2. By choosing the **IPv4** tab as shown in the following screenshot we will see from which IP addresses we get the retransmissions:



3. By choosing the **TCP** tab as shown in the following screenshot we will see from which port numbers (or applications) we get the retransmissions:



To isolate the problem, perform the following steps:

1. Look at the IO graph, and make sure that the line is not busy.

   Tip

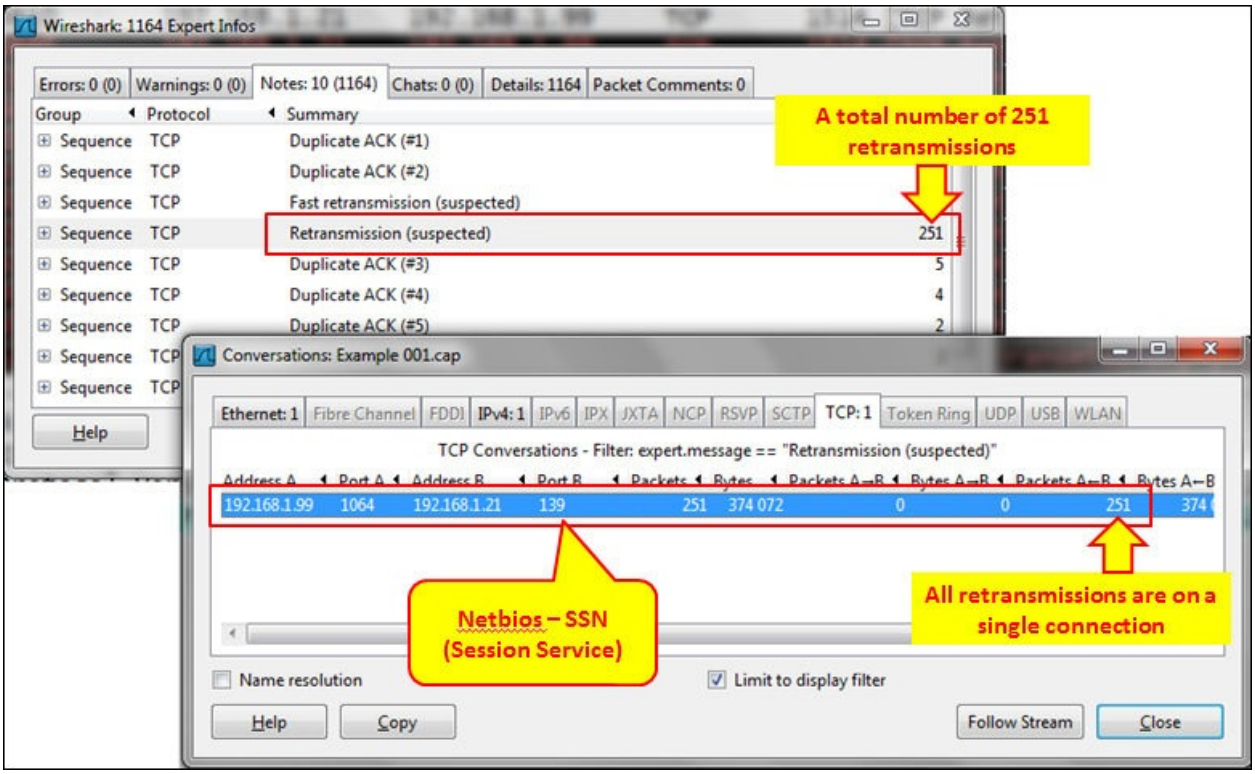
An indication of a busy communication line will be a straight line very close to the maximum bandwidth of the line. For example, if you have a 10 Mbps communication line, you port mirror it, and see in the IO graph a straight line which is close to the 10 Mbps, this is a good indication of a loaded line. A non-busy communication line will have many ups and downs, peaks and empty intervals.

2. If the line is not busy, it can be a problem on the server for the IP address `10.1.1.200` (`10.90.30.12` is sending most of the retransmissions, so it can be that `10.1.1.200` responds slowly).
3. From the packet pane we see that the application is FTP-DATA. It is possible that the FTP server works in an active mode. Hence we've opened a connection on one port (`2350`), and the server changed the port to `1972`, so it can be a slow non-responsive FTP software (that was the problem here eventually).

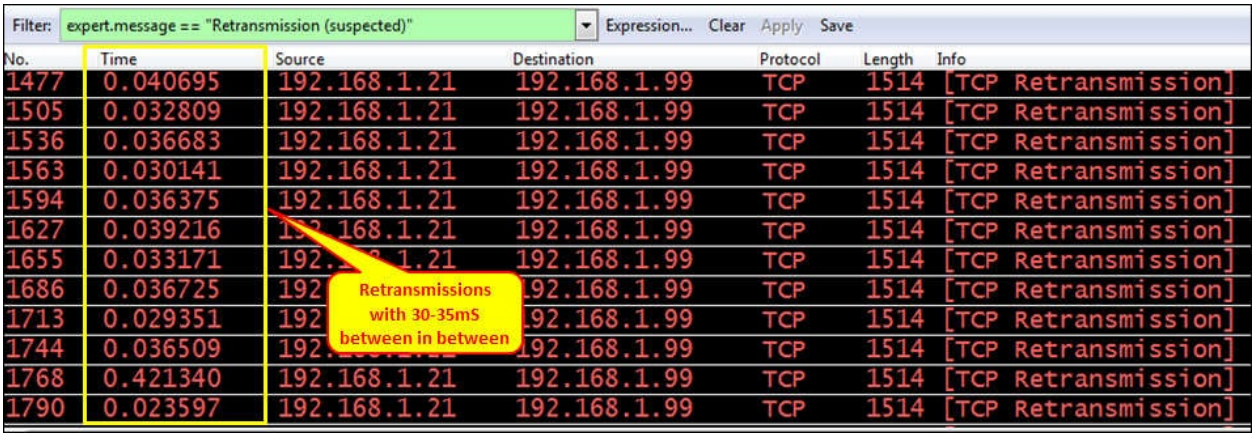
Case 3 – retransmission patterns

An important thing to watch for in TCP retransmissions is if the retransmissions have any pattern that you can see.

In the following screenshot, we see that all retransmissions are coming from a single connection, between a single client and NetBIOS Session Service (TCP port `139`) on the server.



Looks like a simple server/application problem, but when we look at the packet capture pane, we see something interesting (refer to the following screenshot):



The interesting thing is that when we look at the pattern of retransmissions, we see that they occur cyclically every 30 ms. The time format here is seconds,

since the previously displayed packet and the time scale is in seconds.

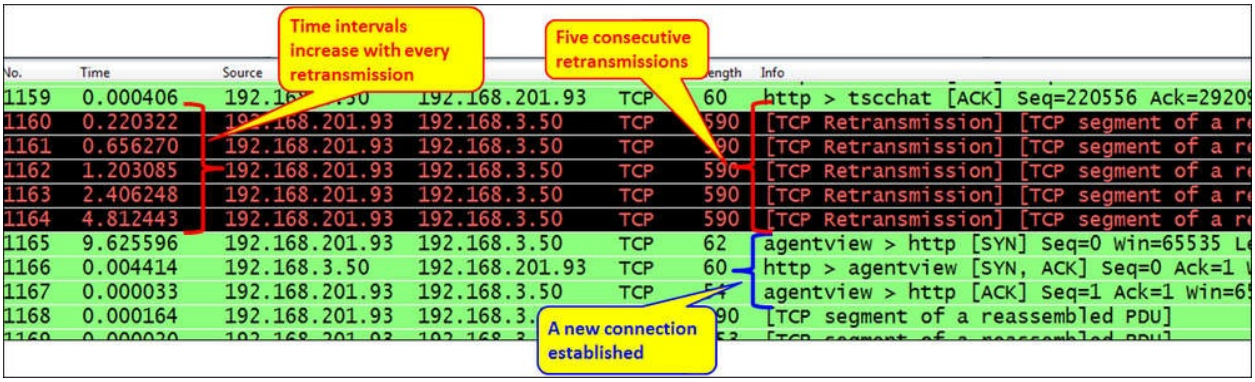
The problem in this case was a client that performed a financial procedure in the software that caused the software to slow down every 30-36 ms.

Case 4 – retransmission due to a non-responsive application

Another reason for retransmissions can be when a client or a server does not answer to requests. In this case, you will see five retransmissions, with an increasing time difference. After these five consecutive retransmissions, the connection is considered to be lost by the sending side (in some cases, reset will be sent to close the connection, depending on the software implementation). After the disconnection, two things may happen:

- An SYN request will be sent by the client, in order to open a new connection. What the user will see in this case is a freeze in the application, and after 15-20 seconds it will start to work again
- No SYN will be sent, and the user will have to run the application (or a specific part of it) again

In the following screenshot we can see a case in which a new connection is opened:



Case 5 – retransmission due to delayed variations

TCP is a protocol that is quite tolerant of delays, as long as the delay does not vary. When you have variations in delay, you can expect retransmissions. The way to find out if this is the problem is as follows:

1. The first thing to do is, of course, to ping the destination, and get the first information of the communications line delay. Look at the *How it works…* section to see how it should be.
2. Check for the delay variations, which can happen due to the following reasons:
    - A delay can happen due to a non-stable or busy communication line. In this case, you will see delay variations using the Ping command. Usually it will happen on lines with a narrow bandwidth, and in some cases on cellular lines.
    - A delay can happen due to a loaded or inefficient application. In this case, you will see many retransmissions on this specific application only.
    - A delay can happen due to a loaded communication equipment (CPU load, buffer load, and so on). You can check this by accessing the communication equipment directly.
3. Use the Wireshark tools as explained in Chapter 13, *Troubleshooting Bandwidth and Delay Problems*.

Tip

The bottom line with TCP retransmissions is that retransmissions are a natural behavior of TCP as long as we don't have too many of them. Degradation in performance will start when the retransmissions are around 0.5 percent, and disconnections will start around 5 percent. It also depends on the application and its sensitivity to retransmissions.

Finding what it is

When you see retransmissions on a communication link (to the Internet, on a server, between sites, or any other link), perform the following:

1. Locate the problem—is it a specific IP address, specific connection, specific application, or some other problem.
2. Check if the problem is because of the communication link, packet loss, or a slow server or PC. Check if the application is slow.
3. If it is not due to any of the preceding reasons, check for delay variations.

How it works...

Let's see the regular operation of TCP, and what are the causes for problems that might happen.

Regular operation of the TCP Sequence/Acknowledge mechanism

One of the mechanisms that is built into TCP is the retransmission mechanism. This mechanism enables the recovery of data that is damaged, lost, duplicated, or delivered out of order.

This is achieved by assigning a sequence number to every transmitted byte, and expecting an acknowledgment (ACK) from the receiving party. If the ACK is not received within a timeout interval, the data is retransmitted.

At the receiver end, the sequence numbers are used in order to verify that the information comes in the order that it was sent. If not, rearrange it to its previous state.

This mechanism works as follows:

1. At the connection establishment, both sides tell each other what will be their initial sequence number.
2. When data is sent, every packet has a sequence number. The sequence number indicates the number of the first byte in the TCP payload. The next packet that is sent will have the sequence number of the previous one, plus the number of bytes in the previous packet, plus one (in the next screenshot).
3. When a packet is sent, the RTO (Retransmission Timeout) counter starts to count the time from the moment it was sent.

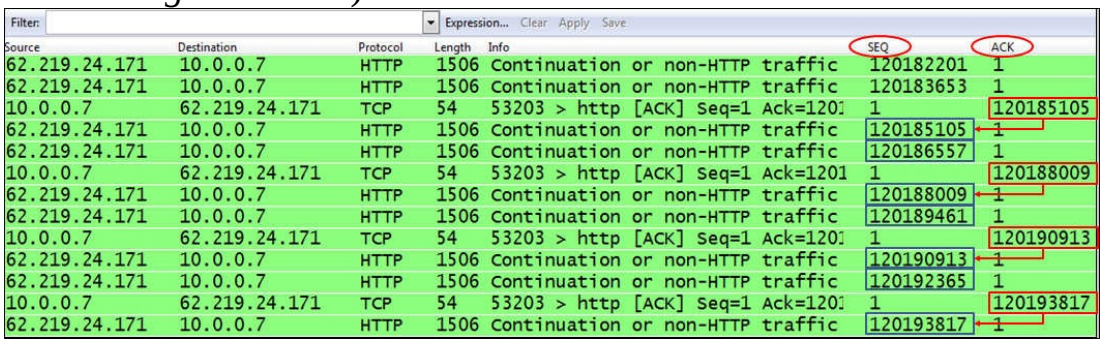
   ### Tip

   The Retransmission Timeout timer is based on the Van Jacobson congestion avoidance and control algorithm, which basically says the TCP is tolerant to high delays, but not to fast delay variations.

4. When the receiver receives the packet, it answers with an ACK (Acknowledge) packet that tells the sender to send the next packet. In the

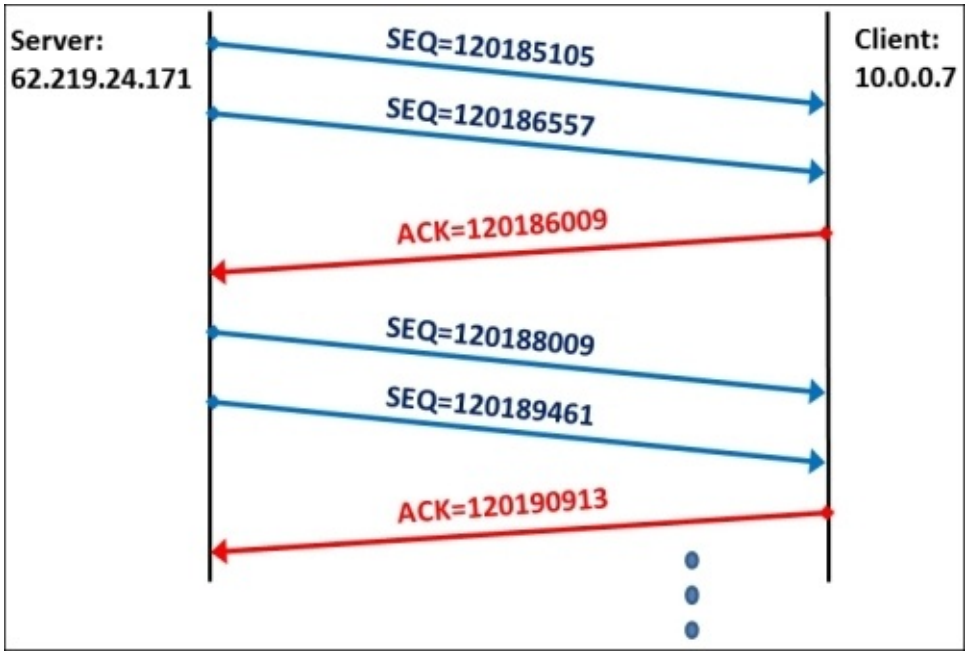
following screenshot you will see how it works:

1. You can see from here that `10.0.0.7` is downloading a file from `62.219.24.171`. The file is downloaded via HTTP (the Wireshark window was configured to show `tcp.seq` and `tcp.ack` from the **Edit** | **Preferences** columns configuration, as described in Chapter 1, *Introducing Wireshark*).



2. You can see from here that `62.219.24.171` sends a packet with a sequence number of `120185105`, and then a packet with the sequence number `120186557`. When receiving these two packets, the client `10.0.0.7` tells the server to send him the next packet with ACK = `120188009`, after which the server sends the packet with the sequence number `120188009`, and the next packet with sequence number `120189461`, and so on.

   You can see a diagram for this.

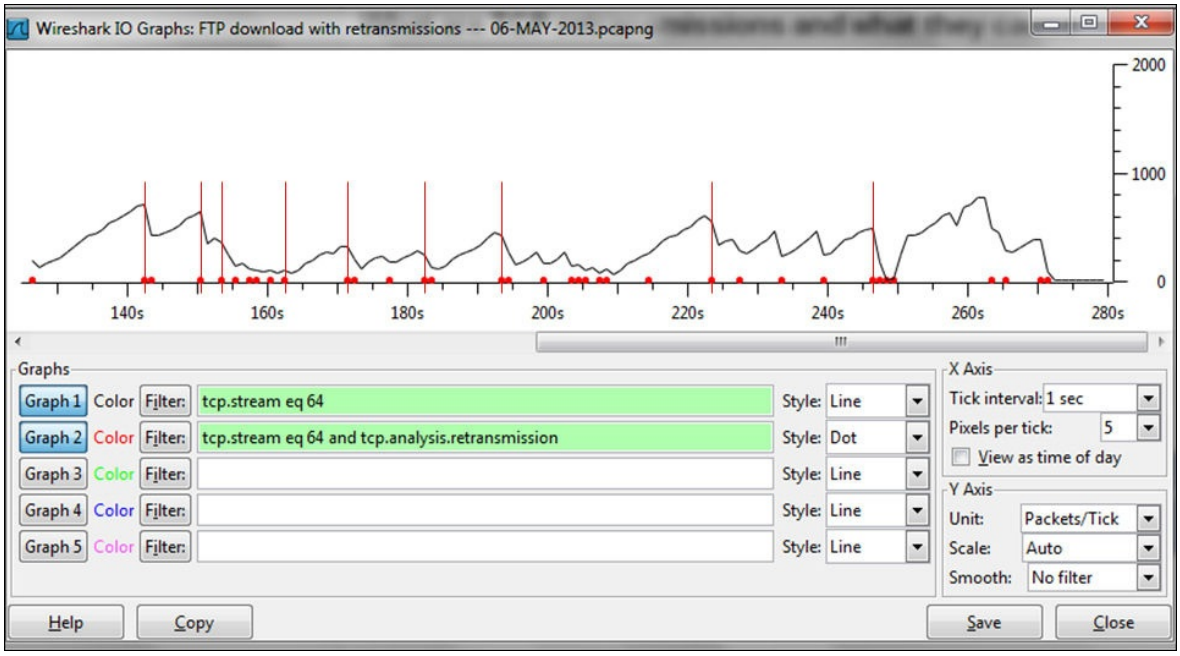


What are TCP retransmissions and what do they cause

When a packet acknowledgment is lost, or when an ACK does not arrive on time, the sender will perform two things:

1. Send the packet again, as described earlier in this recipe.
2. Decrease the throughput.

   In the next screenshot we see an example of retransmissions that reduce the sender throughput (red thin lines added for clarity):


Wireshark IO Graphs: FTP download with retransmissions --- 06-MAY-2013.pcapng
2000
1000
0
140s
160s
180s
200s
220s
240s
260s
280s
Graphs
Graph 1
Color
Filter:
tcp.stream eq 64
Style:
Line
Graph 2
Color
Filter:
tcp.stream eq 64 and tcp.analysis.retransmission
Style:
Dot
Graph 3
Color
Filter:
Style:
Line
Graph 4
Color
Filter:
Style:
Line
Graph 5
Color
Filter:
Style:
Line
X Axis
Tick interval:
1 sec
Pixels per tick:
5
View as time of day
Y Axis
Unit:
Packets/Tick
Scale:
Auto
Smooth:
No filter
Help
Copy
Save
Close

There's more...

TCP is tolerant of high delays, as long as they are reasonably stable. The algorithm that defines the TCP behavior under delay variations (among other things) is called the Van Jacobson algorithm (after the name of its inventor). The Van Jacobson algorithm enables tolerance of up to 3-4 times the average delay, so if for example, you have a delay of 100 ms, TCP will be tolerant to delays of up to 300-400 ms as long as they are not frequently changed.

See also

You can check the Van Jacobson algorithm at
http://ee.lbl.gov/papers/congavoid.pdf.

Duplicate ACKs and fast retransmissions

Another phenomenon that you will see in TCP is what is called duplicate ACKs and fast retransmissions. This phenomenon also happens due to performance problems, and in this recipe we will focus on how to find them and what they indicate.

Getting ready

When you see that the network becomes slow, one of the reasons for this can be duplicate ACKs. Connect the Wireshark in the port mirror to the suspicious client or server and see the results.

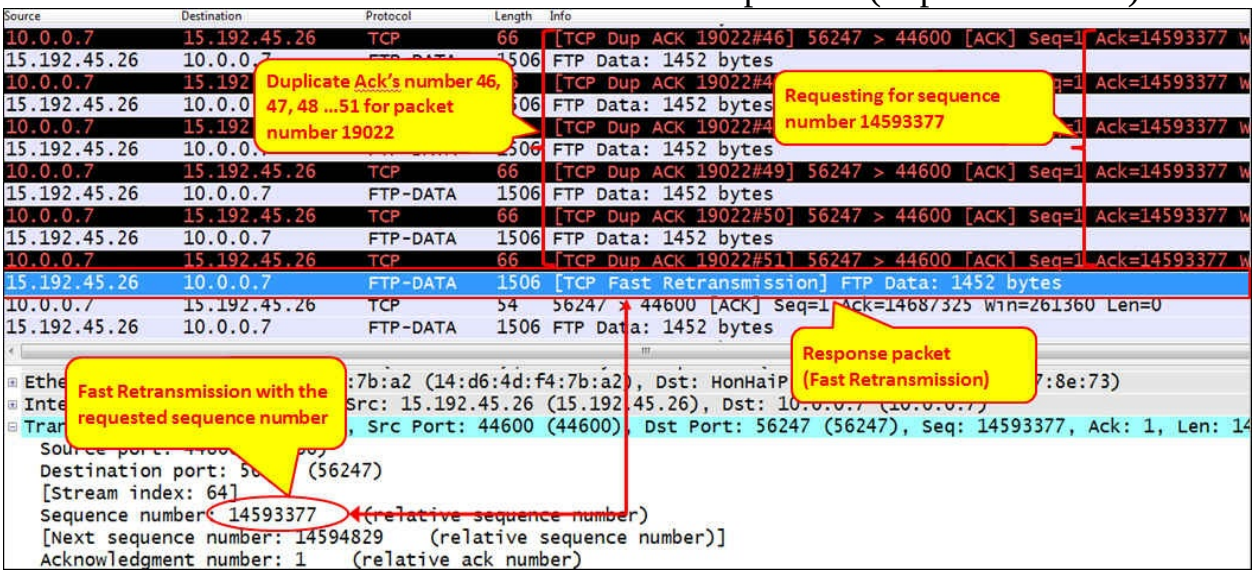
How to do it...

In most cases, duplicate ACKs will happen because of high latency, delayed variations, or a slow end point that simply does not response to ACK requests.

When looking for a reason for slow communication, duplicate ACKs can be one of the reasons for it.

1. When you see a reasonable amount of duplicate ACKs, that is, 1 or 2 percent, this is probably not your problem.
2. When you see a huge number of duplicate ACKs (say ten of them), you might have:
    - A very busy communication line that causes variations in delays
    - A non-responsive server or client (depends on who is not responding)

- A fast retransmission is a packet that is sent in response to the duplicate ACKs.
- In the next screenshot you see an example of the problem. In the example you see how a fast retransmission is sent after 51 DupACKs (duplicate ACKs):



- Here is how you can solve the problem:
    - If you have a small number of duplicate ACKs and retransmissions (less than 1 percent), it's tolerable.
    - When you have this over cellular or wireless networks, or in connections

over the Internet, the delay and delay variations are common to these networks, so there is really nothing much you can do about it.

- If you have it in your organization's network, it might be a problem. If it's on the LAN, check for severe problems, such as switch buffers and CPU load, very slow servers, and so on. If it's on the WAN, check for delays and loaded or unstable lines.

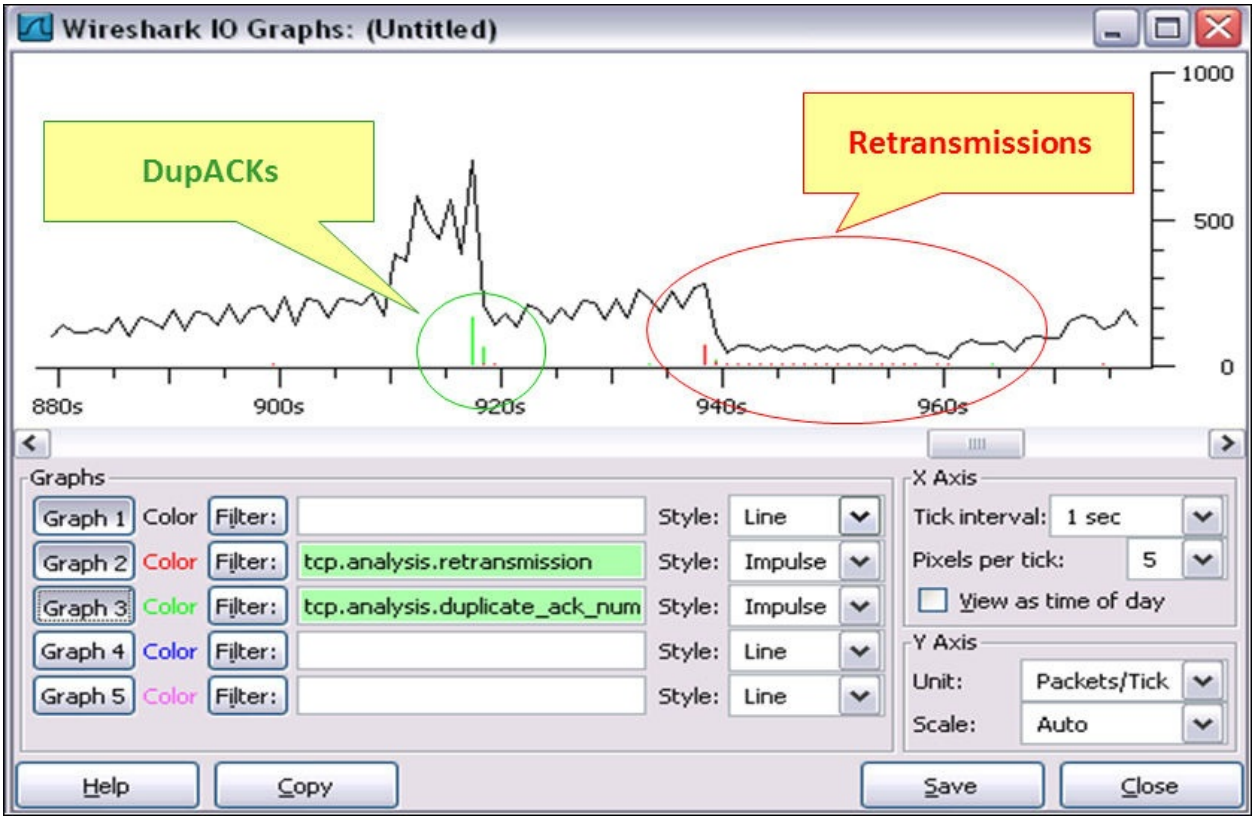
How it works...

A duplicate ACK happens if a packet is detected as missing (the expected sequence number was not received), or when an unexpected sequence number was received. In this case, the receiver generates an ACK indicating the next expected sequence number that it wishes to receive. The receiver will continue to generate ACKs requesting the missing segment, until it will receive it.

On the sender side, when it receives three identical ACKs (the original ACK and two duplicate ACKs) with the same value, it assumes there is a packet loss and it resends the missing packet, regardless of whether the RTO is expired or not. The packet that is sent again is called fast retransmission.

Duplicate ACKs also reduce the throughput that is sent over the network. How much throughput is reduced depends on the TCP version. In the earlier versions of TCP (that is Tahoe, Reno, and New-Reno) the idea was that in the appearance of a duplicate ACK, the sender reduces the throughput to half of what it was before. In case of many DupACKs, the throughput will be reduced to minimum.

In the next screenshot you see a typical example for what is caused by duplicate ACKs and retransmissions:



In this screenshot we see that first duplicate ACKs reduce the throughput to around 40 percent of what it was before, and then retransmissions reduce it to minimum.

There's more...

The mechanism that is used in duplicate ACKs is called fast recovery. In recent years some advanced versions of TCP were introduced, especially for cellular networks, in order to improve the behavior of TCP under high and changing delay conditions. In these examples, you might see some differences in the behavior of the sender and receiver, for example, lighter degradation in performance, faster recovery, and so on. Still, the characteristic behavior of TCP is maintained.

TCP out-of-order packet events

Another phenomenon that you will see in networks is **previous segment loss** and **out-of-order segments**. Both relate to packets arriving out of order, and in some cases indicate a problem.

When you see this on a network connection, it might happen due to network problems or an interruption in capture. In this recipe we will focus on this issue and what it can cause.

Getting ready

Start Wireshark and connect it on a mirrored port. The three phenomena that we want to focus on in this recipe are:

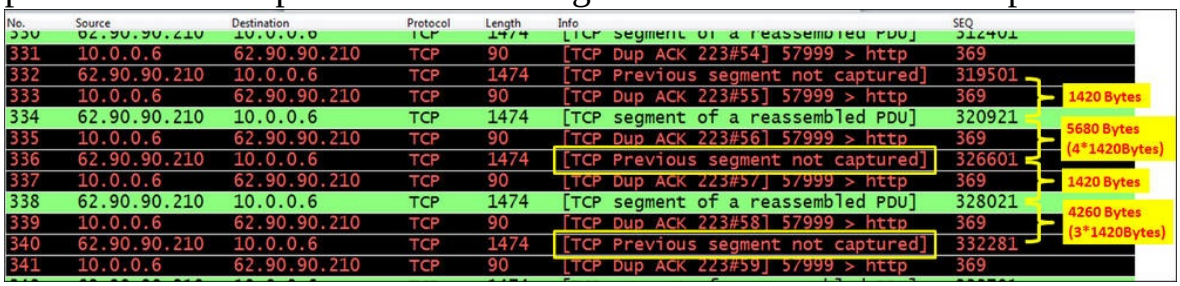
- **Previous segment lost**: This occurs when a packet arrives with a sequence number higher than the next expected sequence number on that connection, indicating that one or more packets prior to the flagged packet did not arrive
- **Out-of-order packet**: This occurs when a packet is seen with a sequence number lower than the previously received packet on that connection
- **Previous segment not captured** (Wireshark Version 1.8.x and higher): This is like the previous segment lost

How to do it...

When will it happen?

You might see these in the following events:

- **At the beginning of capture**: This event occurs when you start a capture during an open connection. In this case, you will see packets on a connection without the SYN/SYN-ACK/ACK, therefore, Wireshark thinks something went wrong.
- **Real packet losses**: In this case you will also see retransmissions of the lost packets and/or duplicate ACKs telling the sender to send the lost packets.



In the previous screenshot, we see a good example for severe packet losses. What we see here is that `10.0.0.6` is trying to browse website `62.90.90.210`. During this, the TCP segments of 1420 bytes each are sent to the web server and we see that between packets 334 and 336 three packets are missing, and between packets 338 and 340 two packets are missing. In both cases, Wireshark notices: **TCP's previous segment is not captured**.

- **Delay variations**: This can happen due to packets that take different routes from the source to destination. To check this use Tracert, and look for route changes between the source and destination (if it happens on the organization network) you can, for example, configure traps on the routers that will tell you when this happens.
- **Data capture problems**: It can be that packets are sent and received properly, but Wireshark will not have captured them. It can be because of various reasons:
    - Because of very heavy traffic Wireshark might lose packets in high bit

rates (over 150-180 Mbps). To avoid this problem, use other tools (mostly commercial).

- In case your laptop is not strong enough, lack of memory or CPU power will not enable Wireshark to work fast enough. This is easy to find out, and you are probably aware of it.
- When port buffers on a LAN switch are too small, packets can be dropped. Connect to the switch (as with console or telnet connection) and use the switch command line to check for the problem.
- Capturing data on a wireless network, when for some reason you don't see all packets that are sent. See Chapter 7, *Ethernet, LAN Switching, and Wireless LAN*.

How it works...

In this case, things are simple. The TCP sender sends the packets to the receiver. These packets are numbered by their bytes. When a packet does not arrive in order, it is a problem that Wireshark notices. We can have two reasons for this:

- **A real problem**: In this case you will see retransmissions and duplicate ACKs that are TCP's response to packets that are received out of order
- **A capture problem**: In this case you will see only out-of-order packets, and since you don't see any response to the suspected lost and out-of-order packets, they probably are not

TCP Zero Window, Window Full, Window Change, and other Window indicators

One of the most important mechanisms of TCP is the **Sliding Window** mechanism, and the **Flow Control** mechanism that uses it in order to control the amount of data that a TCP end node is willing to accept on the connection.

In this recipe we will focus on these types of problems, and how to discover the problem and solve it.

Getting ready

Connect Wireshark with a port mirror to the suspected link or server, and start capture. Keep track of every window message you will see in the capture window.

How to do it...

There are several types of window messages that you should be aware of:

TCP Zero Window, Zero Window Probe, and Zero Window Violation

TCP Zero Window occurs when a receiver advertises a receive window size of zero (in the window field in the TCP header). This tells the sender to stop sending data because the receiver's buffer is full. This indicates a problem on the receiver that might be:

- A weak server that cannot allocate enough memory for the process
- A problem in the application that does not receive a sufficient buffer, so the TCP has to tell the sender to stop sending data
- An application that consumes too much memory so the operating system will limit the application resources

TCP **Zero Window Probe** is a message that is sent by the sender in order to see if the receiver's Zero Window condition still exists. This message works by sending the next byte of data to the receiver. If the receiver answers with window that is still zero, the sender doubles his timer before probing again.

The sender ignores the Zero Window condition of the receiver and sends additional bytes of data. TCP Zero Window Violation can indicate a TCP error or bug in the protocol stack.

In order to check what the problem is, check if these events are coming from:

- A specific end device (server or client) that will indicate a problem in the end device.
- A problem in a specific application that will indicate a general application problem.
- A problem when you do something specific in the application, for example, open a specific table, send a file to the printer, create a report, or anything else on the application. In this case, it is of course an application problem.

TCP Window Update

TCP sends Window Update to the other side in a connection in order to indicate

that it changed the buffer size, and is ready to accept higher or lower data rate (buffer size determines the throughput that the sender is allowed to send). This can happen in the case of:

- The TCP receiver recovers from the Zero Window condition, and asks the sender to start sending data again. In this case, you don't have to do anything about it, just check for the problem that caused Zero Window the first time.
- The TCP receiver changes the window's size frequently. In this case check what is disturbing the receiver. It can be an application problem, memory problem, or any other performance problems on the end devices.

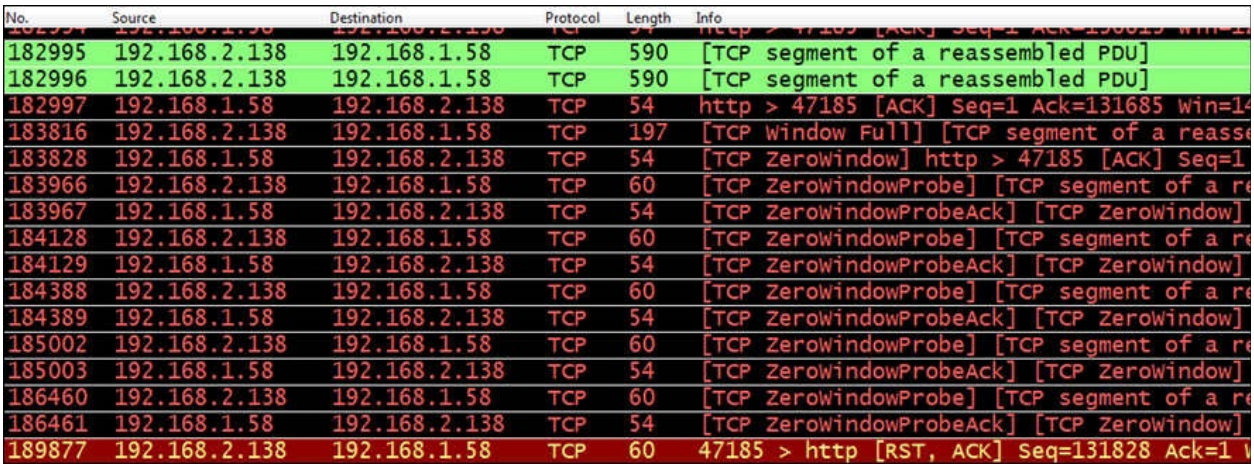
If you see this kind of phenomena, there is nothing to worry about. This is how TCP works.

TCP Window Full

This message is an indication that the sent packet will completely fill the receiver buffer on the receiver. This will happen when the receiver has not sent any ACK confirming the acceptance of the previous data, and therefore, this will be the last packet of data that the sender will send before accepting an ACK from the receiver.

On the receiver side, the moment it gets this packet, it will send a Zero Window message to the sender that will stop sending the data.

This event is triggered for the same reasons that trigger Zero Window. It is simply an indication to a non-responsive server or application. A typical example is shown in the following screenshot:



No.	Source	Destination	Protocol	Length	Info
182995	192.168.2.138	192.168.1.58	TCP	590	[TCP segment of a reassembled PDU]
182996	192.168.2.138	192.168.1.58	TCP	590	[TCP segment of a reassembled PDU]
182997	192.168.1.58	192.168.2.138	TCP	54	http > 47185 [ACK] Seq=1 Ack=131685 Win=14
183816	192.168.2.138	192.168.1.58	TCP	197	[TCP Window Full] [TCP segment of a reasse
183828	192.168.1.58	192.168.2.138	TCP	54	[TCP ZeroWindow] http > 47185 [ACK] Seq=1
183966	192.168.2.138	192.168.1.58	TCP	60	[TCP ZeroWindowProbe] [TCP segment of a re
183967	192.168.1.58	192.168.2.138	TCP	54	[TCP ZeroWindowProbeAck] [TCP ZeroWindow]
184128	192.168.2.138	192.168.1.58	TCP	60	[TCP ZeroWindowProbe] [TCP segment of a re
184129	192.168.1.58	192.168.2.138	TCP	54	[TCP ZeroWindowProbeAck] [TCP ZeroWindow]
184388	192.168.2.138	192.168.1.58	TCP	60	[TCP ZeroWindowProbe] [TCP segment of a re
184389	192.168.1.58	192.168.2.138	TCP	54	[TCP ZeroWindowProbeAck] [TCP ZeroWindow]
185002	192.168.2.138	192.168.1.58	TCP	60	[TCP ZeroWindowProbe] [TCP segment of a re
185003	192.168.1.58	192.168.2.138	TCP	54	[TCP ZeroWindowProbeAck] [TCP ZeroWindow]
186460	192.168.2.138	192.168.1.58	TCP	60	[TCP ZeroWindowProbe] [TCP segment of a re
186461	192.168.1.58	192.168.2.138	TCP	54	[TCP ZeroWindowProbeAck] [TCP ZeroWindow]
189877	192.168.2.138	192.168.1.58	TCP	60	47185 > http [RST, ACK] Seq=131828 Ack=1 W

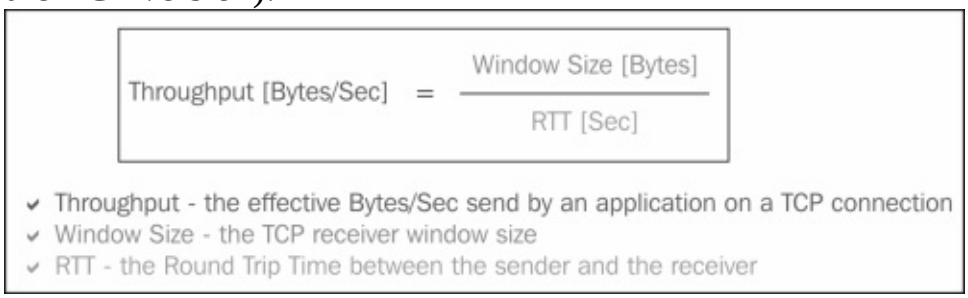
In the previous screenshot we see that:

1. Packet `183816`, `192.168.2.138` tells `192.168.1.58` that the sender window is full.
2. In the next packet, `192.168.1.58` sends a signal to `192.168.2.138`, telling him to stop sending data. This is a Zero Window signal.
3. Both sides continue to send Zero Window and Zero Window Probe.
4. The last packet of the connection is an RST sent by `192.168.2.138` in order to break the connection.
5. In some cases Zero Window condition will be recovered by a window-change message. In some cases it will be closed with a reset (that can be because an application does not receive any data because of Zero Window).

How it works...

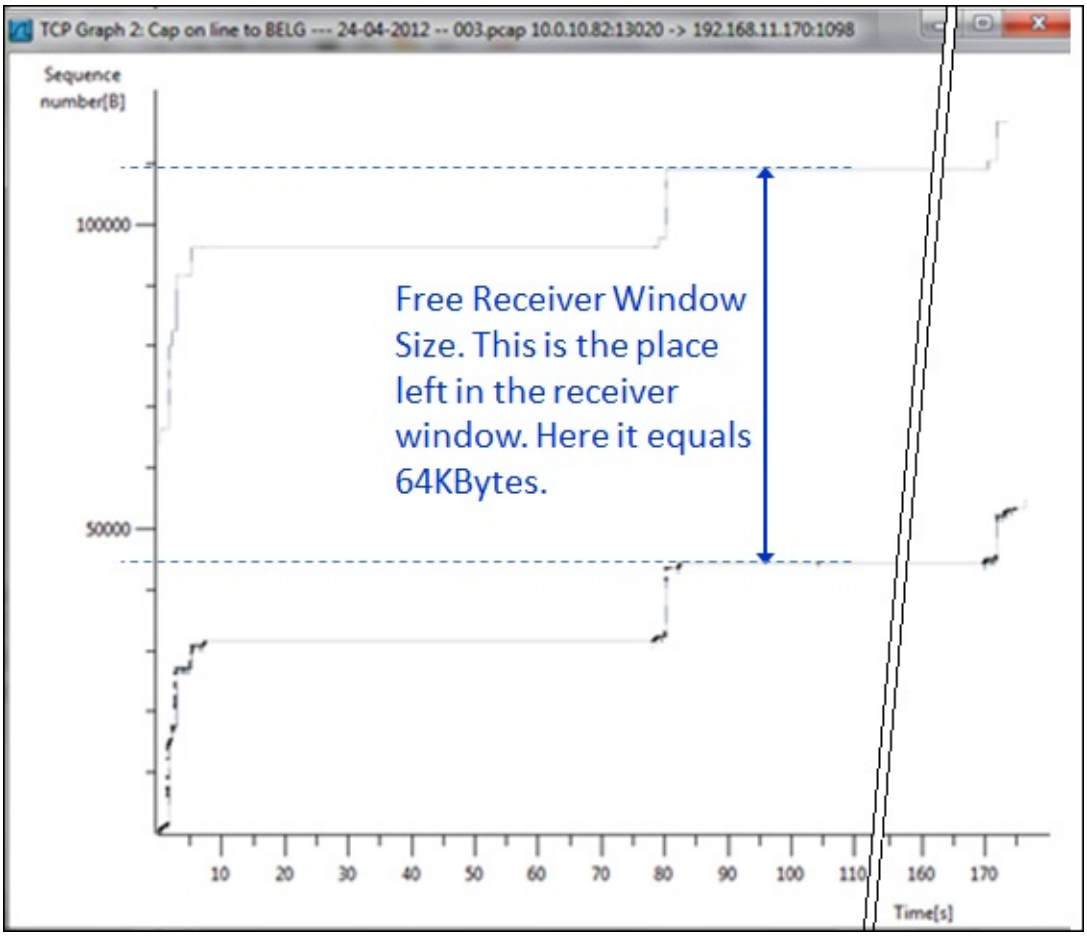
The TCP Sliding Window mechanism works as follows:

1. After the connection is established, the sender sends data to the receiver, filling the receiver window.
2. After several packets, the receiver sends an ACK to the sender, confirming the acceptance of the bytes sent by it. Sending the ACK empties the receiver window.
3. This process is continuous when the sender is filling the window, and the receiving party empties it and sends confirmation of the information.
4. Increasing the receiver window size tells the sender to increase the throughput, and decreasing it tells him to decrease the throughput. It works according to the following WS/RTT rule (with some changes according to the TCP version):



There's more…

You can also use the TCP throughput graphs and the IO graphs to view these problems. In the TCP throughput graphs, use the TCP trace graph, where the upper line indicates the window size, and its distance from the lower line indicates what is on the left-hand side of the window. No distance between them indicates a Zero Window.



A fixed distance between the lines (as shown in the preceding screenshot) indicates a good operation on the receiving side. When the lines are getting closer, it indicates that the sender is overwhelming the receiver. As long as lines are not overlapping, TCP will continue to send data.

TCP resets and why they happen

During a normal operation, TCP will open a connection with SYN signals, and close the connection with FIN signals. One of the characters of TCP is the possibility to close a connection faster due to a problem or just for better efficiency.

In this recipe we will describe these cases, and how to understand exactly what happens, and if it is a regular condition or something went wrong.

Getting ready

Connect Wireshark with a port mirror to the suspected link or server, and start capture. Keep track of every window message you will see on the capture window. TCP resets can be sent in several cases. Some point to the proper working of the protocol, and some suggest a failure or problem. In this recipe, we will get to the reasons for it, and try to point out the problems and how to solve them.

How to do it...

Reset is a TCP signal that is sent in order to tell the receiver to break the connection. Reset is sent by setting the RST flag to a value of `1`.

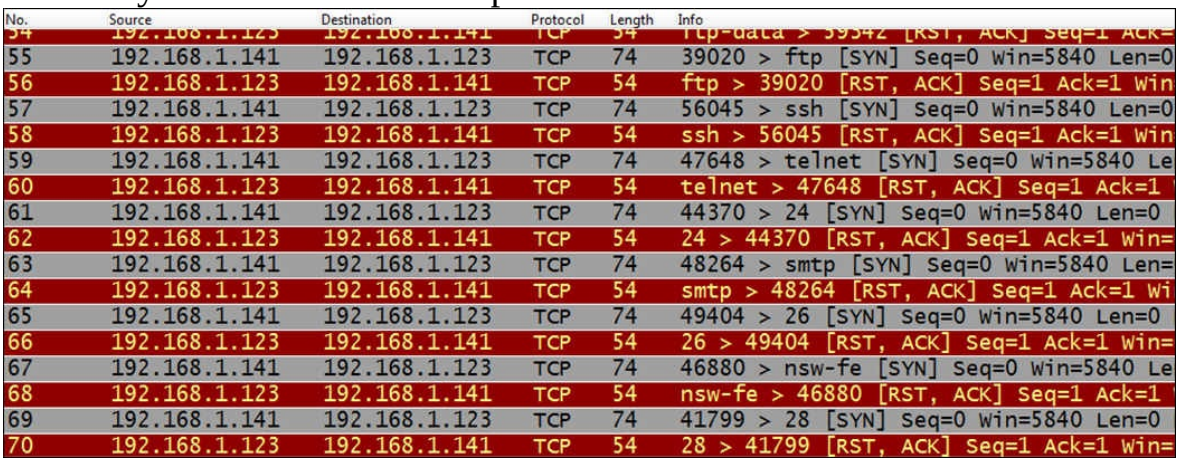
Cases in which reset is not a problem

The standard way of closing a connection in TCP is by FIN and FIN-ACK signals. The problem is that in order to close a connection, you need four packets: FIN/ACK and ACK from one side, and the same from the other side. It can happen, for example, when you open a standard web page, tens of connections (the main page, news bars, commercials, pictures that are updated periodically, and so on) can be opened, and in order to close all of them you will need sometime hundreds of FIN and FIN-ACK packets standard way. In order to prevent it from happening, the web server will, in many cases, send you the requested data and then break the connection with reset. This is a standard thing to do, and it depends on the application.

Cases in which reset can indicate a problem

There are some cases in which resets can indicate a problem (not necessarily a communication problem):

- **A reset sent by a firewall**: When you try to open a connection to a remote server, and don't get anything, you might see an RST signal coming back. This is a firewall blocking a connection. In the next screenshot, you can see that every SYN that is sent is replied to with an RST.



No.	Source	Destination	Protocol	Length	Info
54	192.168.1.123	192.168.1.141	TCP	54	ftp-data > 59542 [RST, ACK] Seq=1 Ack=
55	192.168.1.141	192.168.1.123	TCP	74	39020 > ftp [SYN] Seq=0 Win=5840 Len=0
56	192.168.1.123	192.168.1.141	TCP	54	ftp > 39020 [RST, ACK] Seq=1 Ack=1 Win
57	192.168.1.141	192.168.1.123	TCP	74	56045 > ssh [SYN] Seq=0 Win=5840 Len=0
58	192.168.1.123	192.168.1.141	TCP	54	ssh > 56045 [RST, ACK] Seq=1 Ack=1 Win
59	192.168.1.141	192.168.1.123	TCP	74	47648 > telnet [SYN] Seq=0 Win=5840 Le
60	192.168.1.123	192.168.1.141	TCP	54	telnet > 47648 [RST, ACK] Seq=1 Ack=1
61	192.168.1.141	192.168.1.123	TCP	74	44370 > 24 [SYN] Seq=0 Win=5840 Len=0
62	192.168.1.123	192.168.1.141	TCP	54	24 > 44370 [RST, ACK] Seq=1 Ack=1 Win=
63	192.168.1.141	192.168.1.123	TCP	74	48264 > smtp [SYN] Seq=0 Win=5840 Len=
64	192.168.1.123	192.168.1.141	TCP	54	smtp > 48264 [RST, ACK] Seq=1 Ack=1 Wi
65	192.168.1.141	192.168.1.123	TCP	74	49404 > 26 [SYN] Seq=0 Win=5840 Len=0
66	192.168.1.123	192.168.1.141	TCP	54	26 > 49404 [RST, ACK] Seq=1 Ack=1 Win=
67	192.168.1.141	192.168.1.123	TCP	74	46880 > nsw-fe [SYN] Seq=0 Win=5840 Le
68	192.168.1.123	192.168.1.141	TCP	54	nsw-fe > 46880 [RST, ACK] Seq=1 Ack=1
69	192.168.1.141	192.168.1.123	TCP	74	41799 > 28 [SYN] Seq=0 Win=5840 Len=0
70	192.168.1.123	192.168.1.141	TCP	54	28 > 41799 [RST, ACK] Seq=1 Ack=1 Win=

- **Reset sent due to a problem on one of the sides**: Here you can have many reasons. Some of them are as follows:
    - One of the reasons is the five consecutive retransmissions that are not replied to by an ACK. When the sender does not get any reply for the retransmissions, it will send a reset signal to the other side, telling it to break the connection.
    - Another reason is a connection without any traffic on it for a few minutes (how many minutes is the operation system default). The side that opened the connection will usually send the reset (usually but not always, it depends on the implementation).

How it works...

Here it is simple. Reset is a signal that is used in order to break a connection. It is important to remember here that everything depends on the application. If the programmer chose to send an RST on a specific case, this is what you will see on the capture file. For every reset that you see, try to figure out what caused it and you will understand it from the packets the before reset was sent.

Chapter 10. HTTP and DNS

In this chapter, we will go through the following recipes:

- Filtering DNS traffic
- Analyzing regular DNS operations
- Analyzing DNS problems
- Filtering HTTP traffic
- Configuring HTTP preferences
- Analyzing HTTP problems
- Exporting HTTP objects
- HTTP flow analysis and the Follow TCP Stream window
- Analyzing HTTPS traffic – SSL/TLS basics

Introduction

Domain Name System (**DNS**) is a protocol that is used for resolving names to the IP addresses. It is used over the Internet when you browse a website, and then the DNS resolves the web server name to an IP address. It is also used in enterprise networks when looking for a server name that is translated to an IP address.

Hyper Text Transfer Protocol (**HTTP**) and **Secured HTTP** (**HTTPS**) are both used for browsing the Internet, or connecting to other software that are hosted inside your organization or in the cloud. HTTPS is used when we secure HTTP with SSL/TLS in order to hide the clear text data exchange from hacking. It is used when connecting to your bank, webmail account (for example, Gmail or Hotmail), or any other secured application.

In this chapter, we will discuss these protocols, how they work, and how to use Wireshark in order to find common errors and problems in them.

Filtering DNS traffic

DNS is a protocol responsible for resolving names to the IP addresses. In this recipe, we will learn how to filter important parameters that are related to the DNS service.

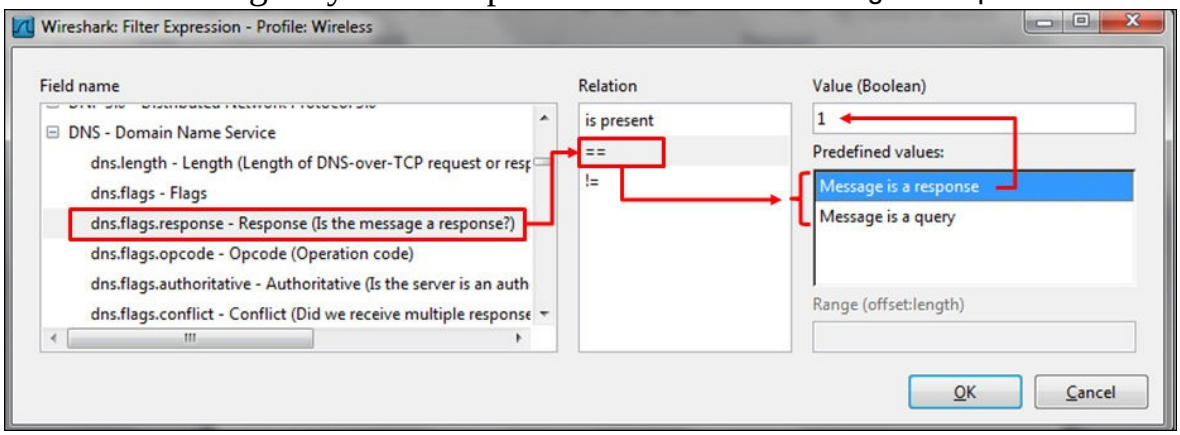
Getting ready

When suspecting a network problem, port mirror the suspected server or install Wireshark on it, then, start capturing the data.

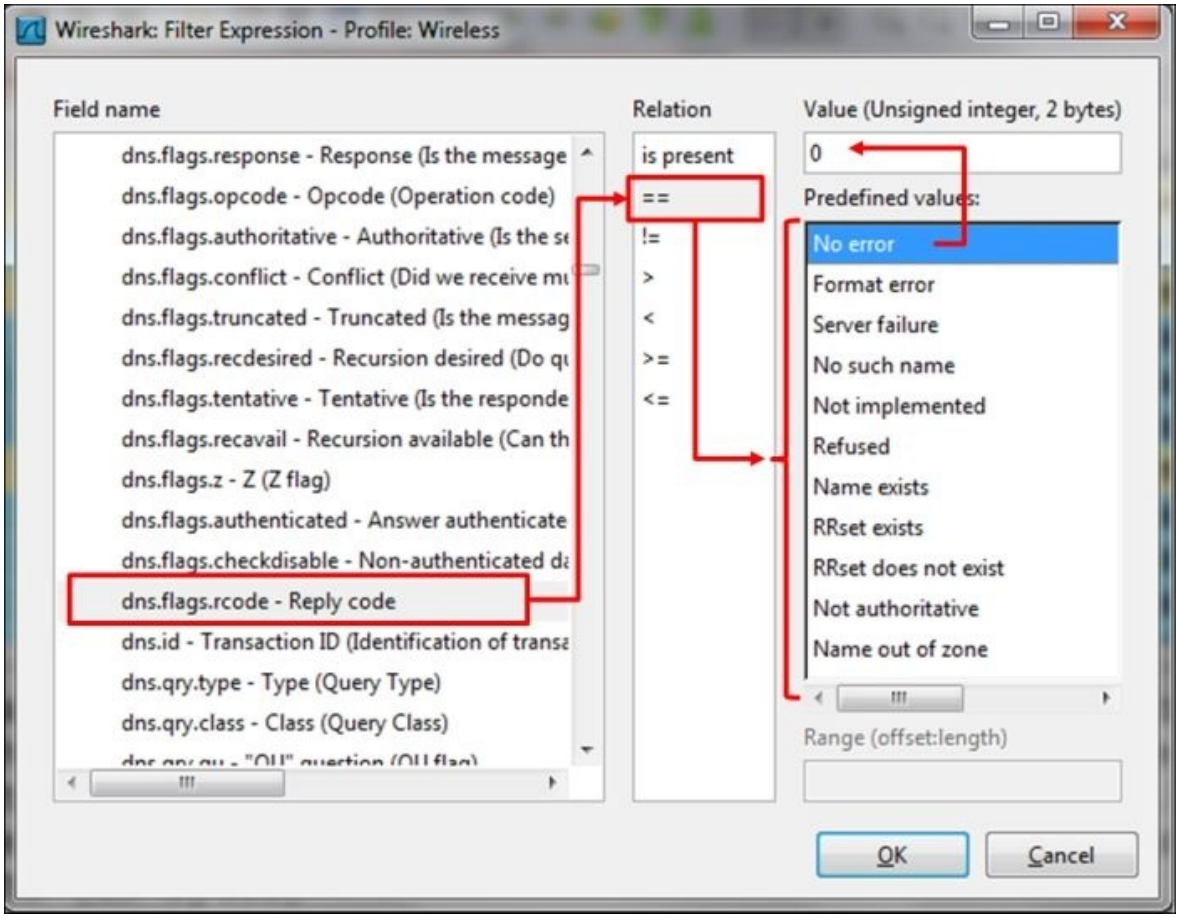
How to do it...

There are some common filters that will assist you in troubleshooting DNS problems. The common display filters are given as follows:

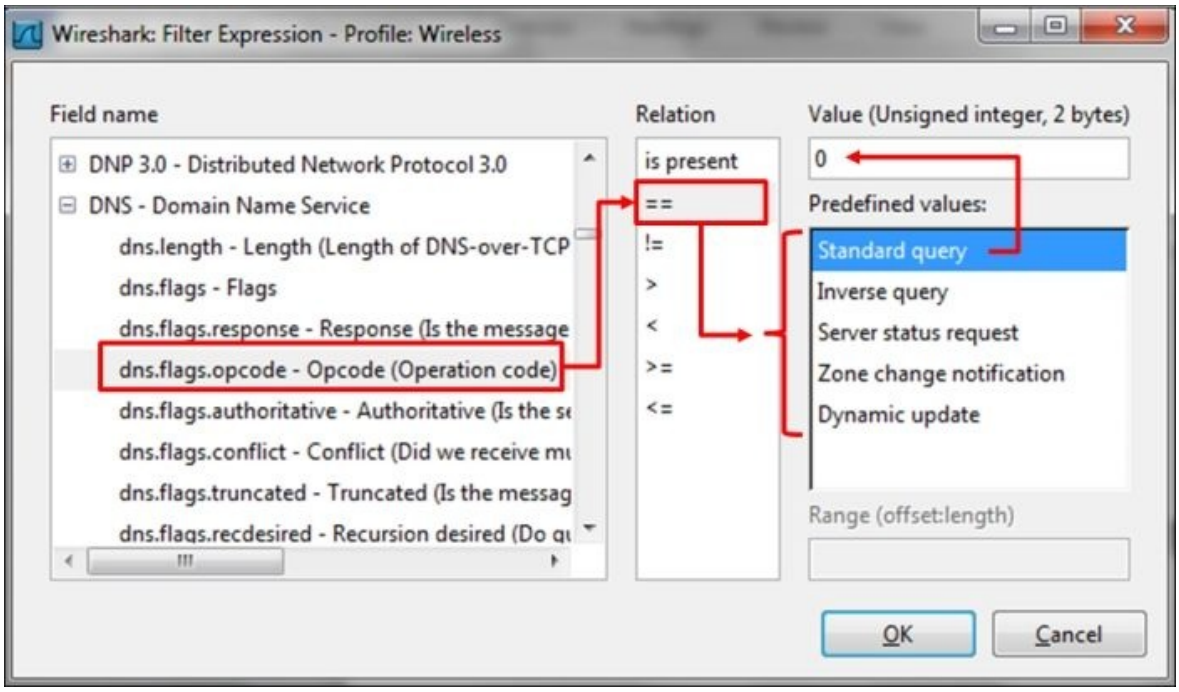
- The basic filter is simply for filtering DNS traffic. The filter is `dns`.
  - For filtering only DNS queries we have `dns.flags.response == 0`
  - For filtering only DNS responses we have `dns.flags.response == 1`



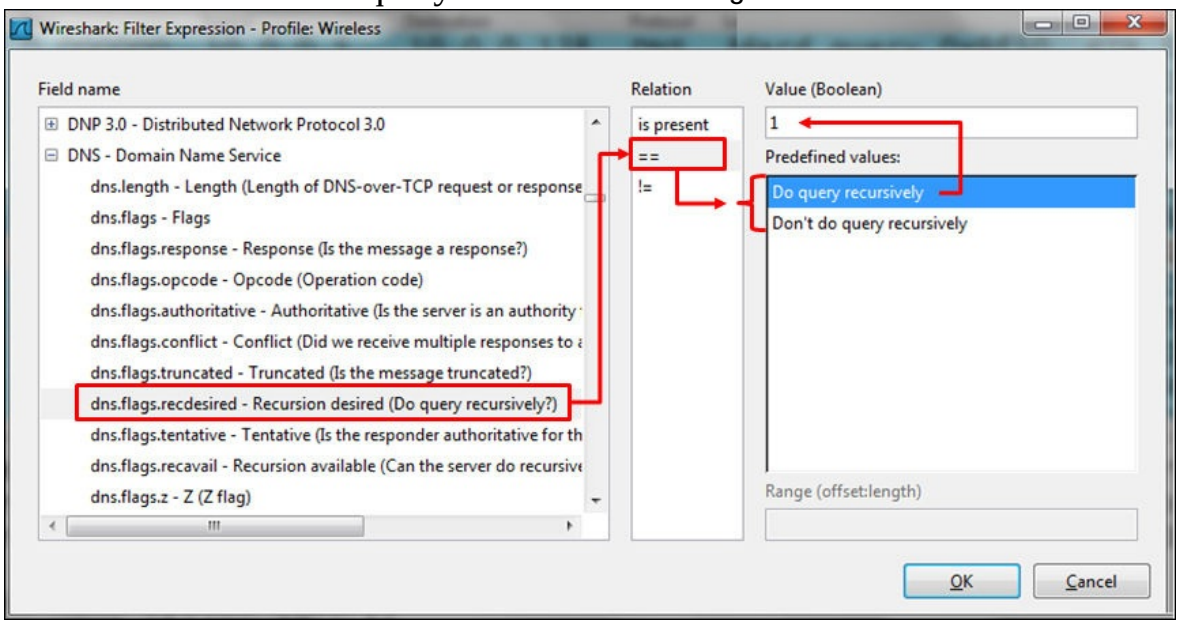
- For filtering error codes, we have the following filters:
  - No error (rcode—reply code), we have `dns.flags.rcode == 0`, marked in the following screenshot
  - No such name, we have `dns.flags.rcode == 3`



- For search problems, we have the following filters:
  - **When looking for a specific URL**: This will be used, for example, when you are not sure whether your PC is sending the DNS query, use `dns.qry.name == "URL Name"`
  - When looking for a query that contains a specific URL: For this case we have `dns.qry.name contains "URL Name"`
- For filtering DNS Opcodes (standard query or other requests or notifications), we have the following filters:
  - For filtering only standard queries: `dns.flags.opcode == 0`
  - For filtering only inverse queries: `dns.flags.opcode == 1`
  - For filtering server status requests: `dns.flags.opcode == 2`
  - For filtering zone change notifications: `dns.flags.opcode == 4`
  - For filtering dynamic updates: `dns.flags.opcode == 5`



- For querying the query types (recursive/non-recursive):
  - For recursive query we have `dns.flags.recdesired == 1`
  - For non-recursive query we have `dns.flags.recdesired == 0`



All other display filters can be found by clicking on the expression button on the right-hand side of the display filter window at the top of the Wireshark window.

How it works...

Display filters are described in depth in Chapter 3, *Using Display Filters*. As described in Chapter 3, *Using Display Filters*, you can do one of the following things to filter DNS parameters:

- Click on the expression button on the right to the display filter window, and choose the required filter from DNS
- Go to the packet details, right-click on the required field, and choose **Apply a filter** or **Prepare a filter**
- Simply write the filter string in the filter window at the top of the Wireshark window

There's more...

DNS is quite a complicated protocol, and the purpose of this chapter is to provide methods to resolve common problems with this protocol and implementation. Not all filters are mentioned here; a full list of DNS filters can be found at http://www.wireshark.org/docs/dfref/d/dns.html.

Analyzing regular DNS operations

In this recipe, we will see how to find out if DNS is working properly or not. We will see some scenarios of DNS operations, and what can go wrong.

Getting ready

Open Wireshark and start capturing data. You should mirror a device that is using DNS, or the DNS server itself.

How to do it...

Connect Wireshark to the LAN switch attached to the monitored device, and configure port mirror to the device from which you suspect the problem is coming. Go through the following steps:

1. In case of user complains, configure the port mirror for monitoring the user device.
2. In case of a general problem in the network, configure port mirror to the DNS server:
    - When the DNS server is configured on the internal server, configure port mirror on the server
    - When the DNS server is configured on external server, configure port mirror to the link that connects you to the Internet

How it works...

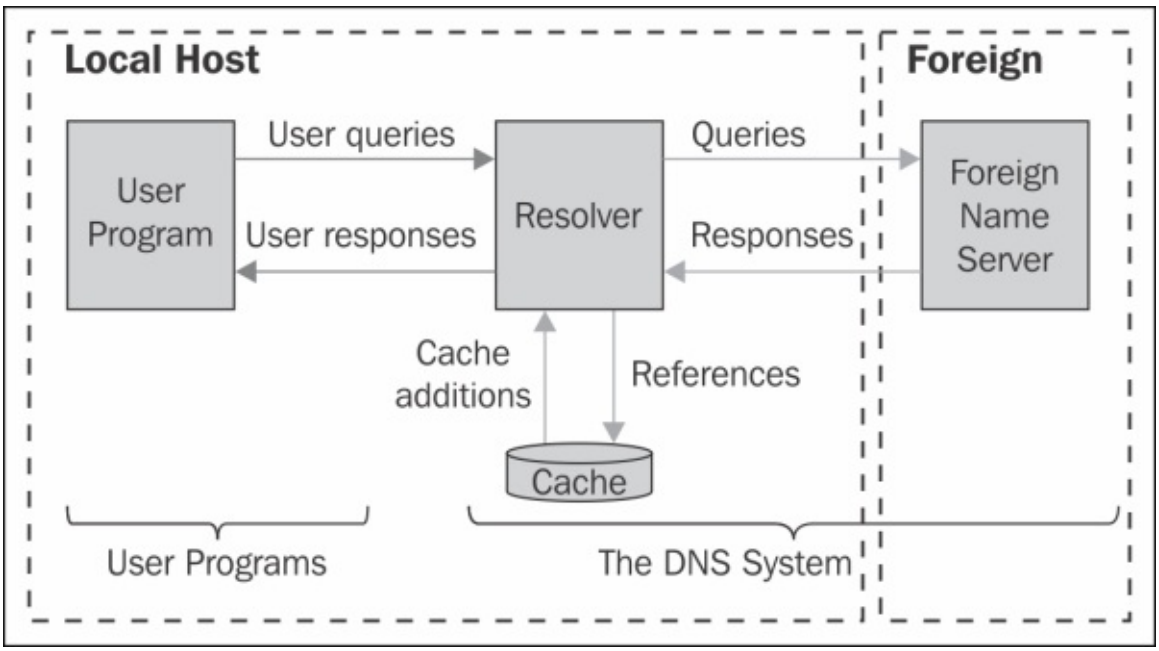
DNS is the major protocol used for name resolution, and it is used when browsing the Internet. It is also used for working in the organization network. The DNS standards describe three functionalities:

- Namespace which is what DNS names look like, and how they are allocated
- The name registration process, that is, how we register DNS names and how they are forwarded through the DNS servers' network
- The resolving process, that is, how names are resolved to the IP addresses

In this recipe we will focus on the third subject, that is, what happens when we browse the Internet, send or receive e-mails, or access internal servers in our organization. The basic DNS operation is shown in the following diagram:

DNS operation

User programs (web browser, mail client, and many others) interact with the DNS server through a resolver, which is also a part of the operating system. The resolver interacts with external name server that provide it with the required IPs (the name server can be local or remote; it is external to the resolver). The way the user queries the DNS server is OS specific. DNS queries and responses are sent and received between the resolver and the name server.



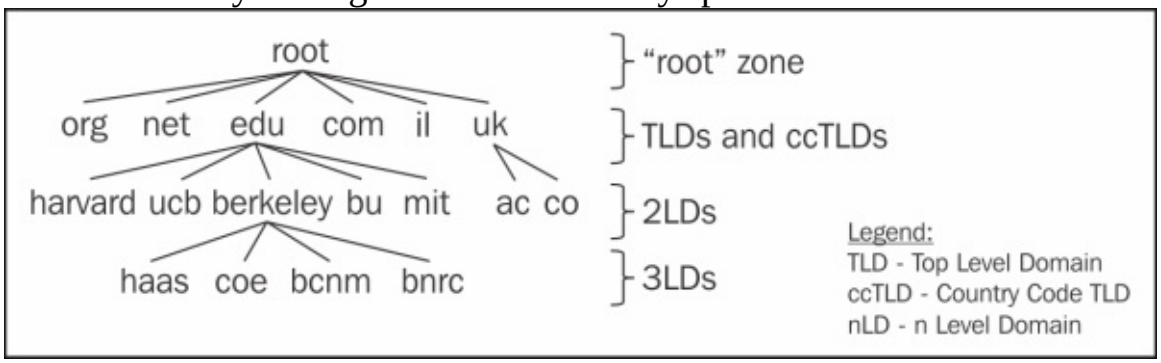
The local name server is usually located in the organization network, and interacts with the DNS server of your ISP. In the case of a home or a small office network, your DNS server can be configured on the router that connects you to the Internet, or directly to the DNS server of your ISP:

- When the DNS server is on the router, you query the DNS on the router that queries your ISP DNS
- When your DNS is located on the ISP network, you query the DNS server directly

DNS namespace

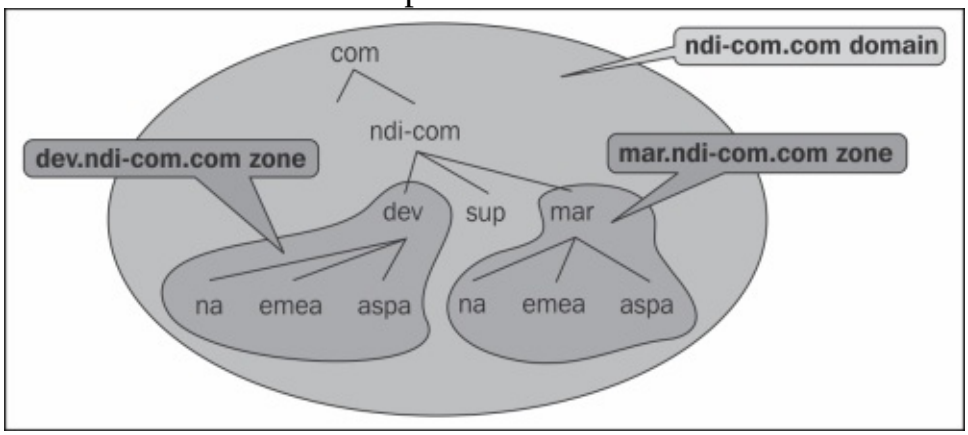
The DNS namespace is based on a hierarchical tree structure, as presented in the next diagram. The structure is as follows:

- The network of root servers (http://www.iana.org/domains/root/servers).
- The network of **Top Level Domain servers** (**TLDs**) (http://www.iana.org/domains/root/db).
- Each top-level domain has name servers similar to that of **IANA administers**. Top-level domains contain second-level domains. TLDs are the highest-level servers, for example, country servers as illustrated in the next diagram.
- **Second Level Domains** (**SLDs**) contain the domains and names for organizations and countries. The names in second-level domains are administered by the organization or country specified.



There are some important definitions, as shown in following diagram:

- **Domain**: It constitutes all branches under **ndi-com.com**, in this case a second level domain
- **Zone**: It is a contiguous portion of a DNS domain in the DNS namespace, whose database records exist and are managed in a particular DNS database file stored on one or multiple DNS servers



The resolving process

There are two reasons for using DNS servers:

- The first reason is that it is used for internal communication in your organization. In this case, you have a DNS server in your organization, which resolves the IP addresses to names in your organization.
- It is used for connecting to the Internet, browsing, sending mails, and so on.

When both services are used, you will send the DNS query to your organization server, which will send the query to the Internet. For example, when you want to get to a local server in your organization, you will send a DNS query to the local DNS, and you will get the server IP. When you browse a website on the Internet, your local DNS server forwards the request to the external DNS, for example, the ISP DNS.

Is it the correct DNS server you have configured? Theoretically, when you connect to the Internet, you can configure any DNS server in the world. Usually, the best DNS server to use is the nearest one. In your organization, you should configure your local DNS as first priority, and then the DNS servers of your ISP.

There are various utilities to check the DNS response. Some of them are as follows:

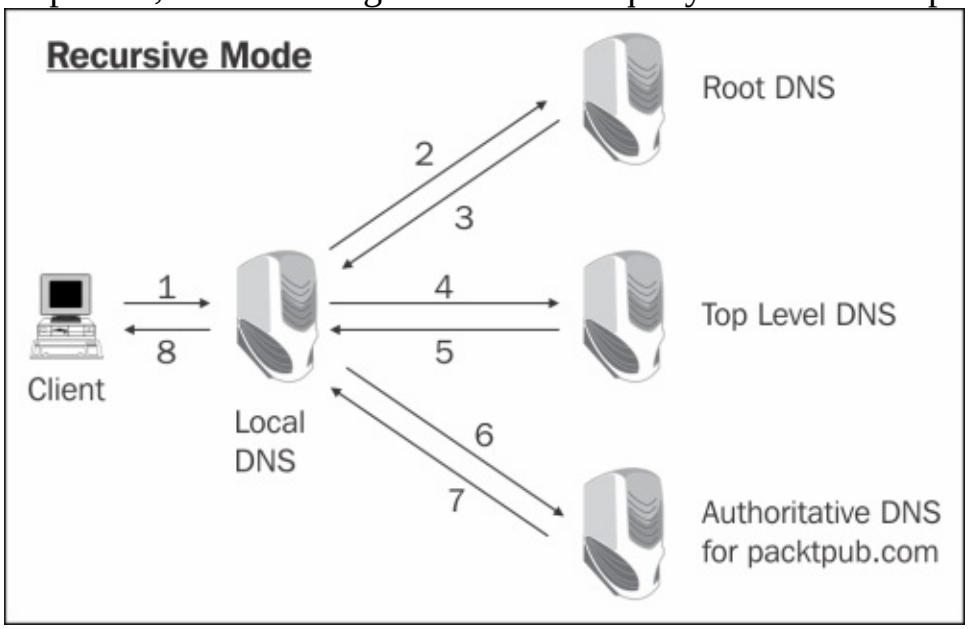
- **Namebench** from Google (http://code.google.com/p/namebench/downloads/detail?name=namebench-1.3.1-Windows.exe&can=2&q=)
- **DNS Benchmark** from GRC (https://www.grc.com/dns/benchmark.htm)

In the test result, you should get a good response time for your configured DNS servers. If not, change them.

There's more...

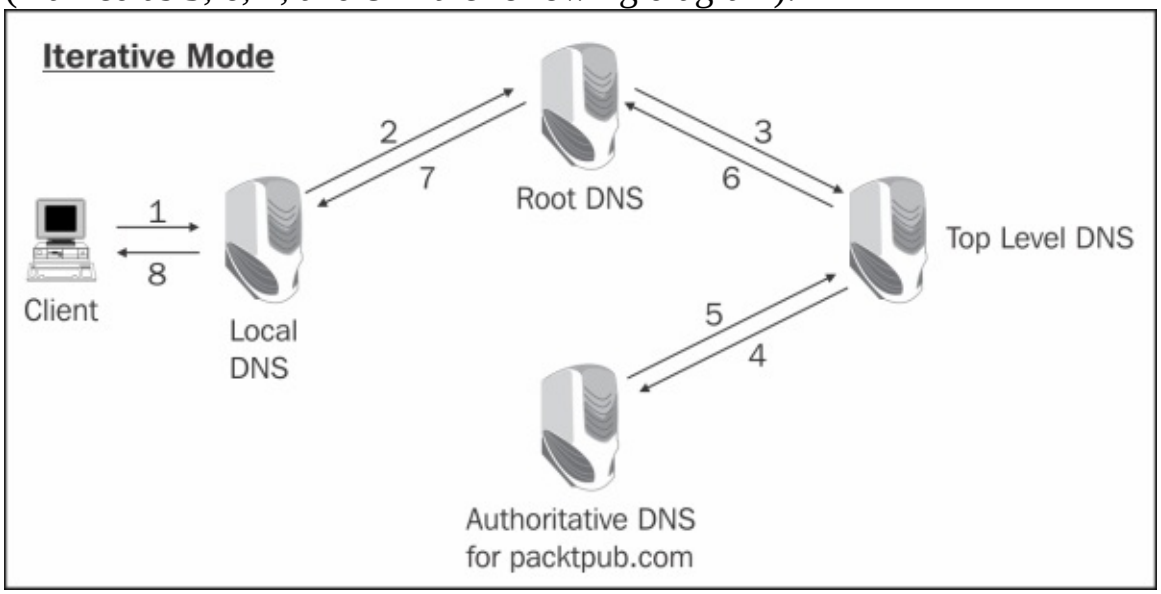
When a process on the end device is looking for the IP address of a specific name, it interacts with the local resolver that goes out to the DNS servers. When the DNS server does not find the entry you are looking for in its database, it can respond in two ways—**recursive** or **iterative**:

- **Recursive mode**: In this mode, when the application (for example, a web browser) wants to resolve the name of the website www.packtpub.com, it sends a DNS request to the local DNS server (marked as **1** in the following diagram). The local DNS server sends the request to a root server (marked as **2** and **3** in the following diagram), then to the TLD (marked as **3** and **4** in the following diagram), and finally to the authoritative server of www.packtpub.com, which gives us the required address (marked as **6** and **7** in the following diagram). Then, the local DNS server sends us the required address (marked as **8** in the following diagram). In each one of the responses, the resolver gets the DNS to query in the next step.



- **Iterative mode**: In this mode, a DNS client can receive a response from the DNS server that will tell the client where to look for the requested name. When the application (for example, a web browser) wants to browse the website www.packtpub.com, it sends a DNS request to the local DNS server (marked as **1** in the following diagram). The local server forwards the request to a root DNS server (marked as **2** in the following diagram). If

it doesn't know the answer, it forwards the request to the TLD (marked as **3** in the following diagram) and the authoritative DNS (marked as **4** in the following diagram). Then, the answer is sent all the way back to the client (marked as **5**, **6**, **7**, and **8** in the following diagram):



Analysing DNS problems

In the previous recipe, we saw how to identify a normal operation of DNS. In this recipe, we will learn how to discover problematic behavior of DNS, and how to figure out its source.

Getting ready

A DNS problem can result in bad performance while browsing the Internet, slow network while working inside the organization network, or any other performance issues. We will see how to isolate these problems and how to find out whether it is a DNS issue or not.

How to do it...

There are two major types of problems in DNS:

- DNS cannot resolve a name
- Slow operation of DNS

In both cases, connect your Wireshark to the network in the following order when you suspect an Internet connectivity problem:

1. First, port mirror the PC of the customer complaining about the problem. In this step, you will see specific problems on the PC.
2. Then, port mirror your DNS server. In this step, you will be able to find the general problems that are common to the entire organization (or at least to the part of it that has a problem).

DNS cannot resolve a name

How will you know that this is the problem?

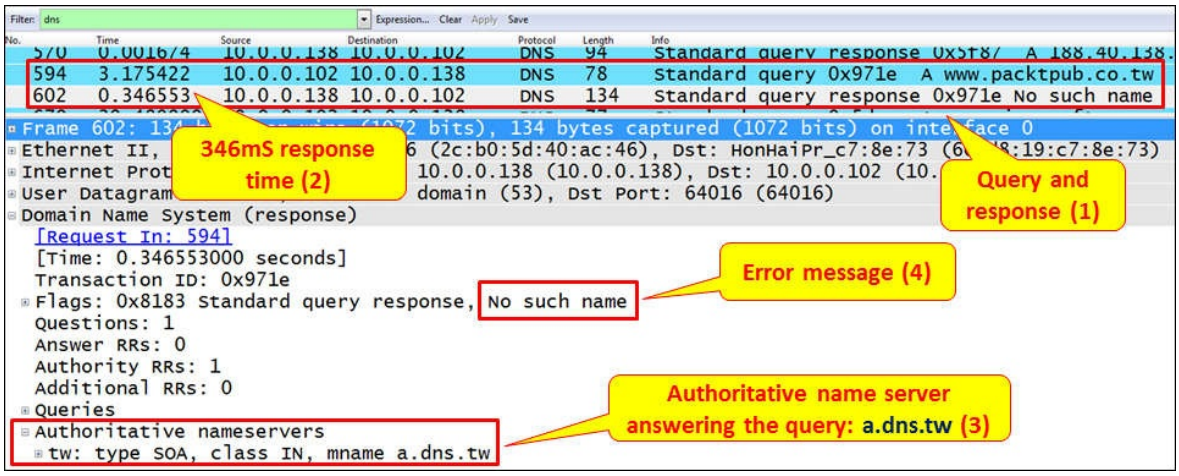
1. You try but cannot browse the Internet, send e-mails, or perform any other operations on the Internet.

   Assuming your connectivity to the network is working properly, ping the website you are trying to browse (for example, issue the command: `ping www.packtpub.com`) and see if you get any response.
2. If you get a response, all is working OK.
3. If you don't get any response, it can be because of the following reasons:
   - The website you are trying to ping blocks the ICMP requests
   - The DNS server you are trying to get the data from is not functioning
4. To make sure that this is a DNS problem, start Wireshark and configure the DNS filter. In case of a problem, you will see one of the following:
   - When a website does not exist
   - Cannot reach the DNS server

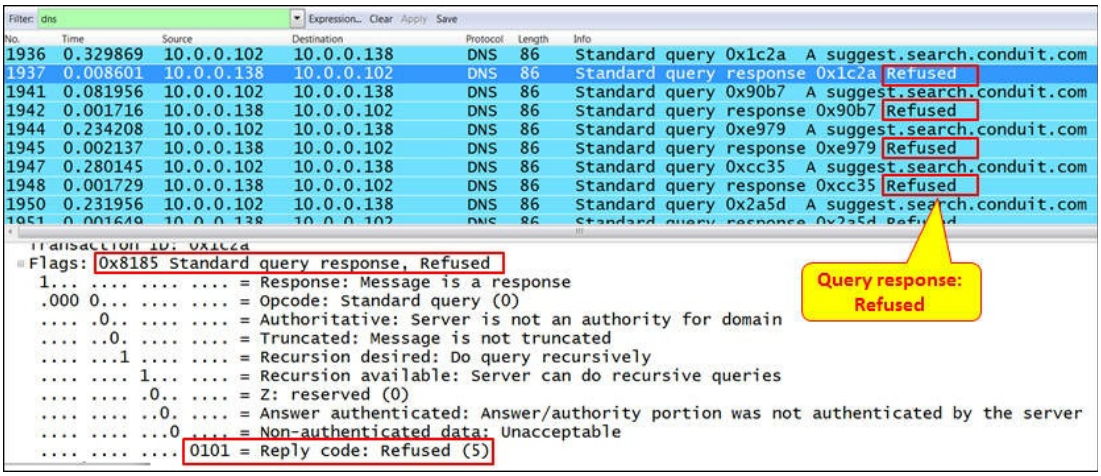
   ### Tip

   You can also use the command `nslookup` in the command line. This command checks the IP of the inserted name.

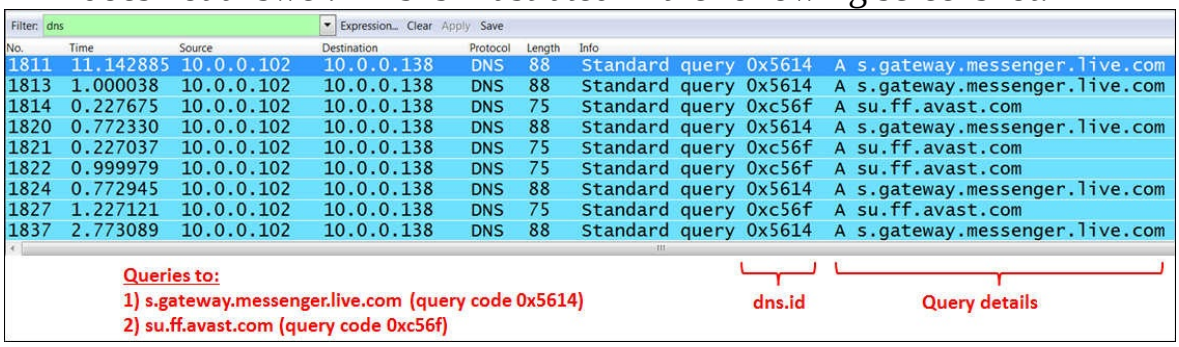
5. When the website does not exist, you will see (example in the following screenshot):
    - The DNS query and response, both with code `0x971e` (the same code in query and response indicates that this is the response to the query)
    - A `346 ms` delay between the DNS query and response, which means that the response came from an overseas server (for example, browsing from Europe when the DNS server is in Taiwan)
    - The request was sent and was replied from `a.dns.tw` (that is, DNS server is in Taiwan), which means that the DNS system works properly and your PC queried one of the authoritative DNS servers for `.tw`
    - The response is **No such name**, which means that there is no such server



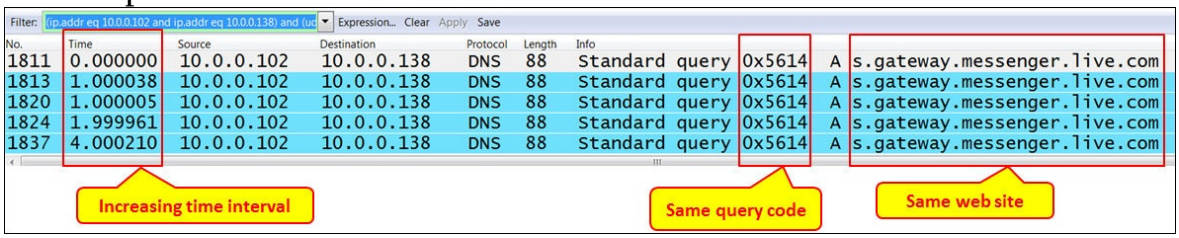
6. When the DNS server does not respond, you will see one of the following screenshots:
    - **The DNS refused message**: In this case, your DNS server refuses the request. This is illustrated in the following screenshot (you will learn why in the *How it works...* section):



- **The DNS consecutive queries**: In this case, the DNS server simply does not answer. This is illustrated in the following screenshot:



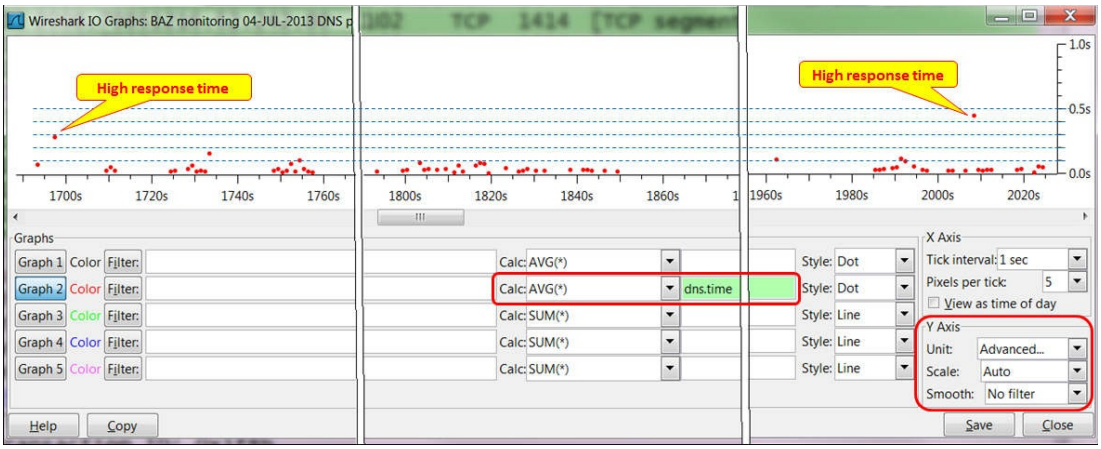
When you right-click on one of the packets in the preceding screenshot and choose **Follow UDP Stream**, you will see that the DNS resolver on your PC sends several queries (with increasing time intervals between them), and then stops. This is shown in the next screenshot:



DNS slow responses

How will you know that this is the problem?

1. When you are browsing the Internet and getting very slow responses, perform the following steps:
    1. Port mirror the connection to the Internet, and check if you have any bottleneck on the way to the Internet. You can use the IO graphs for this purpose, as described in Chapter 5, *Using Advanced Statistics Tools*.
    2. Verify that you don't have a significant number of retransmissions or duplicate ACK's indicating a connection problem.
    3. Verify that you don't have any window-related problem, such as zero window or window full.
2. If answers are no for the preceding checks, it might be a DNS problem. You can have DNS problems in two cases:
    - When working in your organization
    - When connecting to the Internet
3. These issues can be resolved in two ways:
    - When facing problems in your organization, port mirror the switch port that is connected to the DNS server
    - When facing problems with the Internet, port mirror the switch port that connects your organization to the Internet
4. Watch the DNS response time that you get. There are several ways to locate the problem, and they are given as follows:
    - The simplest way is to right-click on a packet from a DNS query stream, choose **Follow UDP Stream**, and then check the time between the query and response.
    - Another way is to use IO graphs for this purpose. In the IO Graphs window, choose **Advanced** in the **Y Axis** configuration and configure the filter **dns.time** with **AVG(*)** in the **Graph** lines. Refer the following screenshot:



You will get a graph of the DNS response times throughout the capture time.

In this graph, you will see that most of the response times fall below 100mSec, which is quite reasonable. We have two peaks that indicate a probable problem, one at the beginning of the capture with 300 ms, and one at the end of the capture with 450 ms.

Tip

Reasonable times inside the organization (in a local site) should be not more than tens of milliseconds. When browsing the Internet, a good response time should be less than 100 ms, while up to 200 ms is still tolerant.

How it works...

There are six basic types of DNS response codes defined in `RFC 1035`. Additional error codes (up to 21) were defined in later standards (`RFC 2136`, `RFC 2671`, `RFC 2845`, and `RFC 2930`). Error codes can be found at http://tools.ietf.org/html/rfc2929#section-2.3.

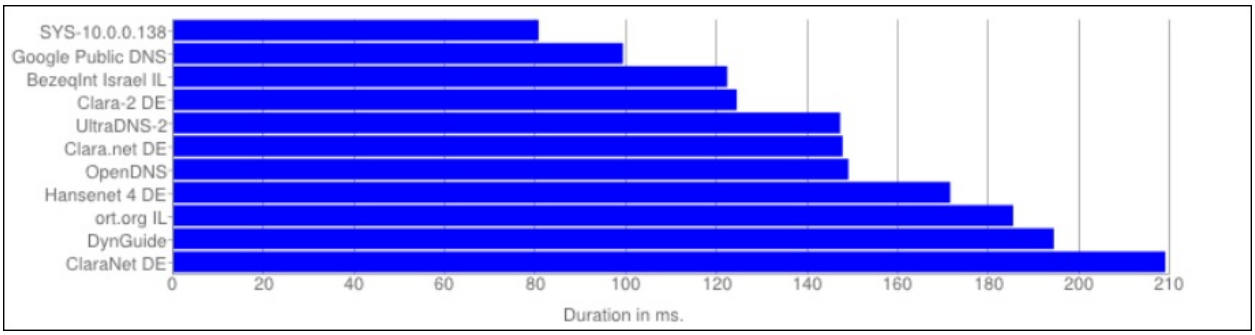
The most common codes are shown in the following table:

Error code	Name	What is it (RFC 1035)	Why it happens	What to do
`0`	No error condition	No error, everything works fine.	This signifies that everything is working.	Be happy.
`1`	Format error	The DNS server couldn't interpret the query.	This error code is usually shown when the DNS server does not support DNS extensions, for example, `EDNS0` (`RFC 2671`).	In most cases, there is nothing to do. The DNS request will be sent again without the extension. If the problem still exists, change the DNS server.
`2`	Server failure	The DNS server was not able to process the query due to a problem with the name server.	This error code signifies that there is a problem in the DNS server.	Configure another DNS server and check again.
`3`	Name error	This is meaningful only for responses that are coming from authoritative name servers.	This error code signifies that the domain name requested in the query does not exist.	Check the domain name.
`4`	Not Implemented	The DNS server does not support the requested type of query.		
`5`	Refused	The DNS server refuses to perform the specified	A name server may not wish to provide the information to the particular requester.	This occurs due to connectivity problems, if the forward DNS is not configured, or if there is a

		operation due to policy reasons.	A name server may not wish to perform a particular operation.	problem in one of the DNS servers on the way.

There's more...

What DNS server should I configure? I have been asked this question many times. My answer to this is simple—a server that is physically close to you (that is, not an overseas server), and one that you know is efficient. An efficient server, that is, overseas will give slow responses due to the communication lines, and a nearby non-efficient server will also give you slow response times.

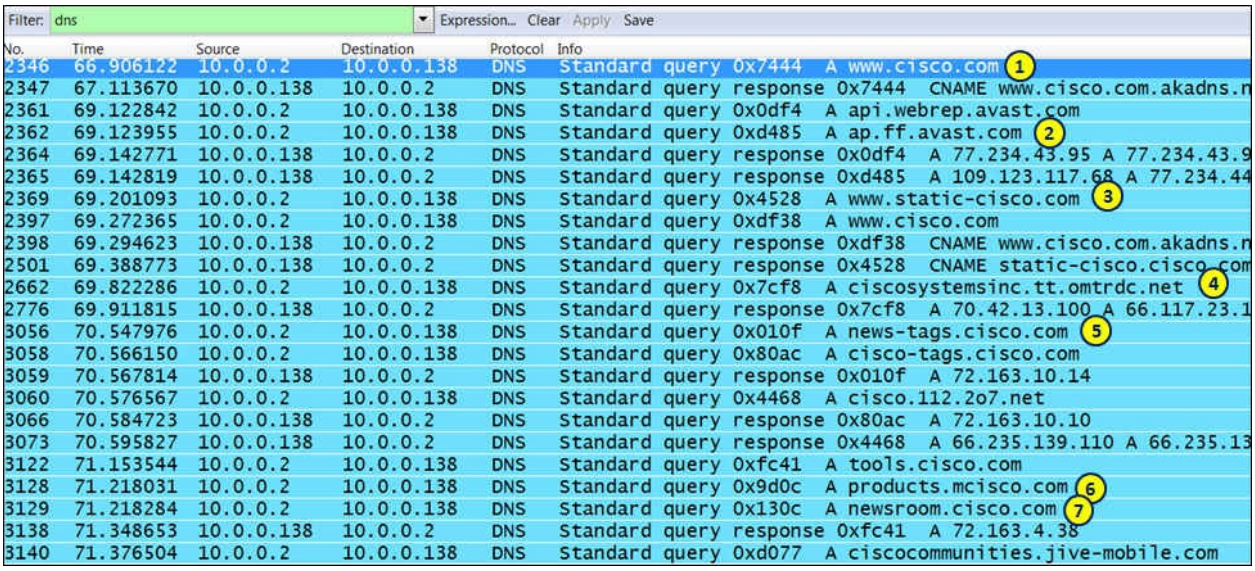


In the preceding graph, we see a measurement taken with the **Google Namebench** open software (freeware). It shows the following details:

- Average DNS response time of 80 ms to our local DNS server (you can see it is local from the unregistered address `10.0.0.138`)
- Average response time of 100 ms to the DNS server of my ISP
- Response times of 120 ms and above to the servers located overseas

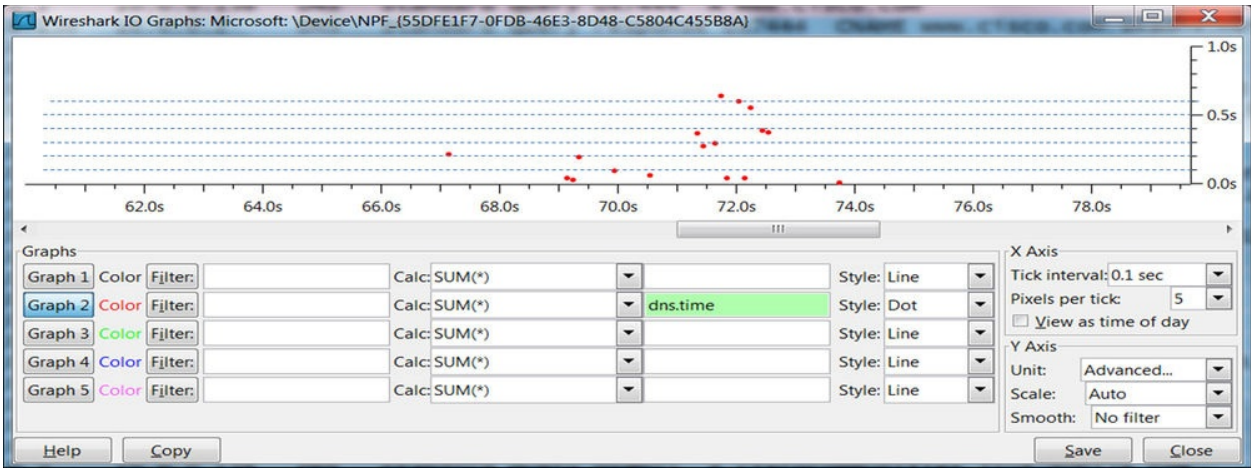
To summarize this, it is OK to have response times of around 100 ms; and in most of the cases, 150-200 ms will also be good enough. Don't worry if there are momentary peaks—it can be that your resolver is querying authoritative servers on the other side of the globe.

When you open a web page that holds a lot of content, your browser can send even tens of DNS queries. In the following screenshot, you see what happens when I open the browser to www.cisco.com.



It starts with a DNS query to the `A` record of www.cisco.com (marked as **1** in the preceding screenshot), then a query to `ap.ff.avast.com` (marked as **2** in the preceding screenshot), which is the web shield server of Avast antivirus, to `www.static-cisco.com` (marked as **3** in the preceding screenshot), `ciscosystems.tt.omtrdc.net` (marked as **4** in the preceding screenshot), news (marked as **5** in the preceding screenshot), products (marked as **6** in the preceding screenshot), and newsroom (marked as **7** in the preceding screenshot) sites.

When we look at the response time graph (shown in the next screenshot), we see that the DNS response times are up to 600 ms. This explains why it took a few seconds to open the entire web page of Cisco.


Wireshark IO Graphs: Microsoft: \Device\NPF_{55DFE1F7-0FDB-46E3-8D48-C5804C455B8A}
1.0s
0.5s
0.0s
62.0s
64.0s
66.0s
68.0s
70.0s
72.0s
74.0s
76.0s
78.0s
Graphs
Graph 1 Color Filter: Calc: SUM(*) Style: Line
Graph 2 Color Filter: Calc: SUM(*) dns.time Style: Dot
Graph 3 Color Filter: Calc: SUM(*) Style: Line
Graph 4 Color Filter: Calc: SUM(*) Style: Line
Graph 5 Color Filter: Calc: SUM(*) Style: Line
X Axis
Tick interval: 0.1 sec
Pixels per tick: 5
View as time of day
Y Axis
Unit: Advanced...
Scale: Auto
Smooth: No filter
Help
Copy
Save
Close

Filtering HTTP traffic

There are many filters that can be configured for HTTP. In this recipe, I will concentrate on the display filters that are mostly used in this context.

Getting ready

Configure port-mirror as described in previous recipes, and take a quick look at Chapter 2, *Using Capture Filters*.

How to do it...

To configure HTTP filters, you can write the filter expression directly in the display window bar; open the expression window and choose the HTTP parameters by right-clicking on the required parameter in the packet pane (as described in Chapter 3, *Using Display Filters*).

There are various filters that can be configured on HTTP:

- Name-based filters
  - Requests to a specific website: `http.host == "www.packtpub.com"`
  - Requests to the websites containing the word PacktPub: `http.host contains "packt.pub"`
  - Requests that were forwarded from PacktPub: `http.referer == "http://www.packtpub.com/"`
- Request methods filters
  - All GET requests: `http.request.method == GET`
  - All HTTP requests: `http.request`
  - All HTTP responses: `http.response`
  - All HTTP requests that are not GET: `http.request and not http.request.method == GET`
- Error codes filters
  - HTTP error responses (code `4xx` for client errors, code `5xx` for server errors): `http.response.code >= 400`
  - HTTP client error responses: `http.response.code >= 400 and http.response.code <= 499`
  - HTTP server error responses: `http.response.code >= 500 and http.response.code <= 599`
  - HTTP response code `404` (not found): `http.response.code == 404`

Tip

When you configure a simple filter such as `http.host == packtpub`, you don't need to close it in the `""` characters. If you need a more complex string such as `packtpub\r\n`, or a string of several words, then you will need to close it in `""`, for example, `"http.host == packtpub\r\n"`.

How it works...

Let us see some details on HTTP.

HTTP methods

The main HTTP requests methods were published in RFCs 2616. There are additional HTTP methods that were standardized over the years. Additional methods were added later by updates to RFC 2616 (2817, 5785, 6266, and 6585) and additional standards (RFC 2518, 3252, 5789).

These are the basic methods as described in RFC 2612:

- **OPTIONS**: This is used for client request to determine the capabilities of a web server.
- **GET**: This is used when we request a URL.
- **HEAD**: This is like `GET`, but the server should not return a message body in the response.
- **POST**: This is used to send data to the server. For example, when using webmail, it will be used to send e-mail commands.
- **DELETE**: This is used to request the server to delete a resource identified by the `Request-URI`.
- **PUT**: This is used to request that the enclosed entity be stored under the Request-URI attached to the request.
- **TRACE**: This is used to request a remote, application-layer loopback of the request message.
- **CONNECT**: This is used to connect to a proxy device.

Status codes

These are the categories of message codes that are standardized by HTTP:

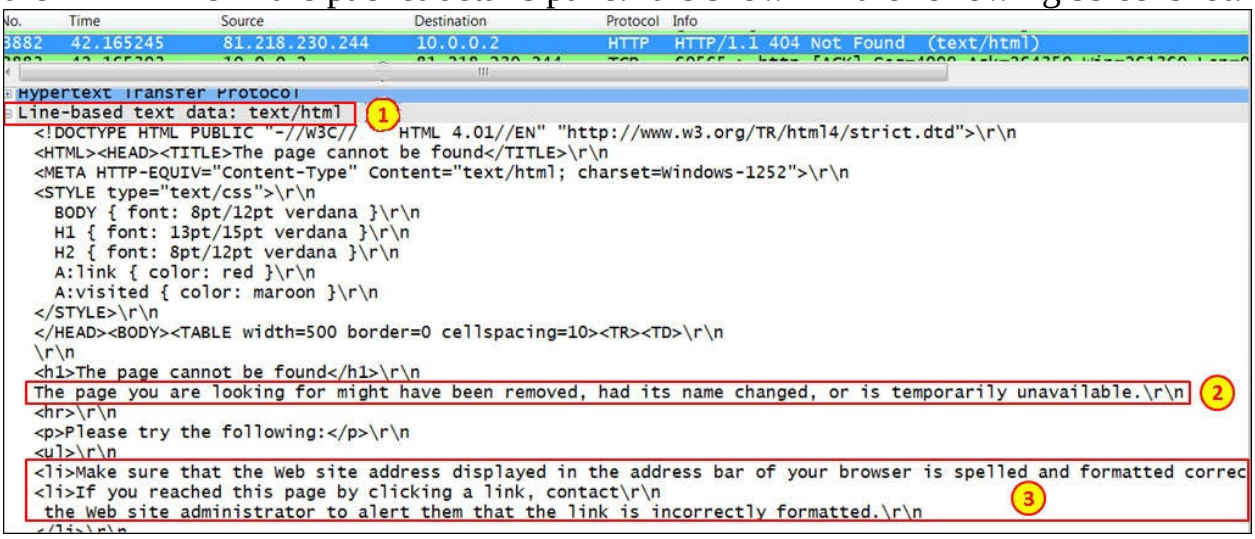
Category	Category name	What is it for
`1xx`	Informational	Provides general information, without any indication of failure or success
`2xx`	Success	Indicates that the action requested by the client was received, accepted, and processed successfully

3xx	Redirection	Indicates that further action should be taken by the user agent to fulfill the request
4xx	Client error	Indicates a client error
5xx	Server error	Indicates a server error

A full list of HTTP status codes can be found at http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml.

There's more...

In some cases, you will see a line called **Line-based text data: text/html** under the HTTP line in the packet details pane. It is shown in the following screenshot:



You will see the **Line-based text data** right beneath to the HTTP line in the packet details pane (marked as **1** in the preceding screenshot). Below this, you will see some explanations (marked as **2** and **3** in the preceding screenshot) for what could be the reason for the error.

Configuring HTTP preferences

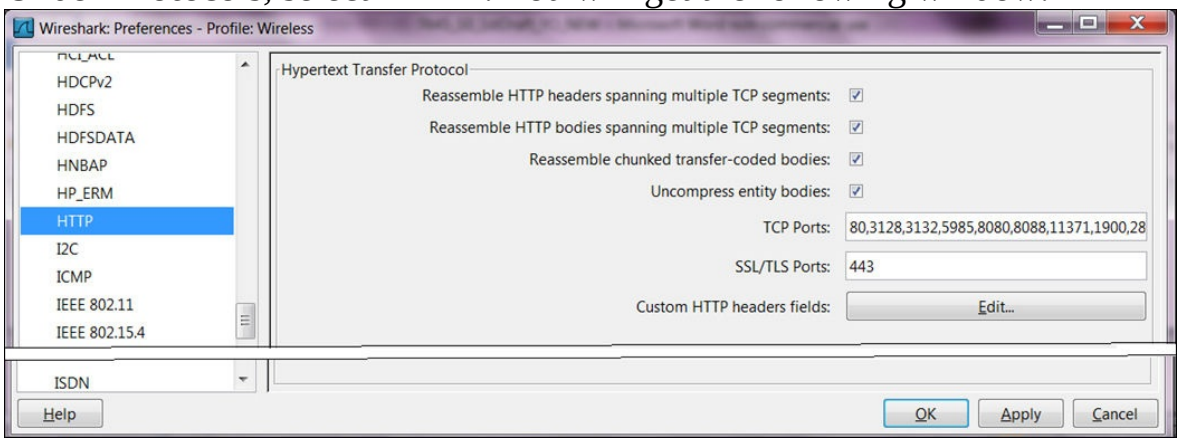
There are some preferences that you can change when working with HTTP. Let's see what they are.

Getting ready

Start Wireshark and go to the next section.

How to do it...

1. Choose **Edit** | **Preferences**.
2. Under **Protocols**, select **HTTP**. You will get the following window:

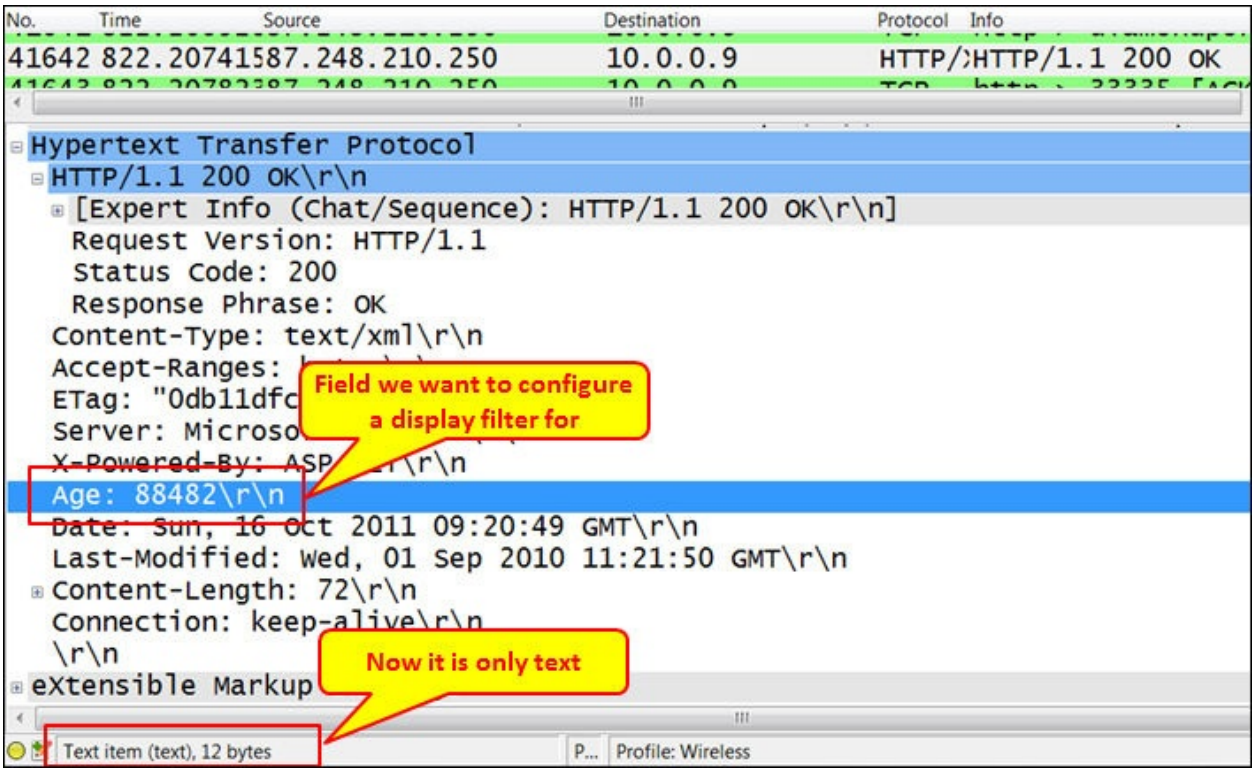


   - By default, the upper four rows are checked. These are options that reassemble the HTTP headers and body when fragmentation is performed on the lower layers.
   - In the **TCP Ports** field, you will get a list of the port numbers that Wireshark will dissect as HTTP. In this list, you see the default port `80`, ports `8080` and `8088` that are usually used for proxies, and others. In case you have an application working with HTTP with a port that is not listed, add it here.
   - The same with HTTPS—the default is `443` (that is for Secured HTTP, or HTTP over SSL/TLS). In case you use another port, add it here.

Custom HTTP headers fields

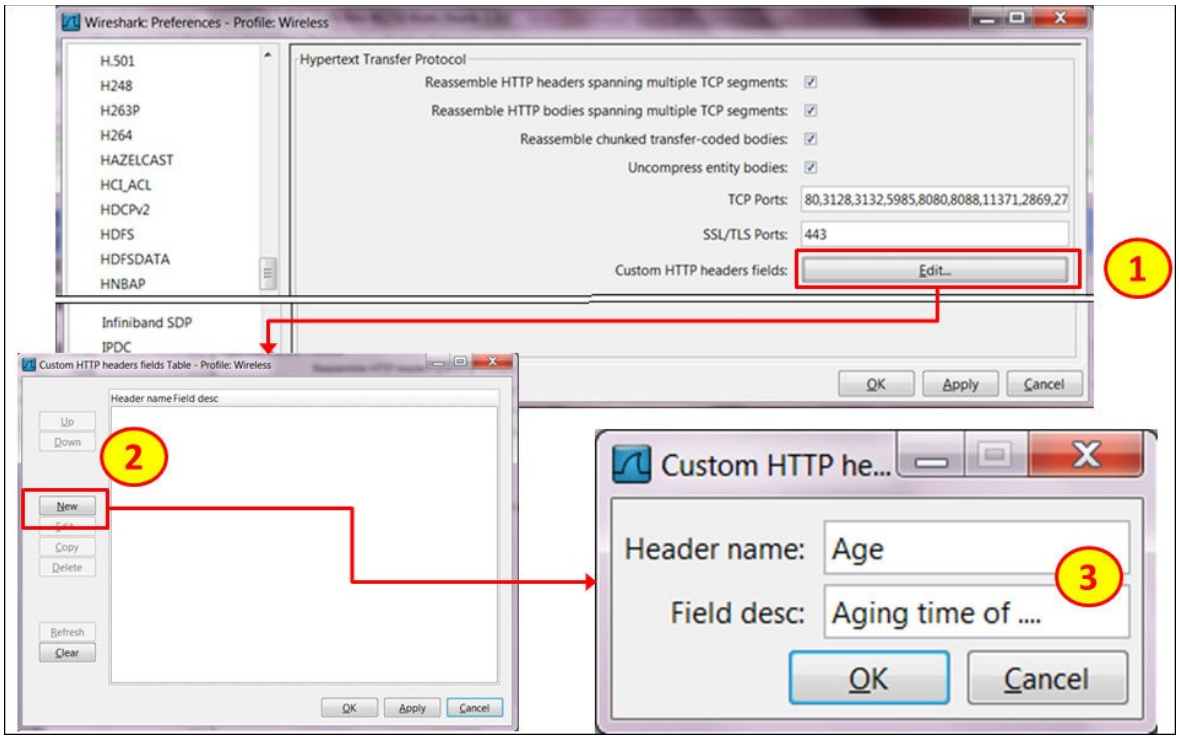
Custom HTTP headers fields enable us to create a new HTTP display filters under the `http.header` filter.

Let's look at the example in the following screenshot:



For creating a new HTTP display filter under the `http.header` filter, perform the following steps:

1. In the HTTP preferences window (marked as **1** in the following screenshot), click on the **Edit** button in **Custom HTTP headers fields**.



2. Click on **New** (marked as **2** in the preceding screenshot).
3. In **Header name**, enter the name of the filter to be used in extension to `http.header` (marked as **3** in the preceding screenshot).

   For example, if you want to configure a filter on the `Age` parameter, type the name `Age` in the **Header name** field (case sensitive!).
4. In the **Field desc** field, type any description that will remind you what you have configured.

   For example, type **Aging time of …** (any description will do, it is just a note).
5. Click on **OK**.
6. In the **Display Filter** textbox, you will be able to use the `http.header.Age` filter.

   For example, you will be able to configure the display filter `http.header.Age` that contains `88482` that will give you all the packets with the `Age` field that contains the requested number

Tip

This filter configuration is mostly used when you are using proprietary parameters in the HTTP header, and you want to filter accordingly.

- You can configure many additional filters with this option.

How it works...

The reassembly feature is important because there are some cases in which IP fragmentation is used, and therefore the TCP message is also segmented. Marking the reassembly options simply tells the Wireshark to reassemble the monitored packets (what the receiver side is doing and therefore is able to understand it).

There's more...

Usually Wireshark shows dissected packets with port `80` as HTTP only if it sees a valid HTTP header. If you want to see all port `80` packets as HTTP, perform the following steps:

1. Go to **Preferences**, and choose **TCP** in **Protocols**.
2. Uncheck/disable **Allow dissector to reassemble TCP streams**.

Analyzing HTTP problems

The bottom line is, of course, how to analyze the HTTP problems. This is what this recipe is all about. HTTP problems can happen because of a slow server or client, TCP performance issues, and some other reasons that we will see in this recipe.

Getting ready

When you experience bad performance while browsing the Internet, connect the Wireshark with port mirror to the PC that experiences the problem, and when it is the whole network that suffers from bad performance, port mirror the connection to the Internet.

How to do it...

There can be various reasons for a slow browsing problem, and we'll try to figure it out step-by-step. The steps are given as follows:

1. First, check that you don't simply have loaded line to the Internet, high error rate on the communications line, or any of these obvious issues that cause most of the problems (see Chapter 4, *Using Basic Statistics Tools* and Chapter 5, *Using Advanced Statistics Tools* for further details).
2. To negate a TCP issue (as explained in detail in Chapter 9, *UDP/TCP Analysis*), check the following details:
    - In the **Expert info** window, you don't get too many retransmissions and duplicate ACKs (up to 0.5-0.8 percent is still tolerable).
    - Make sure that you don't get resets on the HTTP connections. It might be due to firewalls or site restrictions.

- Make sure that you don't get the following DNS problems:
    - Slow response time, as described earlier in this chapter
    - Names are not found, not correct, and so on
- If none of these apply, well, let's dig in to HTTP.

Tip

Don't forget to look at the network and IT environment as a whole. You cannot separate TCP from HTTP, or the DNS problems from the slow browsing of applications. It can be that you have a very slow HTTP server; and because of its slow responses, you will get TCP retransmissions. Or, because of the slow DNS server, you will get a web page that opens after many seconds. Just go step- by-step and isolate the problems.

When you open a web page for the first time, it can take a few seconds. In this case, you should check the following conditions:

1. Check if the line is not loaded.
2. Check the delay on the line (a ping to the website will do the job).
3. Look for error codes. Usually you will see the reason for the error on the browser, but not always.
4. Configure the filter `http.response >= 400` and see how many errors you

get. In the following sections, we see several examples of what you should pay attention to.

Informational codes

Code	Status	Explanation	What to do
`100`	Continue	Request completed successfully and the session can continue.	-
`101`	Switching protocols	The server is changing to a different HTTP version. It will be followed by an `Upgrade` header.	-

Success codes

Code	Status	Explanation	What to do
`200`	OK	Standard OK response.	-
`201`	Created	The request has been fulfilled and a new resource has been created.	-
`202`	Accepted	The request was accepted and is still in process.	-
`203`	Non-authoritative information	The request was received with content from another server, and it was understood.	-
`204`	No content	The request was received and understood, and the answer that is sent back has no content.	-
`205`	Reset content	This is a server request to the client to reset the data that was sent to it.	-
`206`	Partial content	Response for a partial document request.	-

Redirect codes

Code	Status	Explanation	What to do
`300`	Multiple choices	The requested address refers to more than one file. It can happen, for example, when the resource has been removed, and the response provides a list of potential locations for it.	-
`301`	Moved permanently	The requested resource has been moved permanently. Future requests should be forwarded to the attached URI.	-
`302`	Moved temporarily (found)	Page has been moved temporarily, and the new URL is available. Usually, you will be automatically forwarded.	Usually, you will see a `Found` code, and then another `GET` to the URL indicated
`303`	See other	The response to the request can be found in a different URI. It should be retrieved using an HTTP `GET` to that resource.	-
`304`	Not modified	When a request header includes an `if modified since` parameter, this code will be returned if the file has not changed since that date.	-
`305`	Use proxy	The requested resource must be accessed through a proxy.	Check what proxy is required

Client errors

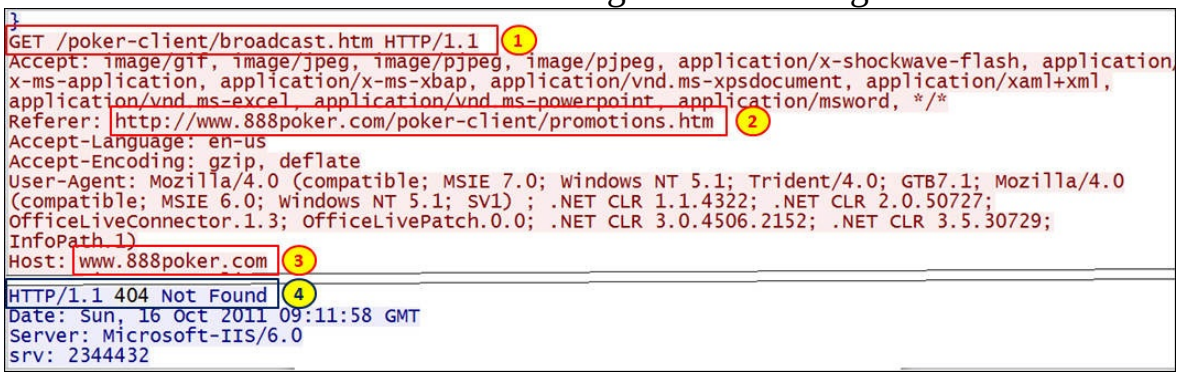
Code	Status	Explanation	What to do
`400`	Bad request	The request could not be understood by the server due to a syntax problem. The request should be modified by the client before resending to it.	Check the website address. This can also happen due to a site error.
`401`	Authorization required	The client is denied access due to the lack of authentication codes.	Check your username and password.
`402`	Payment required	Reserved for future use.	
`403`	Forbidden	The client is not allowed to see a specific	Check the credentials. Also, there are

		file. This can be due to the server access limit.	fewer chances that the server is loaded.
`404`	Not found	The requested resource could not be found.	This can be because the resource was deleted, or it never existed before. It can also be due to URL misspellings.
`405`	Method not allowed	The method you are using to access the file is not supported or not allowed by the resource.	
`406`	Not acceptable	Content generated by the resource is not acceptable according to the client request.	Check/update your browser.
`407`	Proxy authentication required	Request authentication is required before it can be performed.	The client must first authenticate itself with the proxy.
`408`	Request timed out	It took the server longer than the allowed time to process the request.	Check response time and load on the network.
`409`	Conflict	The request submitted by the client cannot be completed because it conflicts with some established rules.	Can be because you try to upload a file that is older that the existing one or similar problems. Check what the client is trying to do.
`410`	Gone	The URL requested by the client is no longer available from that system.	Usually this is a server problem. It can be due to a file that was deleted or location was forwarded to a new location.
`411`	Content length required	The request is missing its `Content-Length` header.	Compatibility issue on a website. Change/update your browser.
`412`	Precondition failed	The client has not set up a configuration that is required for the file to be delivered.	Compatibility issue on a website. Change/update your browser.
`413`	Request entity too long	The requested file was too big to process.	Server limitation.

414	Request URI too long	The address you entered was overly long for the server.	Server limitation.
415	Unsupported media type	The file type of the request is not supported.	Server limitation.

A simple example for a client error is presented in the following screenshot. To get to this window, perform the following steps:

1. Right-click on the packet with the error code.
2. Choose **Follow TCP stream**. You will get the following window:



- You can see the following conditions:
  - I tried to browse the URI `/poker-client/broadcast.htm` (marked as **1** and **3** in the preceding screenshot)
  - The URI was forwarded by the referrer: `http://www.888poker.com/poker-client/promotions.htm` (marked as **2** in the preceding screenshot)
  - The status code was **404 Not Found** (marked as **4** in the preceding screenshot)

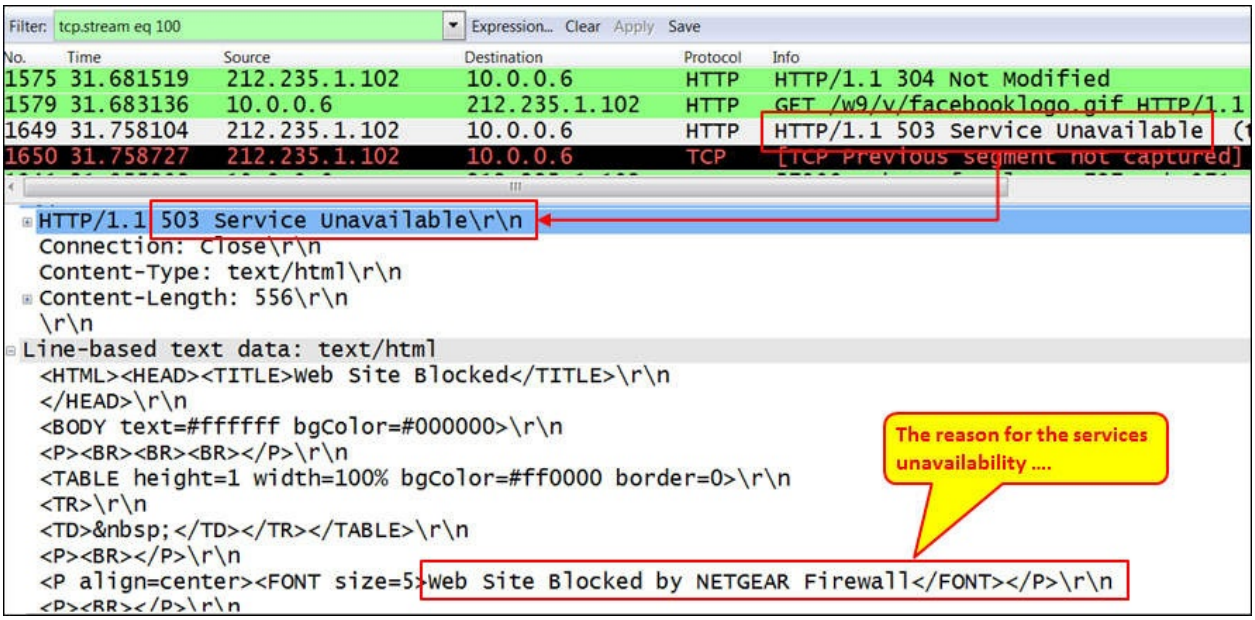
Just to clarify things, I was not playing Poker, I was working on a networking problem.

Server errors

Code	Status	Explanation	What to do

500	Internal server error	The web server encountered an unexpected condition that prevented it from carrying out the client request for access to the requested URL.	Response that is usually caused by a problem in your Perl code when a CGI program is run.
501	Not implemented	The request cannot be executed by the server.	A server problem.
502	Bad gateway	The server you're trying to reach is sending back errors.	A server problem.
503	Service unavailable	The service or file that is being requested is not currently available.	A server problem.
504	Gateway timeout	The gateway has timed out. This message is like the 408 timeout error, but this one occurs at the gateway of the server.	Server is down or nonresponsive.
505	HTTP version not supported	The HTTP protocol version that you want to use for communicating with the server is not supported by it.	Server does not support the HTTP version.

You can get service unavailable (code 503) status due to various reasons. In the following example there is a small office that has the following complaint: they can browse Facebook, but the moment they click on a link on this site, they get the new page as blocked. In the following screenshot, you can see that the problem was simply a firewall that blocked it (obviously).


Filter: tcp.stream eq 100
Expression... Clear Apply Save
No. Time Source Destination Protocol Info
1575 31.681519 212.235.1.102 10.0.0.6 HTTP HTTP/1.1 304 Not Modified
1579 31.683136 10.0.0.6 212.235.1.102 HTTP GET /w9/v/facebooklogo.gif HTTP/1.1
1649 31.758104 212.235.1.102 10.0.0.6 HTTP HTTP/1.1 503 Service Unavailable
1650 31.758727 212.235.1.102 10.0.0.6 TCP [TCP Previous segment not captured]
HTTP/1.1 503 Service Unavailable\r\n
Connection: Close\r\n
Content-Type: text/html\r\n
Content-Length: 556\r\n
\r\n
Line-based text data: text/html
<HTML><HEAD><TITLE>Web Site Blocked</TITLE>\r\n
</HEAD>\r\n
<BODY text=#ffffff bgColor=#000000>\r\n
<P>

</P>\r\n
<TABLE height=1 width=100% bgColor=#ff0000 border=0>\r\n
<TR>\r\n
<TD> </TD></TR></TABLE>\r\n
<P>
</P>\r\n
<P align=center><FONT size=5>Web Site Blocked by NETGEAR Firewall</FONT></P>\r\n
The reason for the services unavailability ....

How it works...

In standard HTTP browsing, you should see a very simple pattern as follows:

1. TCP opens the connection (three-way handshake).
2. HTTP sends a `GET` command.
3. Data is downloaded to your browser.

Tip

In most cases, opening a web page will open multiple connections—in many cases, tens of them. For example, when you open a newspage (www.cnn.com, www.foxnews.com, www.bbc.co.uk), it opens the main page, news bars, commercials, temperature window, connections to other sites, and more. Don't be surprised if a single page will open nearly a hundred connections, or even more.

In case of a web page that opens multiple connections (as most web pages do), each connection requires a DNS query, response, TCP SYN-SYN/ACK-ACK, and HTTP GET; only then the data will start to appear on your screen.

There's more...

When you don't see anything in the packet details pane, right-click on a packet and choose **Follow TCP stream**. This will give you a detailed window, (as in the preceding screenshot) which provides you with a lot of data for the connection.

Another tool that is widely used for HTTP is **Fiddler**. It can be found at http://fiddler2.com/. Fiddler is a free tool that is planned for HTTP debugging. It is not in the scope of this book.

Exporting HTTP objects

Exporting HTTP objects is a simple feature for exporting HTTP statistics—websites and files accessed by HTTP.

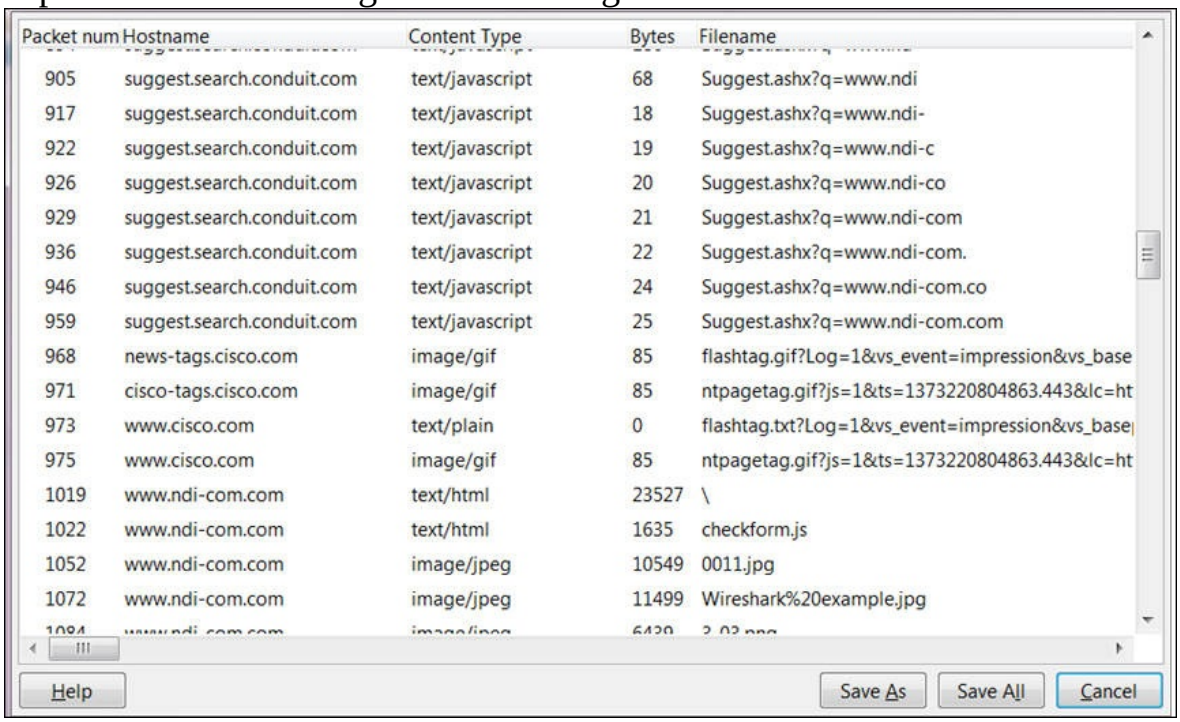
Getting ready

To export HTTP objects, choose **File** | **Export Objects** | **HTTP**.

How to do it...

To export HTTP objects, follow these steps:

1. You can use this feature when capture is running, or you can save the captured file. You will get the following window:



- From here you can get a list of the websites that were accessed, including the files that were accessed in each one of them. You can see the website, file types, size, and names.
- You can use the **Save As** or **Save All** buttons for saving the data in a file.
- In the **Content Type** column, you will see the following contents:
  - Text: **text/plain**, **text/html**, **text/javascript**—if it's a JavaScript, check what it is, it might be a security risk
  - Images: **image/jpeg**, **image/gif**, and other types of images—you can open it with a viewer
  - Applications: **application/json**, **application/javascript**, and other types of applications
  - Any other text file discovered by Wireshark

Tip

For the export HTTP objects feature to work, first go to TCP preferences and enable TCP packets reassembly (allow subdissector to reassemble TCP streams).

You will get a directory with all the objects captured in the capture file. Objects can be pictures (for example, packet `1052` and `1057` in the preceding screenshot), text (packets `1019`, `1022`, and others in the preceding screenshot), and others.

How it works...

This feature scans HTTP streams in the currently opened capture file or the running capture, takes reassembled objects such as HTML documents, image files, executable files, and other readable formats, and lets you save them to a disk. The saved objects can then be opened with the proper viewer, or they can be executed in the case of executable files just by clicking on them. This feature can be helpful for various purposes, including eavesdropping and saving objects for backup (for example, files that were sent through e-mails).

There's more...

You have several pieces of software that perform the same things graphically, some of them are as follows:

- **Xplico** (http://www.xplico.org/)
- **NetworkMiner** (http://www.netresec.com/?page=NetworkMiner)

Tip

When you see an unknown website with an application that you don't know, and a filename that looks suspicious—Google it; it might be a risk (we will get back to this in Chapter 14, *Understanding Network Security*).

HTTP flow analysis and the Follow TCP Stream window

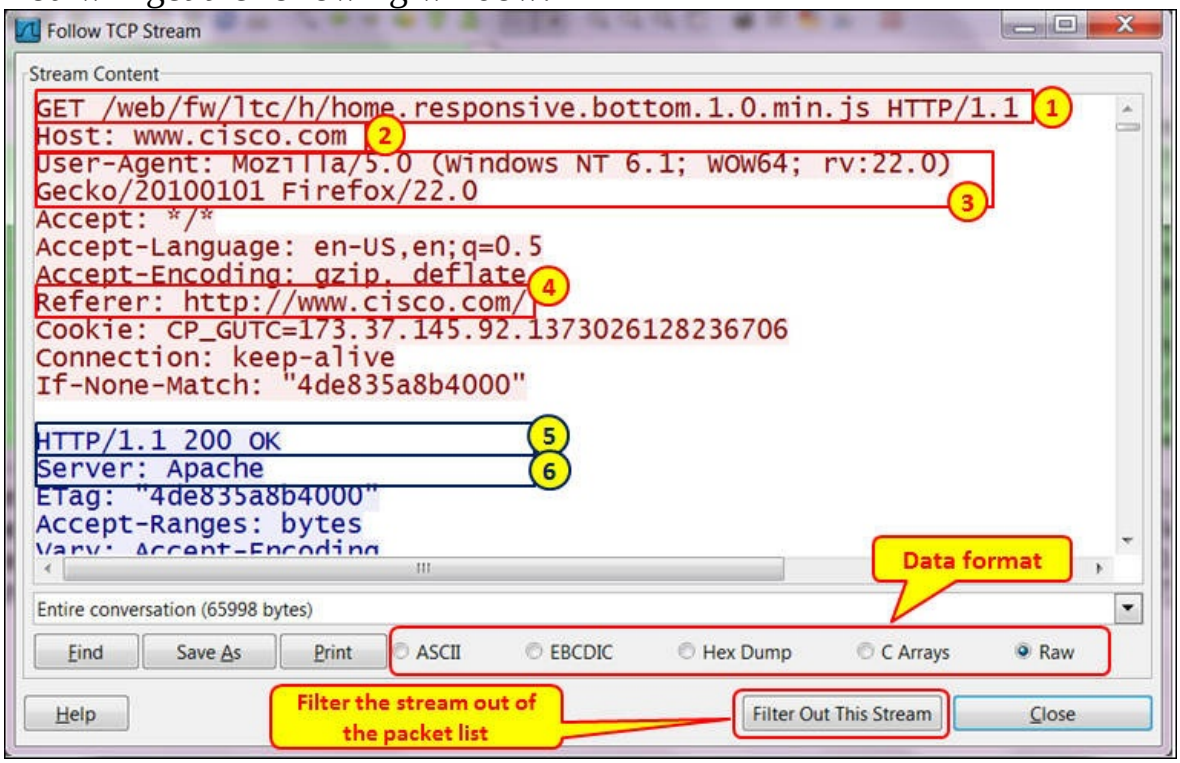
The **Follow TCP Stream** feature that was discussed in brief earlier in the book is a very helpful feature that can help you with in-depth understanding of the TCP flows that are captured when you monitor the network. In this recipe, we will see some of its advantages.

Getting ready

Port mirror the device or link you want to monitor and start packet capture.

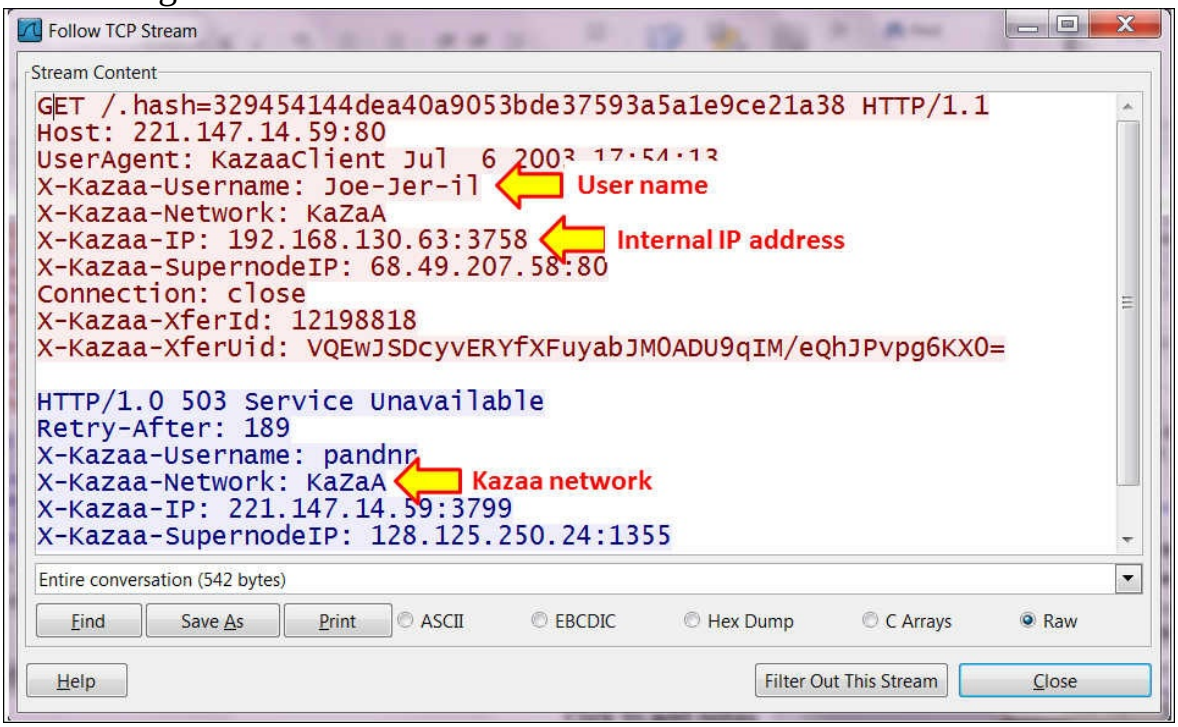
How to do it...

1. For opening the **Follow TCP Stream** window, perform the following steps:
2. Right-click on one of the packets in the stream you want to view.
3. The stream you choose is filtered by the Wireshark. You will see this in the display filter bar that will show you the number of stream in the capture. You will get the following window:

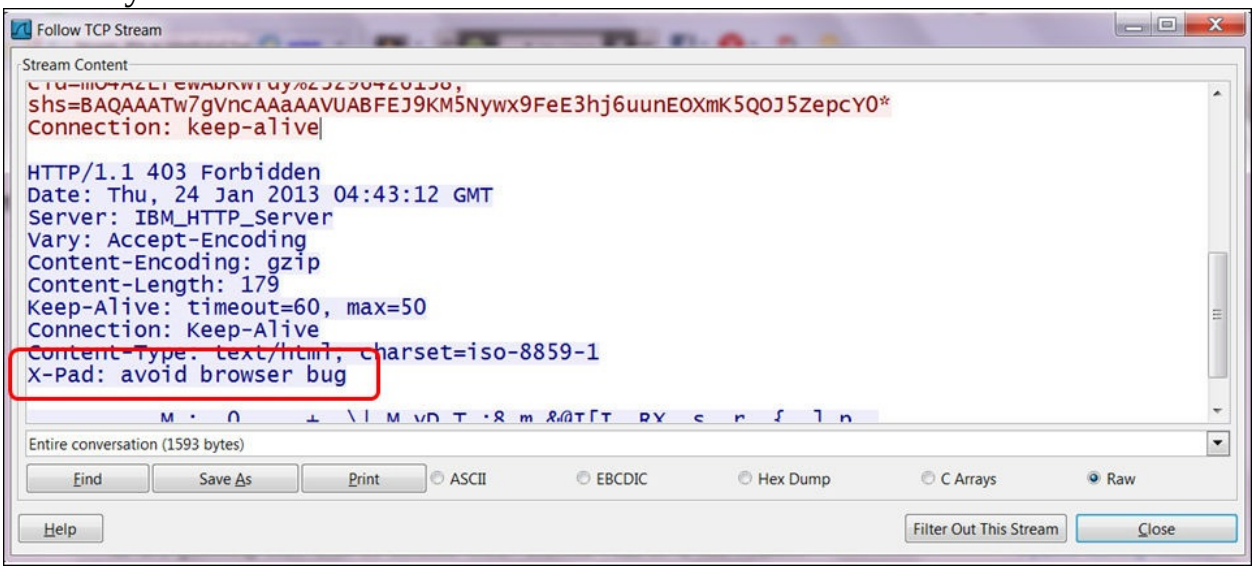


- You can see the stream details, for example:
  - The `GET` method (marked as **1** in the preceding screenshot)
  - The requested `HOST` (marked as **2** in the preceding screenshot)
  - The client type, Mozilla Firefox in this case, (marked as **3** in the preceding screenshot)
  - The referrer, Cisco in this case, (marked as **4** in the preceding screenshot)
  - The HTTP OK response (marked as **5** in the preceding screenshot)
  - The server type (marked as **6** in the preceding screenshot)
- These are obvious examples. When having problems, or just issues to investigate, you will be able to see many types of parameters here that will indicate the following cases:
  - A user is using a Kazaa client (as shown in the following screenshot) for

file sharing.



- In the following screenshot, you can see a software bug. A quick Google search shows that it is an historical one, but other bugs can be found this way.



- You can also check for the following:
  - Error and bugs messages
  - Viruses and worms—names such as blast, probe, and Xprobe, especially

when you see them with `.exe` extension should ring a big warning bell (more details about this issue will be provided in Chapter 14, *Understanding Network Security*)

How it works...

The Follow TCP Stream simply analyzes the TCP data from the first SYN-SYN/ACK/ACK handshake to the end of the connection, which is indicated by RST of the FIN packets. It also isolates the specific stream, helping us to follow the errors and problems in it.

There's more...

There are many problems that can be found and allocated using the Follow TCP Stream feature, and it will be discussed further in the next chapters. Use this feature to isolate a TCP stream.

Analyzing HTTPS traffic – SSL/TLS basics

HTTPS is a secure version of the HTTP. The "S" means that it is secured by **Secure Socket Layer**/**Transport Layer Security** (**SSL**/**TLS**). It is used when you connect to your bank account, webmail service, or any other service that runs over HTTP and requires security.

In this recipe, we will see how it works and what can fail when we are using HTTPS communications.

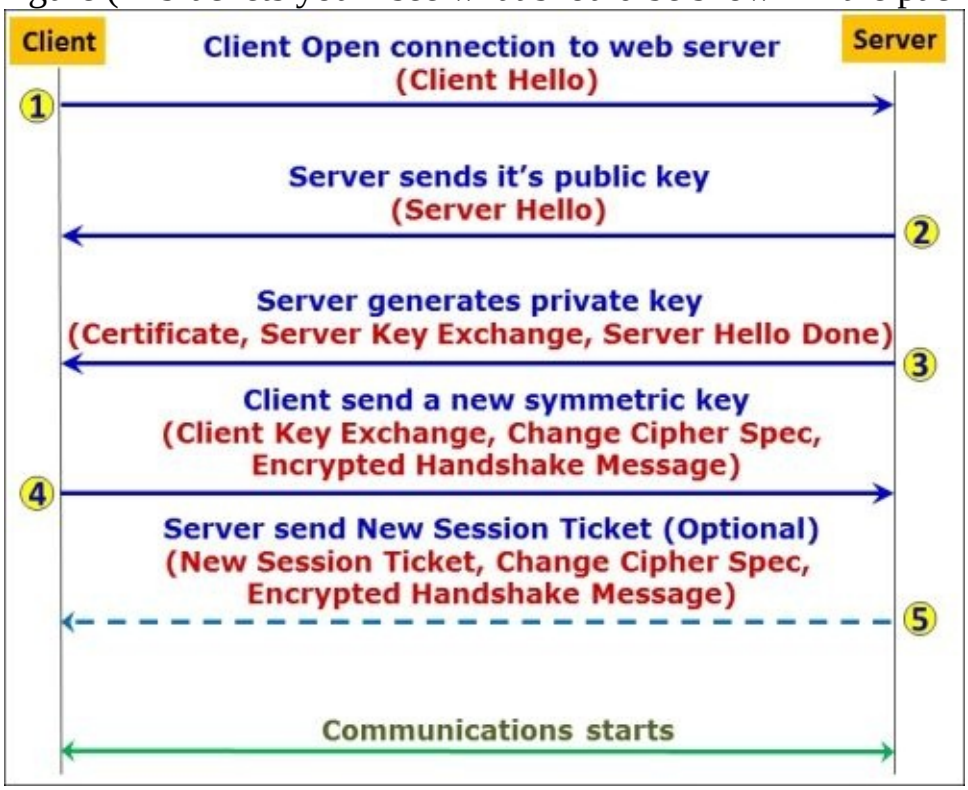
Getting ready

Port mirror to the suspected device or link that forwards traffic from several devices, and start capture. HTTPS works with the TCP port `443`, and this is what we should watch.

How to do it...

To monitor HTTPS sessions, perform the following steps:

1. HTTPS session establishment can be done in four or five steps. It is described in the *How it works...* section of this recipe.
2. Watch the order of the packet in the session establishment, and make sure the messages you get are according to the order shown in the following figure (in brackets you'll see what should be shown in the packet):



- There are some common errors that are described in `RFC 2246`:
  - **close_notify**: This message notifies the recipient that the sender has finished sending messages on this connection. The session can be resumed later.
  - **unexpected_message**: This alert is returned if an inappropriate message was received. This is a critical error that can indicate a bad implementation on one of the sides.
  - **bad_record_mac**: This alert is returned if a record is received with incorrect Message Authentication Code (MAC). This is a critical error that

can indicate a bad implementation on one of the sides.

- **decryption_failed**: This alert is returned if a TLS Ciphertext was decrypted in the wrong way. This is a critical message that can indicate a bad implementation on one of the sides.
- **record_overflow**: This alert is returned if a TLS Ciphertext record was received with a length longer than the allowed length. This is a fatal error, and it usually indicates a bad implementation on one of the sides.
- **decompression_failure**: This message indicates that a decompression function received a wrong input. This is a critical error that can indicate a bad implementation on one of the sides.
- **handshake_failure**: Reception of this alert message indicates a negotiation error that occurred when the sender was unable to negotiate the set of security parameters, given the options available. This is a critical error that can indicate a bad implementation on one of the sides.
- **bad_certificate**: This is a certificate error. It occurs when a certificate is corrupt, contains signatures that were not verified correctly, or any other error.
- **unsupported_certificate**: This indicates that the received certificate was not of the supported type.
- **certificate_revoked**: This indicates that a certificate was canceled by its signer.
- **certificate_expired**: This indicates an invalid certificate or a certificate that has expired.
- **certificate_unknown**: This tells that a certificate was not accepted due to an unspecified reason.
- **illegal_parameter**: This tells that a field in the handshake process was out of range or inconsistent with other fields. This is a critical error that can indicate a bad implementation on one of the sides.
- **unknown_ca**: This indicates that a valid certificate was received, but was not accepted because it couldn't be matched with a known, trusted CA. This is a critical error, and should be checked with the certificate issuer.
- **access_denied**: This tells that a valid certificate was received, but it was not approved by the access control of the receiver, and the sender decided not to proceed with negotiation.
- **decode_error**: This tells that a message was too long and, therefore, could not be decoded. This is a critical error that can indicate a bad implementation on one of the sides.

- **decrypt_error**: This indicates that a handshake cryptographic operation failed, including the ones that failed due to signature verification, key exchange, or validation of a finished message.
- **export_restriction**: This tells that a negotiation which is not compliant with export restrictions was detected.
- **protocol_version**: This tells that the protocol version which the client has attempted to negotiate is not supported.
- **insufficient_security**: This is returned when a negotiation has failed because the server required ciphers with higher security than those supported by the client.
- **internal_error**: This is an internal error not related to the peer of the connection.
- **user_canceled**: This tells that the handshake was canceled for a reason other than a protocol failure.
- **no_renegotiation**: This is sent by the client or the server in response to a hello request after the initial handshaking.

In each one of the failures mentioned, the connection will not be established.

How it works...

SSL and **TLS** are protocols that secure a specific application, for example, HTTP, SMTP, Telnet, and others. SSL Versions 1, 2, and 3 were developed by Netscape in the mid 90s for their Navigator browser, while TLS is a standard from the IETF (`RFC 2246`, `RFC 4492`, `RFC 5246`, `RFC 6176`, and others). TLS 1.0 was first introduced in `RFC 2246` in January 1999 as an upgrade of the SSL Version 3.0 (third paragraph at http://tools.ietf.org/html/rfc2246).

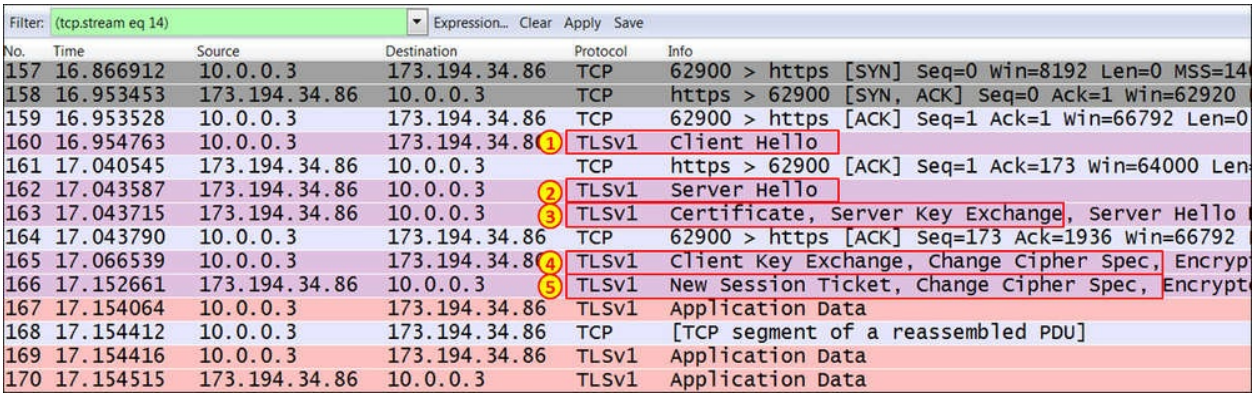
The TLS handshake protocol involves the following procedures for establishing a TLS connection:

1. Exchange hello messages to agree on the algorithms to work with, and exchange random values for the key generation.
2. Exchange the necessary cryptographic parameters to allow the client and the server to agree on a premaster secret key.
3. Exchange certificates and cryptographic information to allow the client and server to authenticate each other.
4. Generate a master secret key from the premaster secret and exchanged random values.
5. Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without being tampered with by an attacker.

These procedures are performed in the following order:

1. Select cryptographic algorithms:
    - The **Client Hello** message (marked as **1** in the following screenshot)
    - The **Server Hello** message (marked as **2** in the following screenshot)

- Authenticate the server and exchange key (marked as **3** in the following screenshot).
- Authenticate the client and exchange key (marked as **4** in the following screenshot).
- Complete the handshake (marked as **5** in the following screenshot).



Let's see how it works. In the preceding screenshot, we see how TCP SSL/TLS establishes a connection (packets `157-158-159`) and packet `160` starts the TLS handshake. Let us see the details:

1. In packet `160`, the client sends a **Client Hello** message that starts the negotiation.
2. The server answers with a **Server Hello** message.
3. The server sends a certificate to the client.
4. The client takes the certificate and generates a premaster key.
5. The server creates the master key, and the conversation begins. This is an optional message.

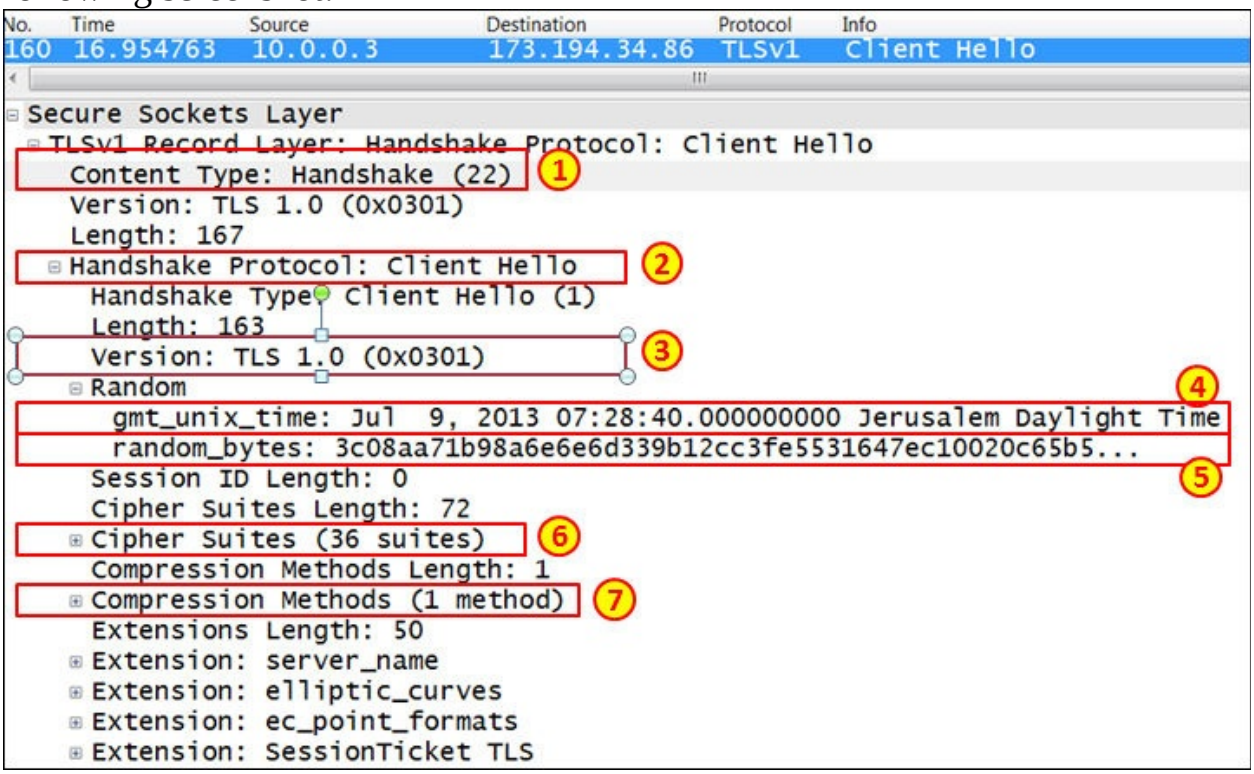
Tip

This refers to a mechanism (defined in `RFC 4507`) that enables the TLS server to resume sessions and avoid keeping the per-client session state. The TLS server encapsulates the session state into a ticket and forwards it to the client. The client can subsequently resume a session using the obtained ticket. This happens, for example, when you re-open a connection to your webmail account (Gmail, Hotmail, and so on) and is common to these scenarios.

Communication between the client and the server will start after step 4 or 5.

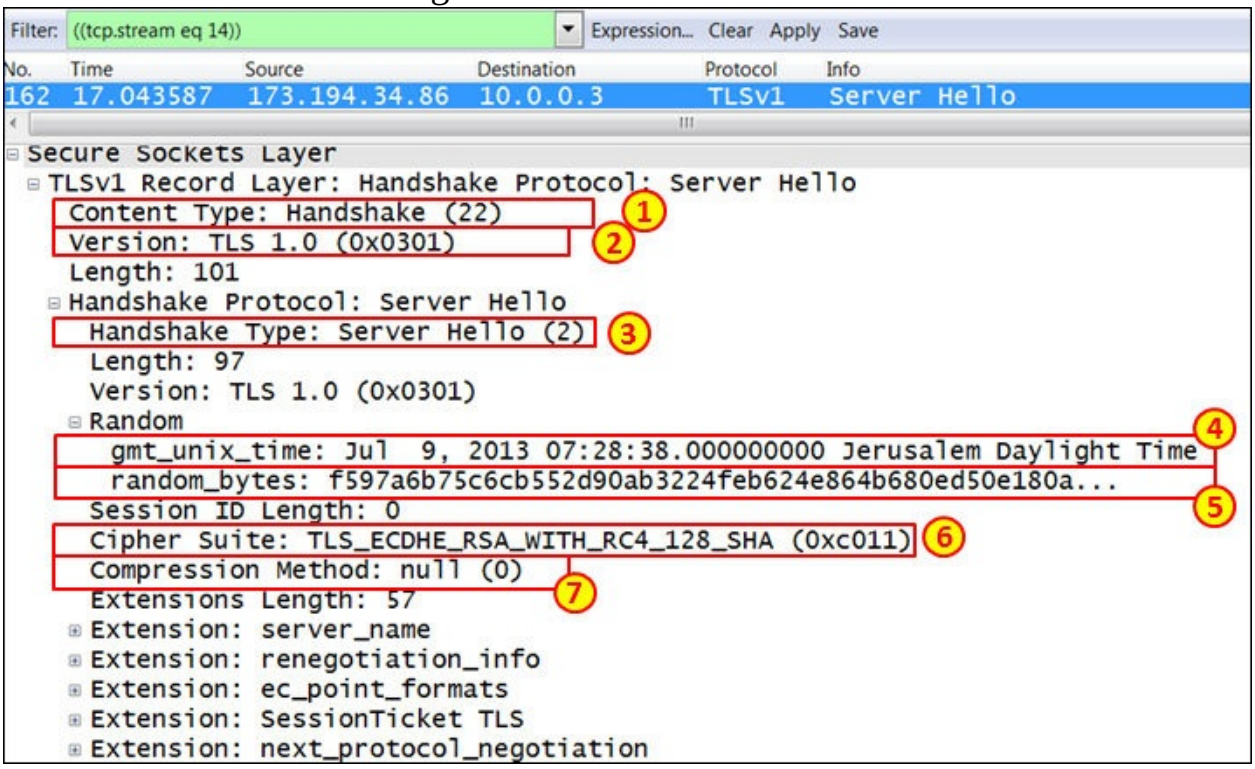
Let's look at each one of them:

In step 1, packet `160` is a **Client Hello** message which is the first packet in the TLS handshake. Some of the parameters that we can see are shown in the following screenshot:



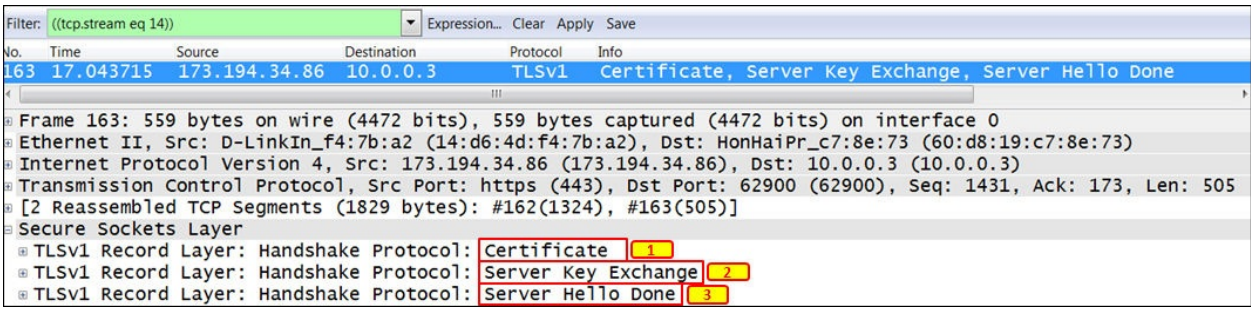
- The area highlighted as **1** shows that the content of the packet is a handshake (`ssl.record.content_type == 22`).
- The area highlighted as **2** shows that the packet is a **Client Hello** message sent from the client to the web server. This message starts the handshake.
- The area highlighted as **3** shows the highest SSL and TLS version supported by the client.
- The area highlighted as **4** shows the client time that will be used in the key generation process.
- The area highlighted as **5** shows the random data that is generated by the client for use in the key generation process.
- The area highlighted as **6** shows the ciphers supported by the client. The ciphers are listed in order of preference.
- The area highlighted as **7** shows the data compression methods that are supported by the client.

As shown in the following screenshot, Packet `162` is a **Server Hello** message, which includes the following details:



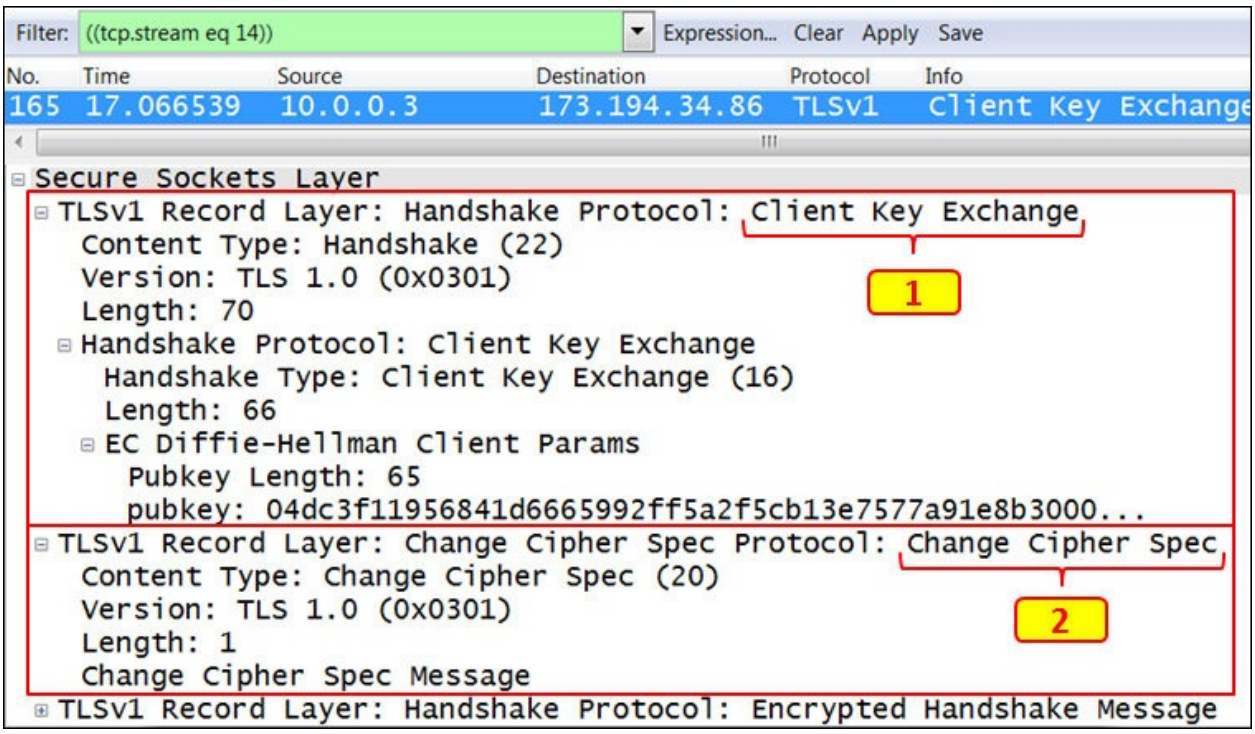
- The area highlighted as **1** shows that the content of the packet is a handshake (`ssl.record.content_type == 22`).
- The area highlighted as **2** shows the TLS version that will be used in this session.
- The area highlighted as **3** shows that the packet is a **Server Hello** message sent from the server to the client.
- The area highlighted as **4** shows the server time used in the key generation process.
- The area highlighted as **5** shows the random data that is generated by the server for use in the key generation process.
- The area highlighted as **6** shows the cipher suite to be used in this conversation. It is chosen from the list of ciphers sent by the client.
- The area highlighted as **7** shows the data compression method that will be used for the session.

The next packet is the response from the server issuing a certificate:



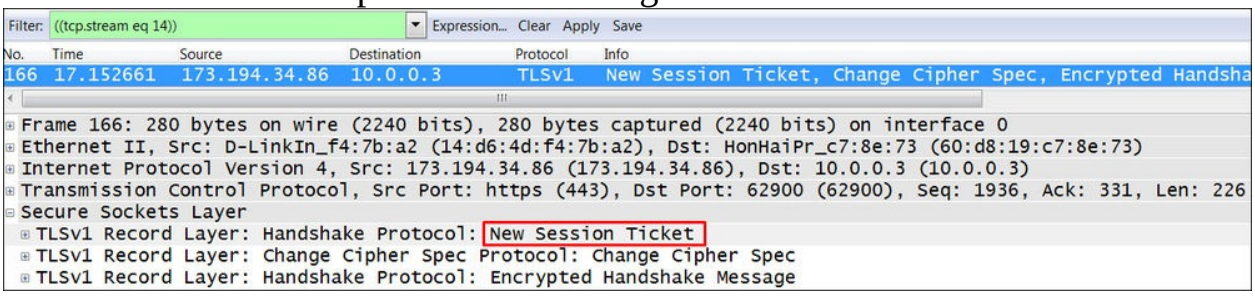
- The area highlighted as **1** shows that the server sends the **Certificate** command, which includes the server's certificate. By clicking on the (+) sign on the left of this line and digging into the details, you will see the certificate issuer, validity time, algorithm, and other data.
- The area highlighted as **2** shows that the server sends the **Server Key Exchange** command (usually Diffie-Hellman), including the required parameters (public key, signature, and so on).
- The area highlighted as **3** shows that the server sends the **Server Hello Done** command. This command indicates that the server has completed this phase of the SSL handshake. The next step is the client authentication.

The next packet (packet `165` in this example) is the response from the server, issuing a certificate.



- The area marked as **1** shows that the client sends the **Client Key Exchange** command. This command contains the premaster secret that was created by the client and was then encrypted using the server's public key. The symmetric encryption keys are generated by the client and the server, based on the data exchanged in the client and server hello messages.
- The area marked as **2** shows that the client sends the **Change Cipher Spec** notification to the server. This is done in order to indicate that the client will start using the new session keys for hashing and encryption.

The last step is when the server sends a **New Session Ticket** to the client, and it will look like the example in the following screenshot:



There's more...

I've been asked several times if it is possible to decrypt sessions that are encrypted with SSL/TLS. Well it's possible if you have the private key, which is provided to you by the server you connect to; and to get it is not an easy thing to do.

There are methods to hijack this key, and in some cases they will work. It is not an obvious thing to do, and in any case it is not in the goal of this book. If you get the private key, you simply add it in the protocol list in the preferences window and continue from there. Additional details about this feature can be obtained from http://wiki.wireshark.org/SSL, as well as from many other websites and blogs.

Chapter 11. Analyzing Enterprise Applications' Behavior

In this chapter, we will cover the following topics:

- Finding out what is running over your network
- Analyzing FTP problems
- Analyzing e-mail traffic and troubleshooting e-mail problems – POP, IMAP, and SMTP
- Analyzing MS-TS and Citrix communication problems
- Analyzing problems in the NetBIOS protocols
- Analyzing database traffic and common problems

Introduction

One of the important things that you can use Wireshark for is application analysis and troubleshooting. When the application slows down, it can be because of the LAN (quite uncommon in wired LAN), the WAN service (common due to insufficient bandwidth or high delay), or slow servers or clients (we will see this in TCP window problems). It can also be due to slow or problematic applications.

The purpose of this chapter is to get in to the details of how applications work, and provide some guidelines and recipes for isolating and solving these problems. In the first recipe, we will learn how to find out and categorize applications that work over our network. Then, we will go through various types of applications, see how they work, how networks influence their behavior, and what can go wrong.

In this chapter, we will learn how to use Wireshark in order to resolve and troubleshoot common applications that are used in an enterprise network. These are FTP, various e-mail protocols, Microsoft Terminal Server and Citrix, databases, NetBIOS protocols, and others.

Finding out what is running over your network

The first thing to do when monitoring a new network is to find out what is running over it. There are various types of applications and network protocols, and they can influence and interfere with each other when all of them are running over the network.

In some cases, you will have different VLANs, different **Virtual Routing and Forwarding** (**VRFs**), or servers that are connected to virtual ports in a **Bladeserver**. Eventually everything is running on the same infrastructure, and they can influence each other.

Tip

There is a common confusion between VRFs and VLANs. Even though their purpose is quite the same, they are configured in different places. While VLANs are configured in the LAN in order to provide network separation in the OSI layers 1 and 2, VRFs are multiple instances of routing tables to make them co-exist in the same router. This is a layer 3 operation that separates between different customer's networks. VRFs are used in **Multi Protocol Label Switching** (**MPLS**) to provide layer 3 connectivity to different customers over the same router's network, in such a way that no customer can see any other customer's network.

In this recipe, we will see how to get to the details of what is running over the network, and the applications that can slow it down.

Tip

The term **Bladeserver** refers to a server enclosure, which is a chassis of server shelves on the front and LAN switches on the back. There are several different acronyms for it; for example, IBM calls them **Bladecenter** and HP calls them **Bladesystem.**

Getting ready

When you get into a new network, the first thing to do is connect Wireshark to sniff what is running over the network. Make sure you follow these points:

- When you are required to monitor a server, port mirror it and see what is running on its connection to the network.
- When you are required to monitor a remote office, port mirror the router port that connects you to the WAN connection. Then, check what is running over it.
- When you are required to monitor a slow connection to the Internet, port mirror it to see what is going on there.

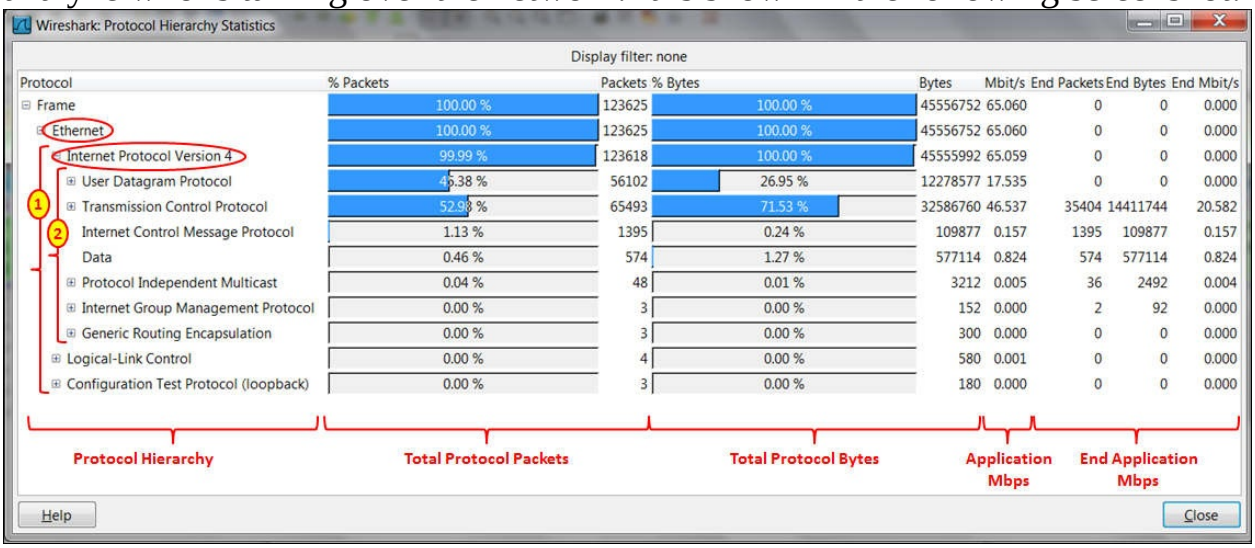
In this recipe, we will see how to use the Wireshark tools for analyzing what is running and what can cause the problems.

How to do it...

For analyzing who is talking, follow these steps:

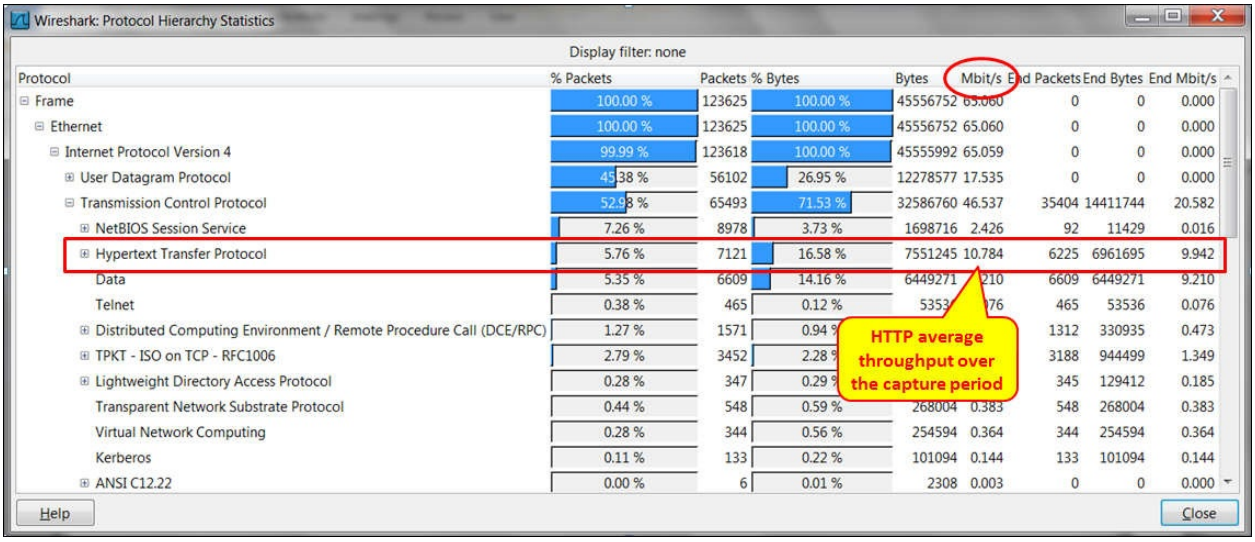
1. Connect Wireshark using one of the options mentioned in the previous section.
2. You can use the following tools:
    - Navigate to **Statistics** | **Protocol Hierarchy** for viewing the protocols that run over the network and their percentage of the total traffic
    - Navigate to **Statistics** | **Conversations** to see who is talking and what protocols are used

- In the **Protocol Hierarchy** feature, you will get a window that will help you analyze who is talking over the network. It is shown in the following screenshot:



- In the screenshot you can see the protocol distribution:
    1. **Ethernet**: IP, Logical-Link Control (LLC) and Configuration Test Protocol (loopback)
    2. **Internet Protocol Version 4**: User Datagram Protocol (UDP), Transport Control Protocol (TCP), Protocol Independent Multicast (PIM), Internet Group Management Protocol (IGMP), and Generic Routing Encapsulation Protocol (GRE)

- If you click on the + sign, all underlying protocols will be shown.
- To see a specific protocol throughput, click down to the protocols as shown in the following screenshot. You will see the application average throughput during the capture (HTTP in this example):



- Clicking on the + sign to the left of HTTP will open a list of protocols that run over HTTP (XML, MIME, JavaScripts, and more) and their average throughput during the capture period.

There's more...

In some cases (especially when you need to prepare management reports), you are required to provide a graphical picture of the network statistics. There are various sources available for this, for example:

- **Etherape (for Linux)**: http://etherape.sourceforge.net/
- **Compass (for Windows)**: http://download.cnet.com/Compass-Free/3000-2085_4-75447541.html?tag=mncol;1 (from Wildpackets)

Analyzing FTP problems

File Transfer Protocol (**FTP**) is a protocol created for transferring files over TCP/IP across a network. FTP is a protocol that runs over TCP ports `20` and `21` for the data and control connections (FTP commands) respectively.

FTP has two modes of operation:

- **Active mode** (**ACTV**): In this mode, the client initiates a control connection to the server, and the server initiates a data connection to the client
- **Passive mode** (**PASV**): In this mode, the client initiates the control and data connections to the server

Both types of connections can be implemented, and they will be explained later in this recipe in the *How it works...* section.

Getting ready

When working with FTP, if you suspect any connectivity or slow response problems, configure port mirror to one of the following:

- The FTP server port
- The client port
- A link that the traffic crosses

If required, configure a capture or display filter.

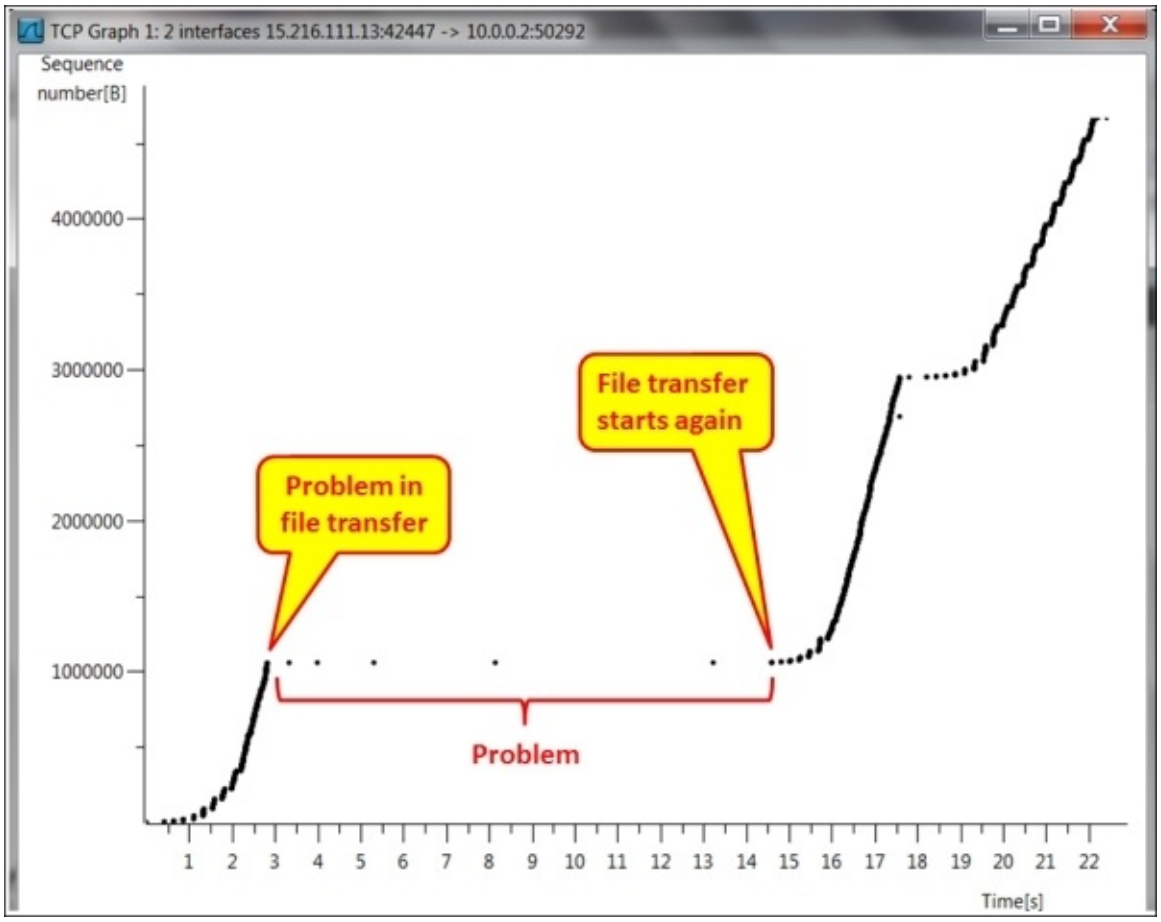
How to do it...

To check FTP performance problems, follow these steps:

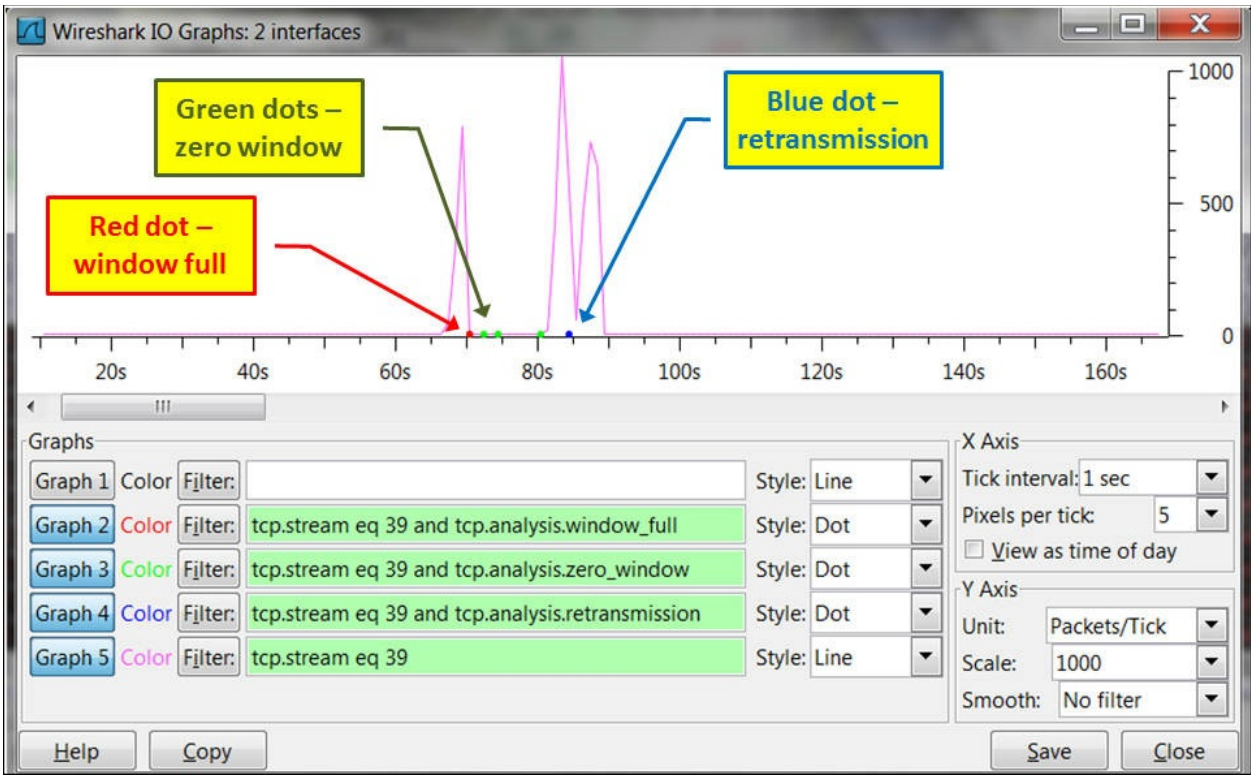
1. First, check for any Ethernet, IP, or TCP problems, as described in previous chapters. In many cases, slow responses happen due to networking problems and not necessarily due to application problems.
2. Check for TCP retransmissions and duplicate ACKs. Check if they are on the entire traffic or only on the FTP connection.

   If you get it on various connections, it is probably due to a slow network that influences the entire traffic.

   If you get it only on FTP connections to the same server or client, it can be due to a slow server or client.
3. When you are copying a single file in an FTP file transfer, you should get a straight line in the IO graph and a straight gradient in the TCP stream graph (time-sequence).
4. In the following screenshot, we can see what a bad FTP looks like in the TCP stream graph (time-sequence):

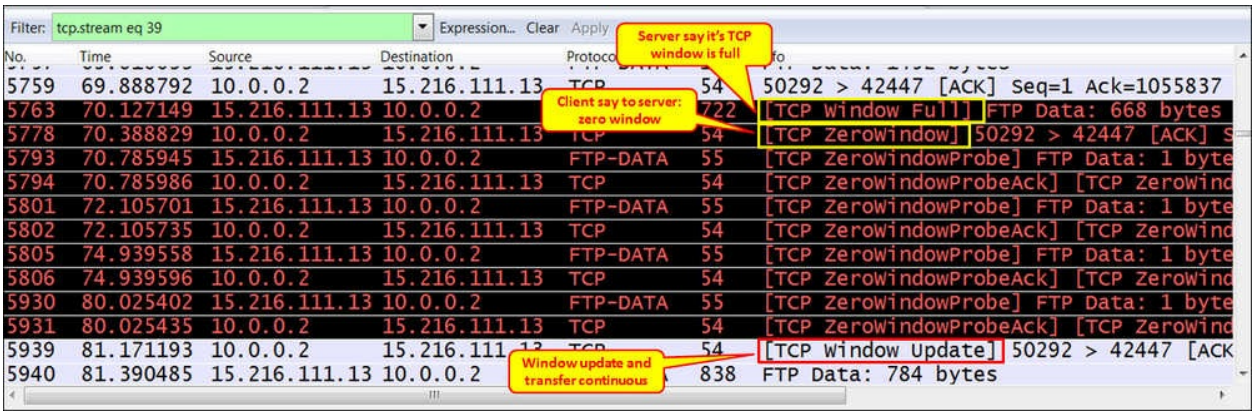


- In the following screenshot, we can see how it looks in the IO graph (configured with filters):



- In the capture file shown in the following screenshot, we can see TCP window problems. These are listed as follows:

1. The server `15.216.111.13` sends a **TCP Window Full** message to the client, indicating that the server send window is full (packet `5763`).
2. The client `10.0.0.2` sends a **TCP Zero Window** message to the server, telling the server to stop sending data (packet `5778`).
3. The server keeps sending **TCP Zero Window Probe** messages to the client, asking the client if the condition is still zero window (that tells the server not to send any more data). The client answers these messages with **TCP Zero Window Probe Ack**, indicating that this is still the case (packets `5793` to `5931`).
4. After a while, the client sends the message **TCP Window Update** to the server, telling it to start increasing the FTP throughput (packet `5939`).



- In the preceding case, it was simply a slow client. We solved the problem by working over it and deleting some unnecessary processes.

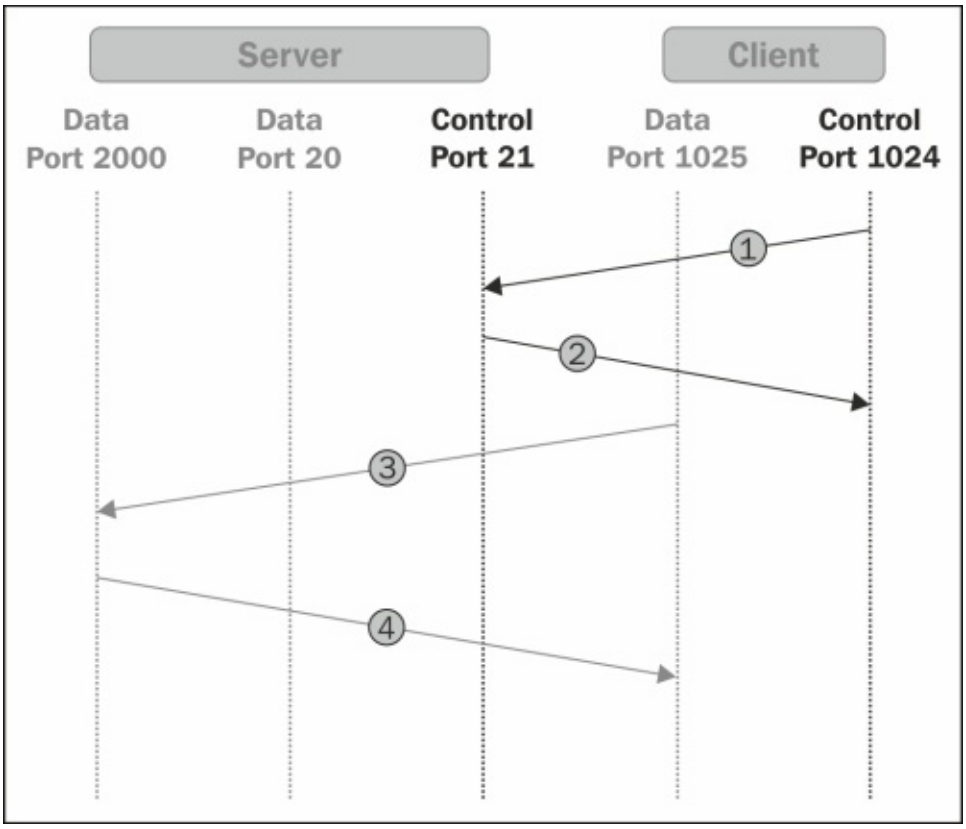
If you are facing connectivity problems, it can be due to a non-functioning server, firewall that blocks the connection on the way, or software installed on the server or client that blocks it. In this case, go through the following steps:

1. Was the TCP connection opened properly with the `SYN`/`SYN-ACK`/`ACK` packets? If not, it can be due to:
    - The firewall that blocks communications. Check with the system administrator.
    - The server that is not running. Check this on the server— in the process table, FTP server management, and so on.
    - A software of the server blocks connectivity. It can be an antivirus that has an additional firewall that blocks connections, VPN client, or any other security or protection software.
    - Check the connectivity on the client, too. It can be that it is blocked by a VPN client, a firewall on the client, and so on.
2. In the active mode, the client opens connection to the server that opens another connection. Make sure that the firewalls on the way support it, or use passive mode.

How it works...

There are two modes of FTP: active and passive. In the active mode, the server opens another connection to the client, while in passive mode, it is the client that opens the second connection to the server. Let's see how it works.

In passive mode, the operations are as shown in the following screenshot:

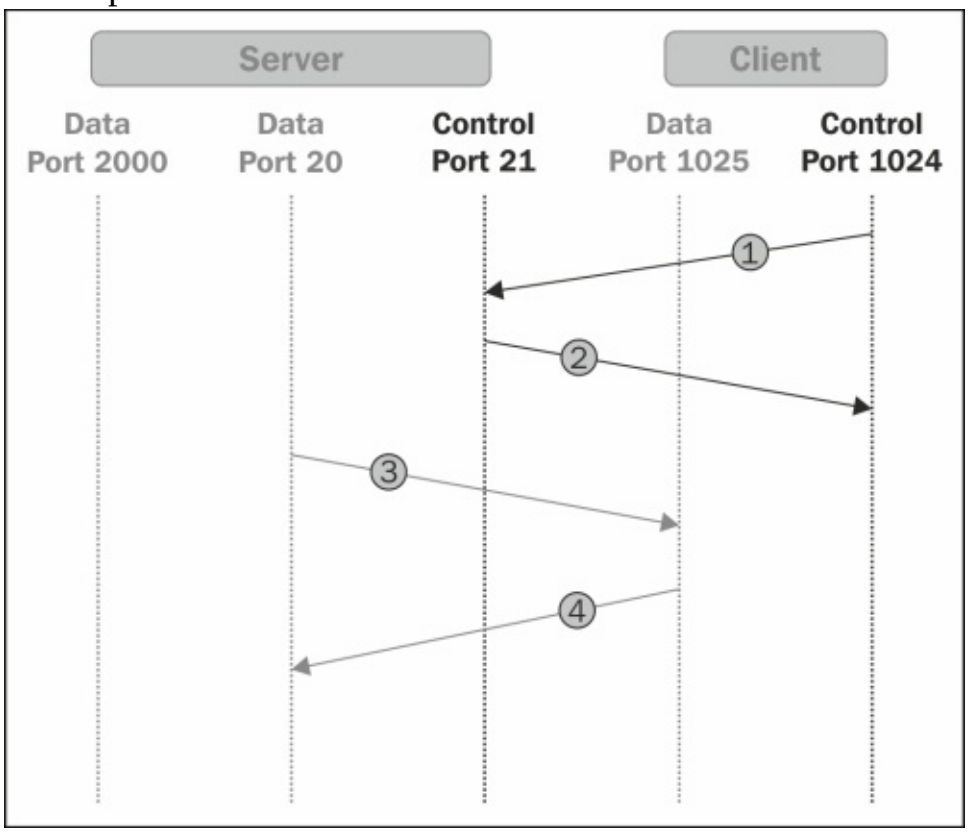


1. The client opens a control connection from a random port *P* (`1024` in the example) to the server port `21`.
2. The server answers back from port `21` to the client port `1024`.
3. Now, the client opens a data connection from the port *P+1* (`1025` in the example) to a data port that the server has opened and notified the client about (port `2000` in the example).
4. The server answers from the data port (`2000` in the example) to the client port that initiated the connection, that is, the data port *P+1* (`1025` in the

example).

In the active mode, the operation is slightly different:

1. The client opens a control connection from a random port *P* (`1024` in the example) to the server port `21`.
2. The server answers from port `21` to the client port `1024`.
3. The server opens the data connection from port `20` to the client port *P+1* (`1025` in the example).
4. The client answers from the data port *P+1* (`1025` in the example) to the server port `20`.



There's more...

FTP is a very simple application; and in most cases, FTP problems have very simple solutions. Some examples are as follows:

- **Problem 1**: I've monitored an international connection with FTP clients on one side of the network and an FTP server on the other side. The customer complained about slow performance and blamed the international service provider. After checking with the service provider, they said the connection is nearly not loaded (20 percent of a 10 Mbps line), a fact that I confirmed when I checked the line. When I looked at the TCP issues (retransmissions, window problems, and so on), there were none. Just to check, I removed the FTP server and installed another one (there are many free ones), and it started to work. It was a simple problem of an inefficient FTP server.
- **Problem 2**: A customer complained that when connecting to an FTP server, the connection was refused after every 5 or 6 trials. When I checked it with Wireshark, I saw that the FTP connection refused messages (and I already knew about this from the customer's complaint), so it looked like a dead end. Just to check, I started to stop the services running on the server, and the problem came out. It was an antivirus software that was interfering with this specific FTP server.

The bottom line is: even with Wireshark (and other software), sometimes common sense will help you more.

Analyzing e-mail traffic and troubleshooting e-mail problems – POP, IMAP, and SMTP

The common mail protocols for mail client to server and server to server communications are **Post Office Protocol version 3** (**POP3**), **Simple Mail Transfer Protocol** (**SMTP**) and **Internet Message Access Protocol version 4** (**IMAP4**).

Another common method for accessing e-mails is web access to mail, in which you have common mail servers such as Gmail, Yahoo!, and Hotmail. Some examples include **Outlook Web Access** (**OWA**) and **RPC over HTTPS** for the Outlook web client from Microsoft and others.

In this recipe, we will talk about the most common client-server and server-server protocols: POP3 and SMTP. We will also look at some typical problems by using the other methods.

Getting ready

When users are complaining about mail problems, first check if there are any obvious problems such as wrong username, bad password, and authentication protocols that are not configured. If none, connect Wireshark with port mirror to the complaining client; and if there are many of them, configure port mirror to the common server or the communications line connecting to it (when there is a remote server).

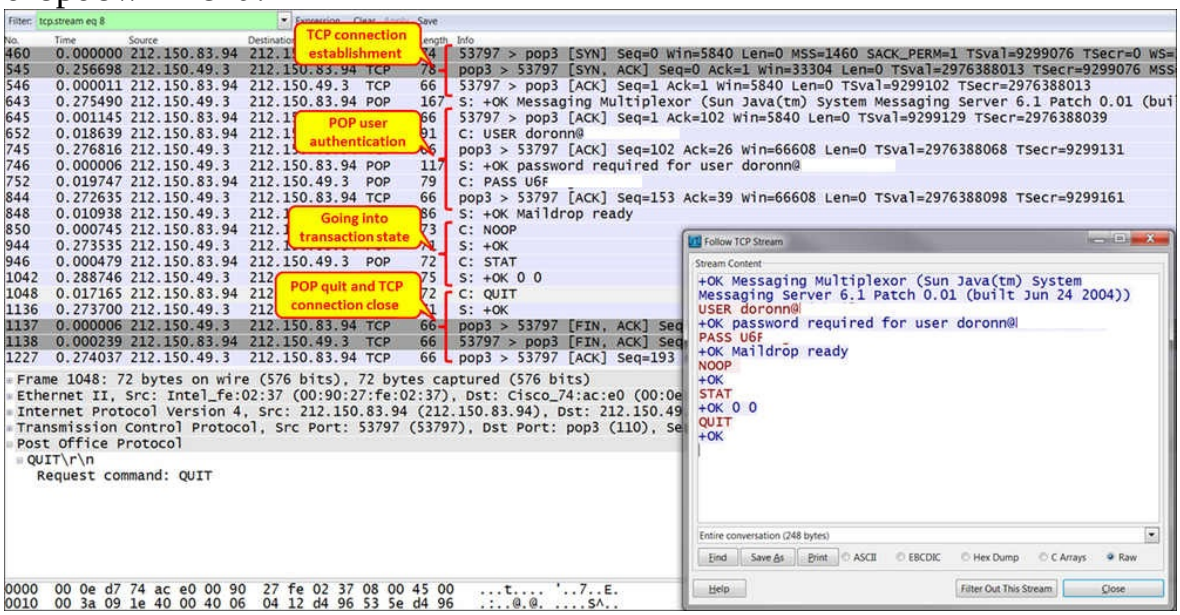
How to do it...

POP3 will usually be used for client to server communications, while SMTP will usually be used for server to server communications.

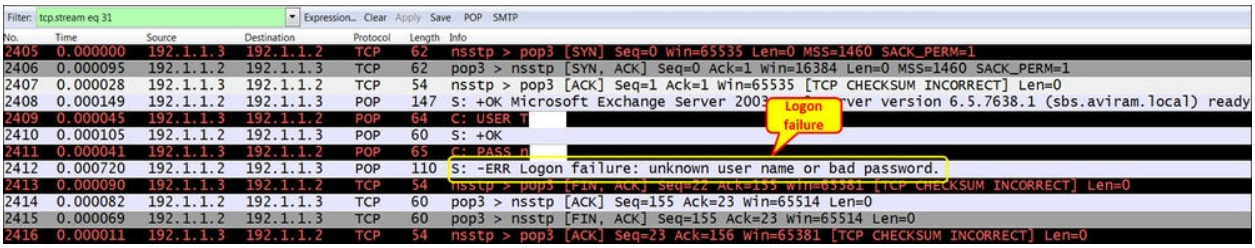
POP3 communications

POP3 is usually used for mail client to mail server communications. When a client cannot access the mail server, perform the following checks:

1. First, check if the correct username and password have been configured.
2. Then, check if the authentication has passed correctly. In the following screenshot, you can see a session opened with a username that starts with `doronn@` (all IDs were deleted) and a password that starts with `u6F`.
3. To see the TCP stream shown in the following screenshot, right-click on one of the packets in the stream and choose **Follow TCP Stream** from the dropdown menu:



- Any error messages in the authentication stage will prevent the communications from being established. You can see an example of this in the following screenshot where user authentication failed. In this case, we see that when the client gets the **Logon failure**, it closes the TCP connection.



- During the mail transfer, be aware that mail clients can easily fill a narrow-band communications line. You can check this by simply configuring the IO graphs with a filter on POP.
- Always check for common TCP indications: retransmissions, zero-window, window-full, and others. They can indicate a busy communication line, slow server, and other problems coming from the communications lines or end nodes and servers. These problems will mostly cause slow connectivity.

SMTP communications

SMTP is commonly used for the following purposes:

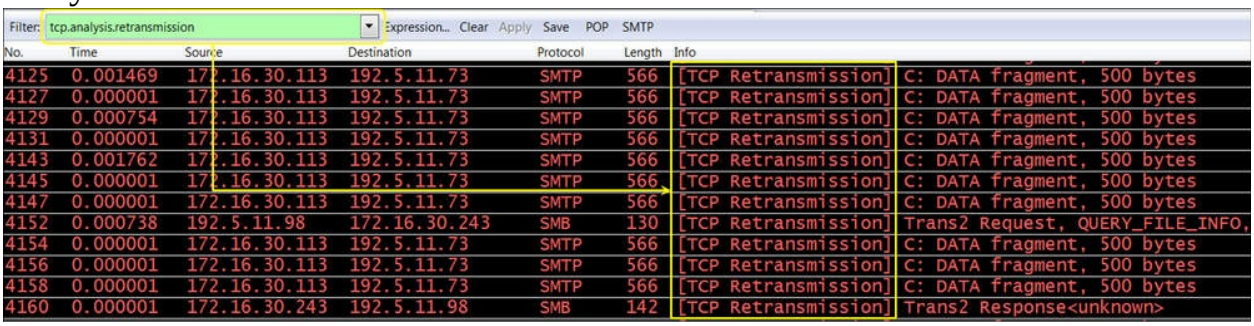
- Server to server communications, in which SMTP is the mail protocol that runs between the servers
- In some clients, POP3 or IMAP4 are configured for incoming messages (messages from the server to the client); while SMTP is configured for outgoing messages (messages from the client to the server)

When you suspect slow server-to-server communications, follow these steps to resolve the problems.

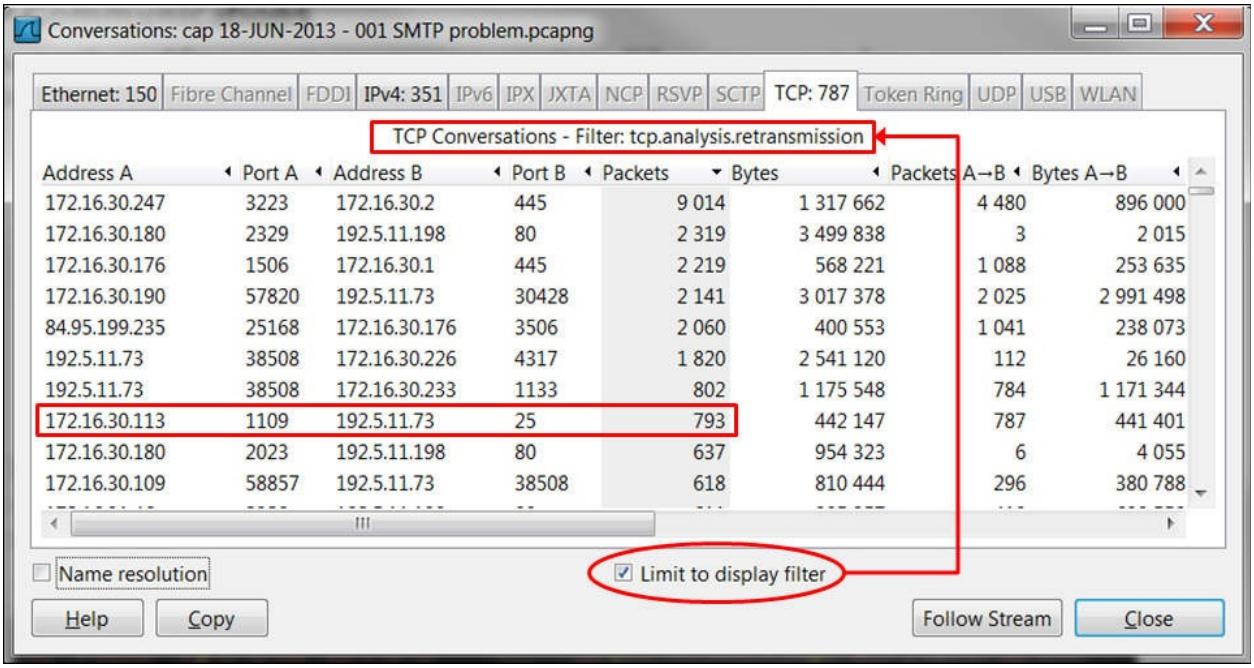
1. Check if the servers are located on the same site:
    - If they are located on the same site, you probably have slow servers or another application problem. In most of the cases, the LAN will not cause any problems—especially when both servers are in the same data centre.
    - If they are not located on the same site (when the servers are located in a remote site through WAN connections), check the load on the WAN connections. When sending large mails, they can easily block these lines—especially when they are narrow band (several Mbps).

- First, look for TCP problems; and check if you see them only on SMTP or on all other applications. For example, in the following screenshot, you can see

many TCP retransmissions:



- Check if they are because of a slow SMTP server. Is it a mail problem? When you look at the following screenshot, you see that I've used the **TCP Conversation** statistics. After checking the **Limit to display filter** checkbox and clicking on **Packets** at the top of the window (to get the list from the higher amount of packets), we can see that only 793 packets are SMTP from the retransmitted packets. There are 9014 packets retransmitted between `172.16.30.247` and `172.16.30.2` on port `445` (Microsoft DS), 2319 packets are retransmitted between `172.16.30.180` and `192.5.11.198` on port `80` (HTTP), and so on.



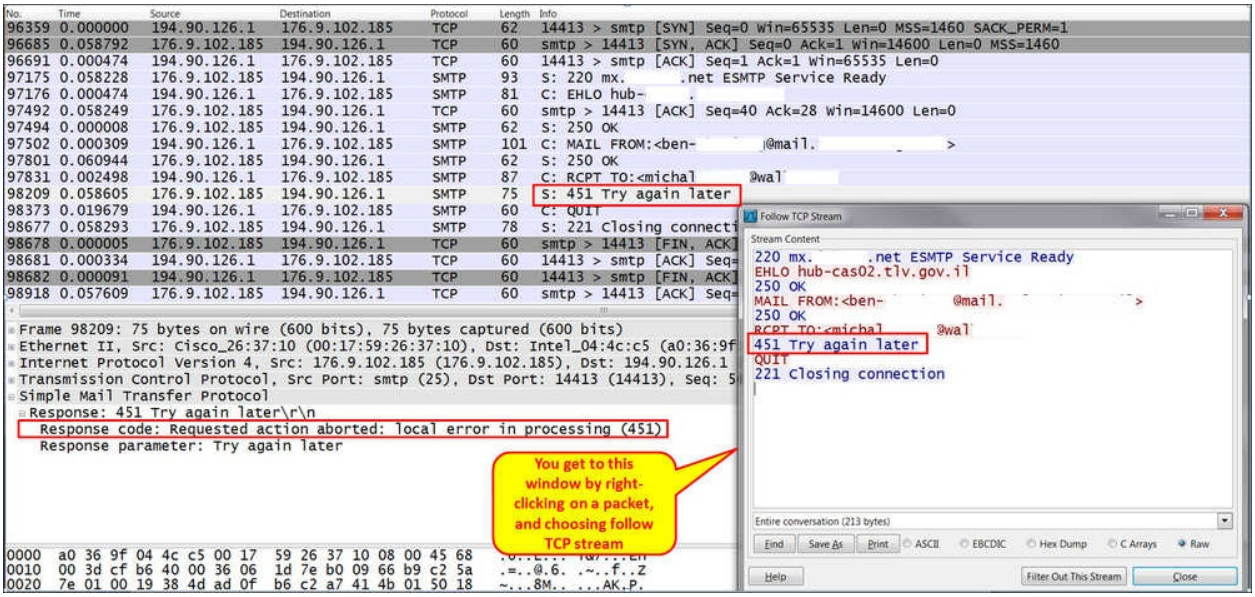
- In this case, SMTP is influenced only by bad communications. It is not an SMTP problem.
- Check for SMTP errors. In the following screenshot, you see an error `code 451`, which is also called the `local error in processing` server error. Also, a

list of errors is listed.

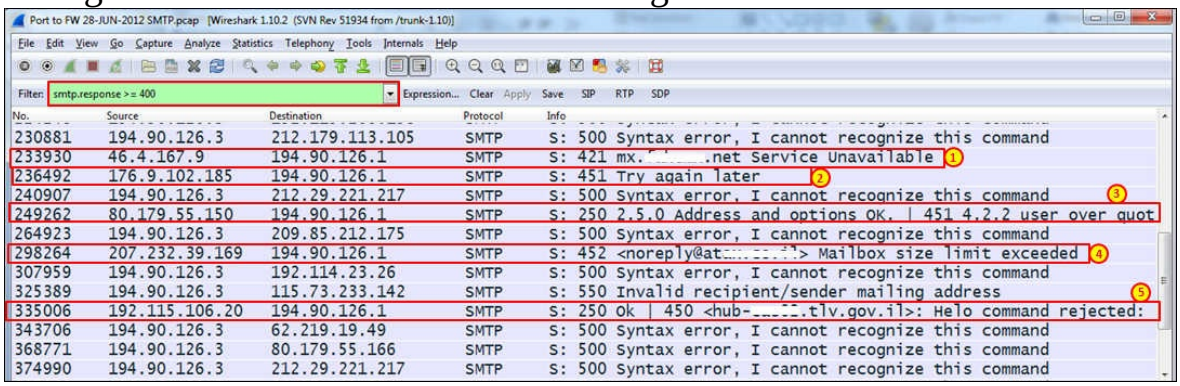
Tip

When something goes wrong, in most cases the server or the client will tell you about it. You just have to look at the messages and Google them. We will see many examples of this later.

You can also find a list of SMTP status codes in RFC 1893 (http://www.ietf.org/rfc/rfc1893.txt).



1. When you want to know which errors have been sent by the two sides, configure a filter as shown in the following screenshot:



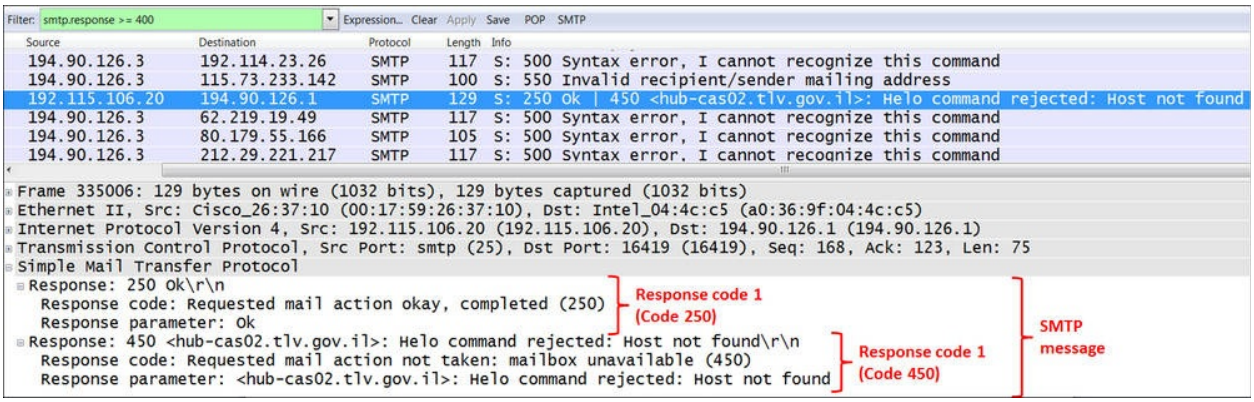
- Here you can see various events:
  - **Code 421**: This indicates that the mail service is probably unavailable (**1**).

- **Code 452**: This indicates that the server cannot respond, and tells you to try again later. This happens due to load on the server or a server problem (**2**).
- **Code 451**: (code 250 is shown in the screenshot, see the following note) This indicates the user over quota (**3**).
- **Code 452**: This indicates that the mailbox size limit has been exceeded (**4**).
- **Code 450**: (code 250 is shown in the screenshot, see the following note) This indicates that the host was not found (**5**).

Tip

In SMTP (like in many other protocols), you can get several error codes in the same message. What you see in the packet list in Wireshark can be the first one, or a partial list of it. To see the full list of errors in the SMTP message, go to the packet details and open the specific packet, as in the following screenshot.

When you see too many codes, it indicates unavailability of the server. check with the server administrator.



Some other methods and problems

Some other common methods that I mentioned earlier are web mail and RPC over HTTP:

- In web mail, we connect to the server with HTTPS; therefore this is exactly like working with HTTPS, as described in Chapter 10, *HTTP and DNS*. After logging in to the server, if any problems occur, they will be HTTPS problems.
- RPC over HTTPS will be same. Since RPC is a protocol which usually

loads the network, it is considered to be sensitive to high delays and jitter. Microsoft came up with a solution to work with their Outlook client over HTTPS and not with the standard RPC. Again, since communication runs over HTTPS, problems will be HTTPS problems.

How it works...

Mail clients will mostly use POP3 for communications with the server. In some cases, they will use SMTP as well. IMAP4 is used when server manipulation is required, for example, when you need to see messages that exist on a remote server without downloading them to the client. Server to server communications are usually implemented by SMTP.

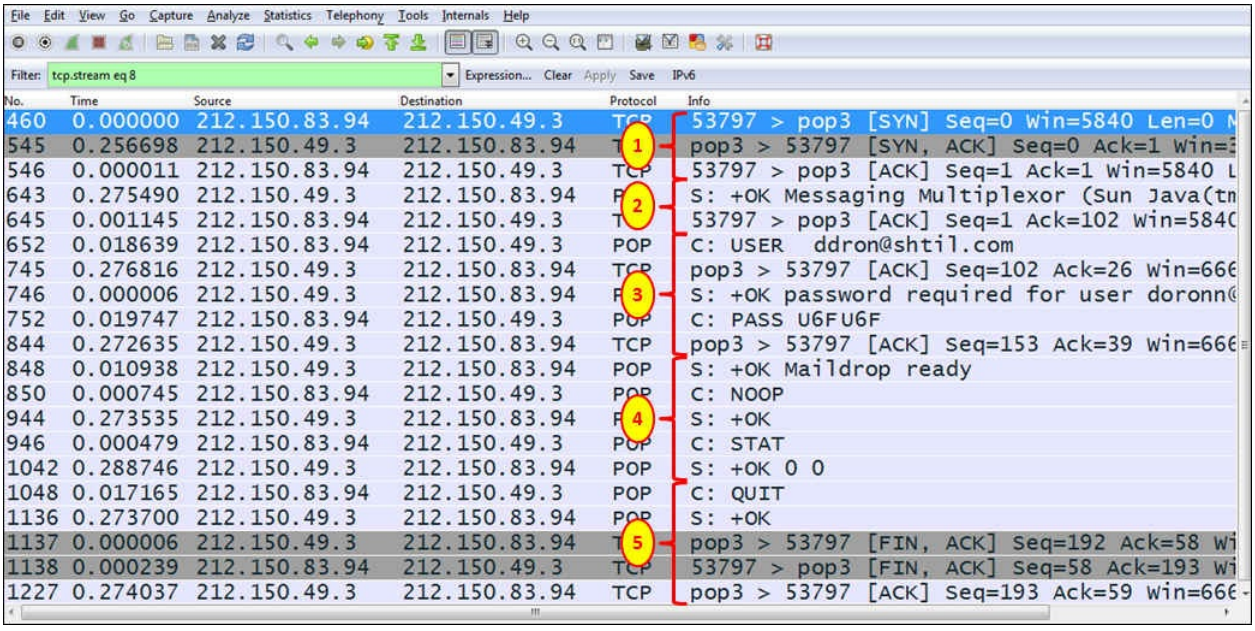
Tip

The difference between IMAP and POP is that in IMAP the mail is always stored on the server. If you delete it, it will be unavailable from any other machine. In POP, deleting a downloaded email may or may not delete that e-mail on the server.

In general, SMTP status codes are divided into three categories, which are structured in a way that helps you understand what exactly went wrong. The method and details of SMTP status codes is discussed in the following section.

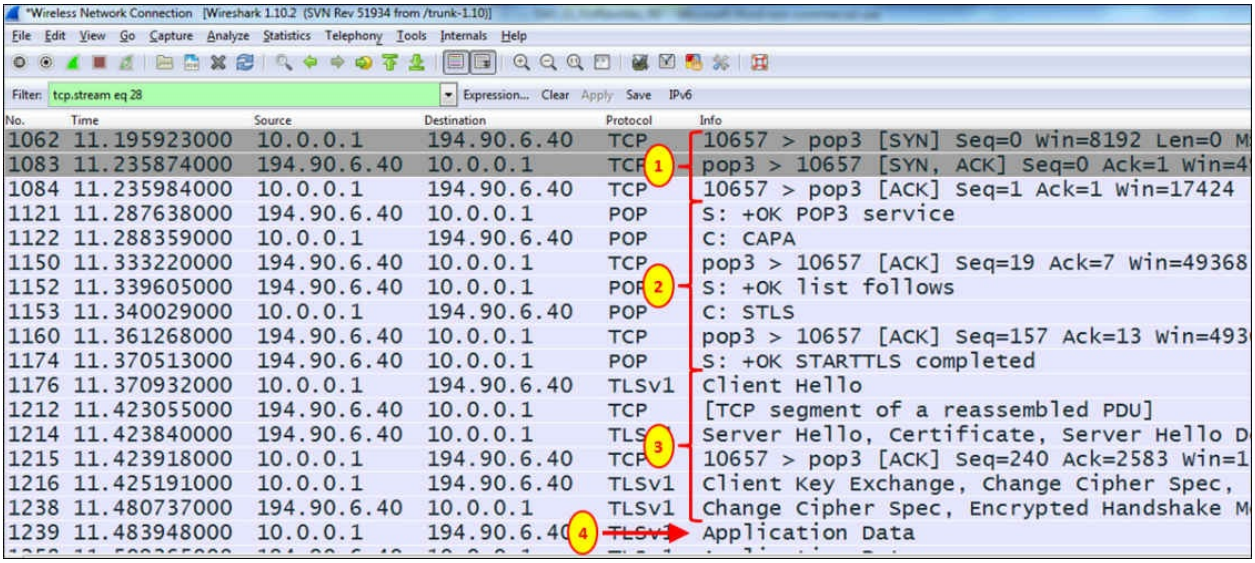
POP3

POP3 is an application layer protocol used by mail clients to retrieve e-mail messages from the server. A typical POP3 session will look like the following screenshot:



1. The client opens a TCP connection to the server.
2. The server sends an OK message to the client (**OK Messaging Multiplexor**).
3. The user sends the username and password.
4. The protocol operations begin. **NOOP** (no operation) is a message sent to keep the connection open, **STAT** (status) is sent from the client to the server to query the message status. The server answers with the number of messages and their total size (in packet `1042`, **OK 0 0** means no messages and it has total size zero).
5. When there are no mail messages on the server, the client sends a **QUIT** message (`1048`), the server confirms it (packet `1136`) and the TCP connection is closed (packets `1137`, `1138`, and `1227`).

In the case of encrypted connection, it will look nearly the same (see the following screenshot). After the connection establishment (**1**), there are several POP messages (**2**), TLS connection establishment (**3**), and then the encrypted application data.



SMTP and SMTP error codes (RFC3463)

The structure of SMTP status codes is as follows:

```
class . subject . detail
```

For example, when you see status code 450, it means the following:

- **Class 4** indicates that it is a temporary problem
- **Subject 5** indicates that it is a mail delivery status
- **Detail 0** indicates an undefined error (RFC 3463, Paragraph 3.6)

The following table lists the various classes:

Status code	Meaning	Reason
2.x.xxx	Success	Operation succeeded
4.x.xxx	Persistent transient failure	A temporary condition has prevented the server from sending the message. It can be due to server load or network bottleneck. Usually, sending the message again will succeed.
5.x.xxx	Permanent failure	A permanent problem prevented the server from sending the message. Usually server or compatibility errors.

The following table lists the various subjects:

Status code	What is it	What can be the reason
x.0.xxx	Other or undefined status	-
x.1.xxx	Addressing status	-
x.2.xxx	Mailbox status	-
x.3.xxx	Mail system status	-
x.4.xxx	Network and routing status	-
x.5.xxx	Mail delivery protocol status	-
x.6.xxx	Message content or media status	-

x.7.xxx	Security or policy status	-

The list of status details are too long to be listed here. A full list can be found in the standard pages at http://tools.ietf.org/html/rfc3463.

Some common status codes are listed in the following table:

Status code	What is it	What can be the reason
220	Service is ready	Service is running and ready to perform mail operations.
221	Service closing transmission channel	Usually OK. This is how the server closes the service when it is not required.
250	Requested mail action is OK	Message is delivered successfully.
251	Not a local user, mail will be forwarded	Everything is OK.
252	Cannot verify the user	The user couldn't be verified by the server. The mail will be delivered.
421	Service not available	The mail transfer service is not available and cannot serve incoming mail due to a transient event. This can be due to a server problem (service that is not running) or server limitation.
422	Mail size problem	The recipient mailbox has passed its quota or has a limitation on incoming mail.
431	Out of memory or disk full	Server disk is either full, or out of memory. Check the server.
432	Incoming mail queue has been stopped	It can be due to a server error (a service that stopped).

441	The receiving server is not responding	The server that sends the message indicates that the destination server does not respond.
442	Bad connection	There is a problem with the connection to the destination server.
444	Unable to route	The server was unable to determine the next hop for the message.
445	Mail system congestion	The mail server is temporarily congested.
447	Delivery time has expired	The message was considered too old by the rejecting system. This is usually due to queuing or transmission problems.
450	Requested action not taken	Message could not be transmitted. This is usually due to a problem with the mail service on the remote server.
451	Invalid command	This indicates an unsupported or out of sequence command. The action was aborted by the receiving server. This was mostly due to load on the sending or the receiving server.
452	Requested action was not taken	Insufficient storage on the receiving server.
500	Syntax error	The command sent by the server was not recognized as a valid SMTP or ESMTP command.
512	DNS error	The host server, which is the destination for the mail that was sent, could not be located.
530	Authentication problem	Authentication is required from the receiving server, or your server has been added to a black list by the receiving server.
542	Recipient address was rejected	A message indicating that your server address was rejected by the receiving server. This is usually due to Anti-spam, IDS/IPS systems, smart firewalls or other security system.

There's more...

E-mails are sometimes referred to as one of the "silent killers" of networks, especially in small enterprises that use asymmetric lines to the Internet. When sending text messages, they will not consume anything from the network; but when you send a large file of several megabytes or even tens of megabytes over a narrow-band uplink to the ISP, the rest of the users in your office will suffer from network slowdown for many seconds, even minutes. I've seen this problem in many small offices.

Another issue with mail clients is that in some cases (configurable), mail clients are configured to download all new data from the server when they start to work. If you have a customer that complains of a network slowdown at the time when all employees start their day in the office, it might be due to the tens or hundreds of clients who opened their mail clients simultaneously and the mail server is located over a WAN.

Analyzing MS-TS and Citrix communications problems

Microsoft Terminal Server (**MS-TS**) that uses **Remote Desktop Protocol** (**RDP**) and **Citrix Metaframe Independent Computing Architecture** (**ICA**) protocols are widely used for local and remote connectivity for PCs and thin clients. The important thing to remember about these types of applications is that they are transferring screen changes over the network. If there are only a few changes, they will require low bandwidth. If there are many changes, they will require high bandwidth.

Another thing is that the traffic in these applications is entirely asymmetric. Downstream traffic takes from tens of Kbps up to several Mbps, while the upstream traffic will be at most several Kbps. When working with these applications, don't forget to design your network according to this.

In this recipe, we will see some typical problems of these applications and how to locate them. For the convenience of writing, we will refer to Microsoft TS; and every time we will write MS-TS, we will refer to all applications in this category, for example, Citrix Metaframe.

Getting ready

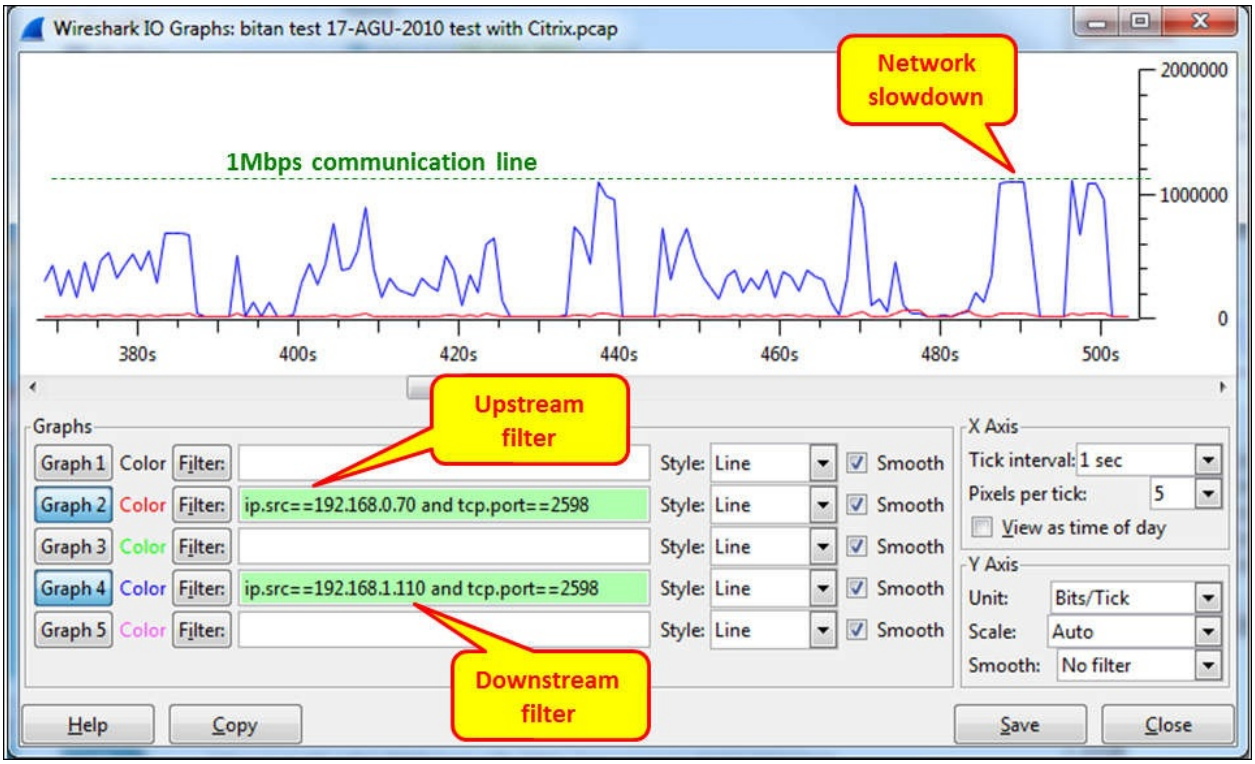
When suspecting a slow performance with MS-TS, first check with the user what the problem is. Then, connect the Wireshark to the network, with port mirror to the complaining client or to the server.

How to do it...

For locating a problem when MS-TS is involved, start with going to the users and asking questions. Follow these steps:

1. When users complain about a slow network, ask them a simple question: Do they see the slowness in the data presented on the screen, or when they switch between windows?
2. If they say that the switch between windows is very fast, it is not an MS-TS problem. MS-TS problems will cause slow window changes, picture freezes, slow scrolling of graphical documents, and so on.
3. If they say that they are trying to generate a report (when the software is running over MS-TS) but the report is generated after a long period of time, this is a database problem and not MS-TS or Citrix.
4. When a user works with MS-TS over a high-delay communication line and types very quickly, they might experience delays with the characters. This is because MS-TS is transferring window changes, and with high delays these windows changes will be transferred slowly.
5. When measuring the communication line with Wireshark:
    - Use IO graphs for monitoring the line
    - Use filters to monitor the upstream and the downstream directions
    - Configure bits per second on the y-axis

- You will get the following screenshot:



- In the preceding screenshot, you can see a typical traffic pattern with high downstream and very low upstream traffic. Notice that the **Y-Axis** is configured to **Bits/Tick**. In the time between 485 s and 500 s, you see that the throughput got to the maximum. This is when applications will slowdown and users will start to feel screen freezes, menus that move very slowly, and so on.

Tip

When a Citrix ICA client connects to a presentation server, it uses TCP ports 2598 or 1494.

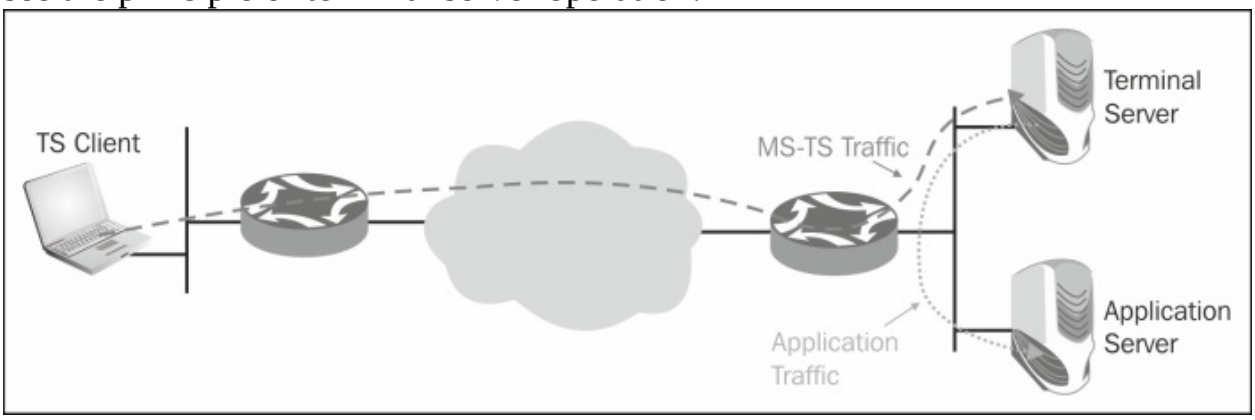
- When monitoring MS-TS servers, don't forget that the clients access the server with MS-TS and the servers access the application with another client that is installed on the server. The performance problem can come from the MS-TS or from the application.
- If the problem is an MS-TS problem, it is necessary to figure out if it is a network problem or a system problem:
  - Check the network with Wireshark to see if there are any loads. Loads such as the one shown in the previous screenshot can be solved by simply

increasing the communication lines.

- Check the server's performance. Applications like MS-TS are mostly memory consuming, so check mostly for memory (RAM) issues.

How it works...

MS-TS, Citrix Metaframe, and applications simply transfer window changes over the network. From your client (PC with software client or thin client), you connect to the terminal server; and the terminal server runs various clients that are used to connect from it to other servers. In the following screenshot, you can see the principle of terminal server operation:



There's more…

From the terminal server vendors, you will hear that their applications improve two things. They will say that it improves manageability of clients because you don't have to manage PCs and software for every user—you simply install everything on the server, and if something fails you fix it on the server. They will also say that traffic over the network will be reduced.

Well, I will not get into the first argument. This is not our subject, but I strongly reject the second one. When working with a terminal client, your traffic entirely depends on what you are doing:

1. When working with text/characters-based applications, for example, some **Enterprise Resource Planning** (**ERP**) screens, you type in and read data. When working with the terminal client, you will connect to the terminal server that will connect to the database server. Depending on the database application you are working with, the terminal server can improve performance significantly or does not improve it at all. We will discuss this in the database section. Here, you can expect a load of tens to hundreds Kbps.
2. In cases where you are working with regular office documents such as Word, PowerPoint, and so on, it entirely depends on what you are doing. Working with a simple Word document will require tens to hundreds of Kbps. Working with PowerPoint will require hundreds Kbps to several Mbps, and when you present the PowerPoint file with full screen (function **F5**) the throughput can jump up to 8 to 10 Mbps.
3. Browsing the Internet will take between hundreds of Kbps to several Mbps, depending on what you are doing. High resolution movies over terminal server to the Internet—well, just don't do it.

Before implementing any terminal environment, test it. I once had a software house that wanted their logo (at the top-right corner of the user window) to be very clear and striking. They refreshed it 10 times a second, which caused the 2 Mbps communication line to be blocked. You never know what you don't test.

Analyzing problems in the NetBIOS protocols

Network Basic Input/Output System (**NetBIOS**) is a set of protocols developed in the early 1980s for LAN communications. A few years later, it was adopted by Microsoft for their networking over the LAN, and then it was migrated for working over TCP/IP (NetBIOS over TCP/IP, RFCs 1001, and 1002).

In today's networks, NetBIOS provides three services:

- **Name service** (port 137) for name registration and name to IP address resolution.
- **Datagram distribution service** (port 138) for service announcements by clients and servers.
- **Session service** (port 139) for session negotiation between hosts. This is used for accessing files, open directories and so on.

In this chapter, we will get into some common problems with the NetBIOS suite of protocols, and we will learn how to try and solve them. Since the NetBIOS set of protocols is quite complicated, and there are hundreds of scenarios of things that might go wrong, we will try to provide some guidelines for how to look for common problems and what might go wrong.

Getting ready

NetBIOS protocols work in the Windows environments, along with MAC and Linux machines communicating with Windows. When facing problems such as instability, slow response times, disconnections, and so on in these environments NetBIOS issues can be one of the reasons for it. When facing these problems, the tool for solving them is Wireshark. It will show you what runs over the network, and Windows tools will show you what runs in the clients and servers.

How to do it...

To try and find out what a problem could be, connect your laptop with the Wireshark to the network, and port mirror the suspected clients or server as described below. In the following sections, we will see several scenarios for several problems.

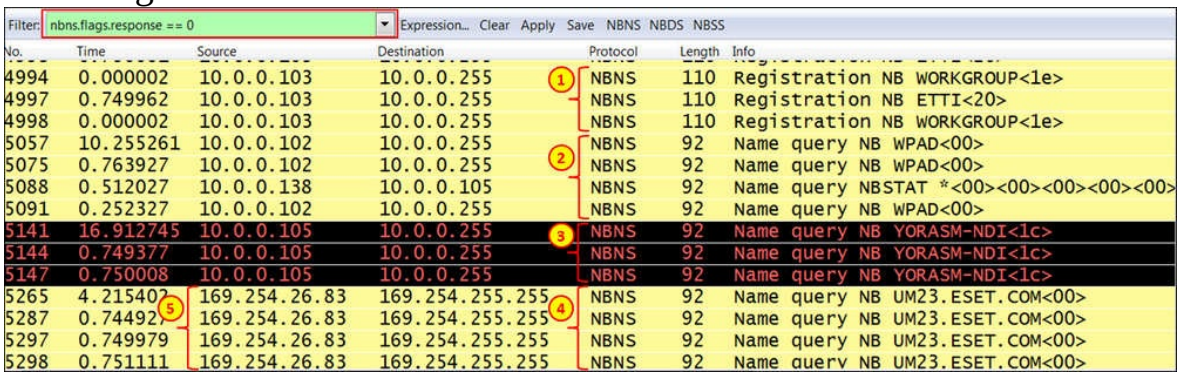
There are many predefined filters that are used with NetBIOS. You can find them by clicking on the **Expression** button, which is on the right-hand side of the **Display Filters** window.

1. For general NetBIOS commands, they start with `netbios`.
2. For NetBIOS name service, they start with `nbns`.
3. For NetBIOS datagram service, they start with `nbds`.
4. For NetBIOS session service, they start with `nbss`.
5. For SMB, they start with `smb`.

General tests

First, take a general look at the network. Then, look for suspicious patterns:

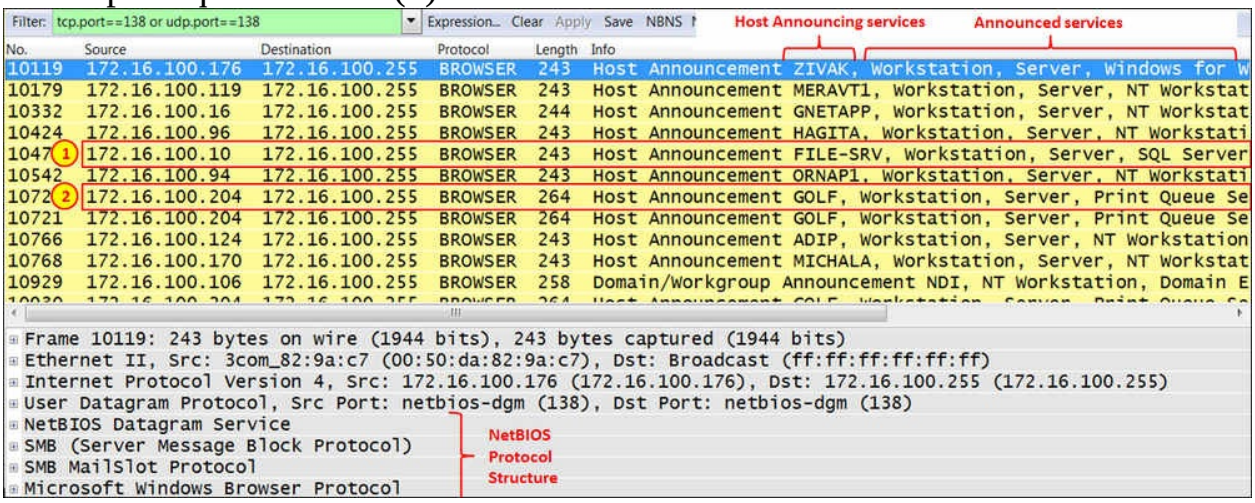
1. Connect Wireshark to the network. Each one of the ports will do fine, as long as you are on the same broadcast domain with the clients that are having the problems.
2. Configure the display filter `nbns.flags.response == 0`. It will give you the NBNS requests. You will see many broadcasts, as shown in the following screenshot:



- As you saw in the previous screenshot, in the capture file you will see the following:

- **NBNS registration packets (1)**: In the examples, there are registrations with the names `WORKGROUP` and `ETTI`. NBNS server will accept or reject the name registration by issuing a positive or negative Name Registration Response to the requesting node. If none are received, the requesting node will assume it is OK.
- **NBNS Queries (2, 3 and 4)**: Queries are sent for the name specified. If there is an NBNS server (this is the domain controller), you will see one of the following responses:
- requested name does not exist (code 3)
- no error (code 0)

- Make sure there is no registration or any other requests coming from addresses that start with `169.254` (**5**). These are **Automatic Private IP Addressing** (**APIPA**) addresses. This actually means that the PC is configured to accept addresses automatically (by DHCP) and it has not received one.
- There are many announcement packets as well. These will be broadcast on UDP port `138`. Here, you will see that every station announces its capabilities: workstation, server, print server, and so on. For example, you can see here that:
  - `172.16.100.10` name is `FILE-SRV`, and it functions like workstation, server, and SQL server (**1**)
  - `172.16.100.204` name is `GOLF`, and it functions like workstation, server, and a print queue server (**2**)



- There are some worms and viruses that are using the NetBIOS name service to scan the network. Look for unusual patterns like massive scanning, high broadcast rate, and so on.
- Verify that you don't have too many broadcasts. Five to 10

broadcast/minute/device are reasonable. More than this usually means problems.

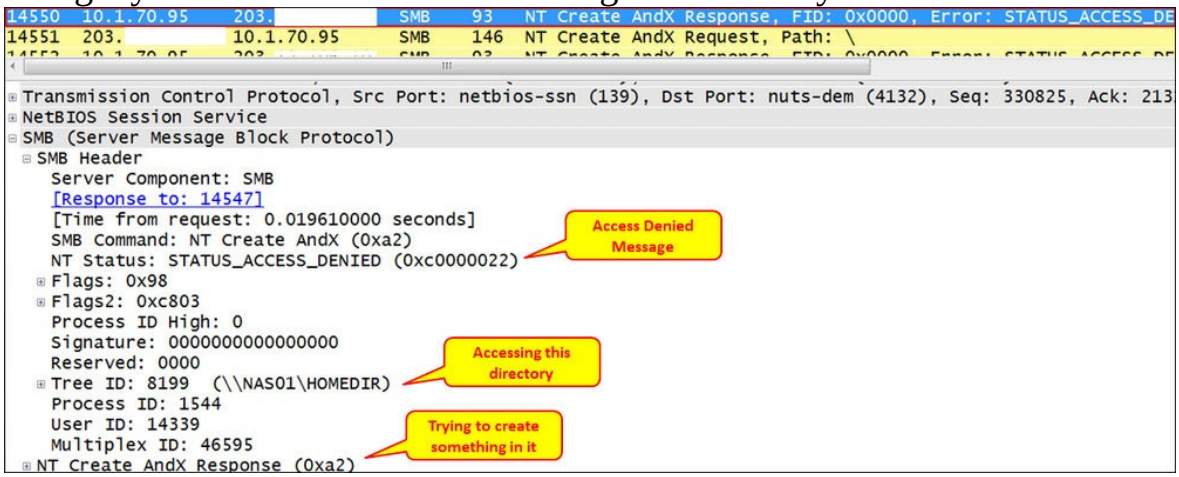
Tip

There are hundreds of message scenarios you can see here. Use the Wireshark Expert system, Google, and common sense to discover the problem.

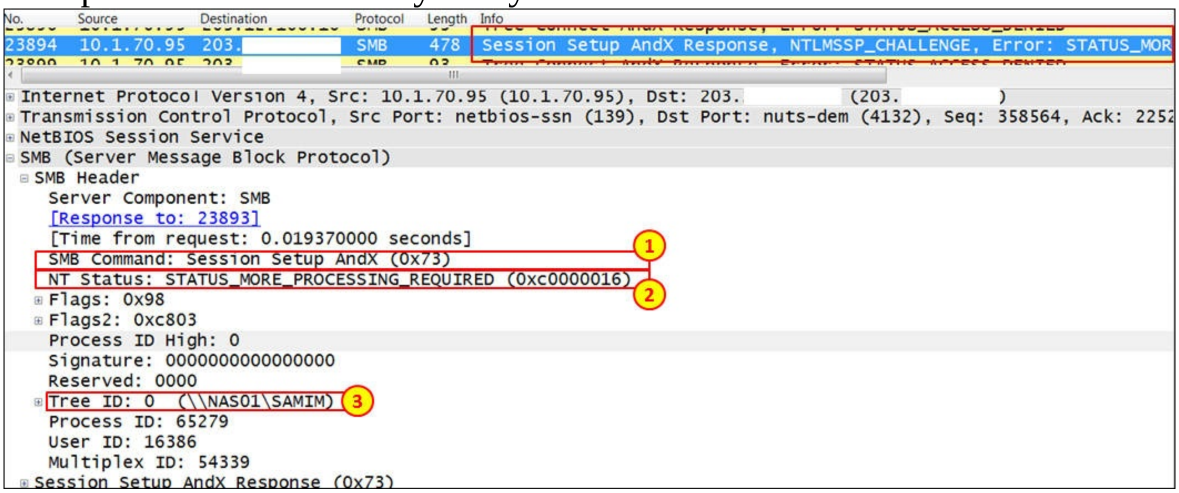
Specific issues

Here are some issues and problems you might see during usual operation:

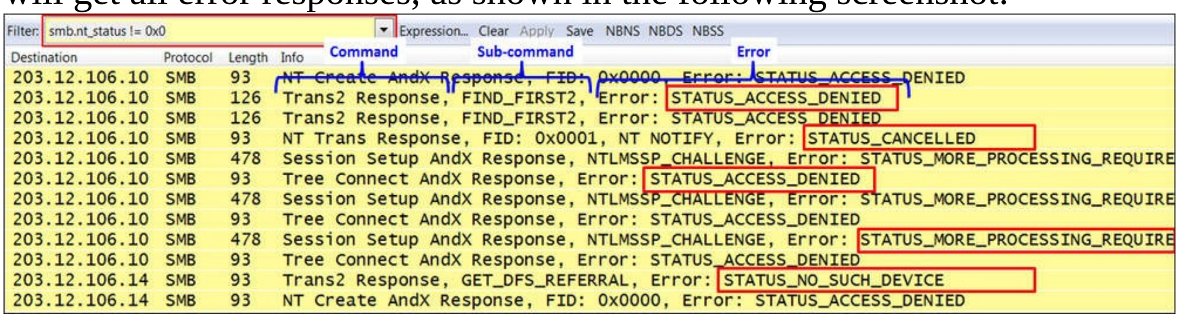
1. Using **Server Message Block** (**SMB**), which is the protocol that is used for browsing directories, copying files, and other operations over the network, you might see some error codes. The full list of error codes is listed in Microsoft MSDN: http://msdn.microsoft.com/en-us/library/ee441884.aspx.
2. Code `0` means `STATUS_OK`, when everything works fine and there is no problem. Any other code should be examined.
3. In the following example, you can see a message `STATUS_ACCESS_DENIED`. This is one of many error codes you should look for. In the example, access to `\\NAS01\HOMEDIR` on a server with an IP address that starts with `203` (full address hidden due to security reasons) was denied.
4. When you try to see the home directory by browsing it, Windows will usually show you an **ACCESS DENIED** message or something similar. The problem can happen when an application is trying to access a directory, and cannot get access to it. In this case, you can see an **ACCESS DENIED** message, a software message of communication problem, or any other message the programmers have made for you. Using Wireshark in this case will get you to the exact error and Google will show you the reason for it.



5. In the next example, we see a status `STATUS_MORE_PROCESSING_REQUIRED` (**2**) that happened during session setup (**1**) on `\\NAS01\SAMIM` (**3**).
   - Looking at the link mentioned earlier, we see that this is because on the designated named pipe, there is more data available to read.
   - A short Google lookup tells us that it might indicate a credentials problem. Check with your system administrator.



6. To see all SMB error messages, type the filter `smb.nt_status != 0x0`. You will get all error responses, as shown in the following screenshot:



How it works...

As we saw in the introduction to this section, NetBIOS provides three services: **Net BIOS Name Service** (**NBNS**), **NetBIOS Datagram Distribution Service** (**NBDS**), and **NetBIOS Session Service** (**NBSS**).

NBNS is the service that registers and translates names to IP addresses. Registration happens when a client registers its name in the domain controller. The client sends a registration request, and then gets a response whether the registration is OK or the name is registered with another device. Microsoft environment was implemented with WINS when most networks did not use it, and later it was replaced by DNS. It works over UDP port `137`.

NBDS is used for service announcements by clients and servers. With this service, devices on the network announce their names, services that they can provide to other devices on the networks, and how to connect to these services. It works over UDP port `138`.

NBSS is used to establish sessions between hosts, open or save files, and execute remote files and other sessions over the network. It works over TCP port `139`.

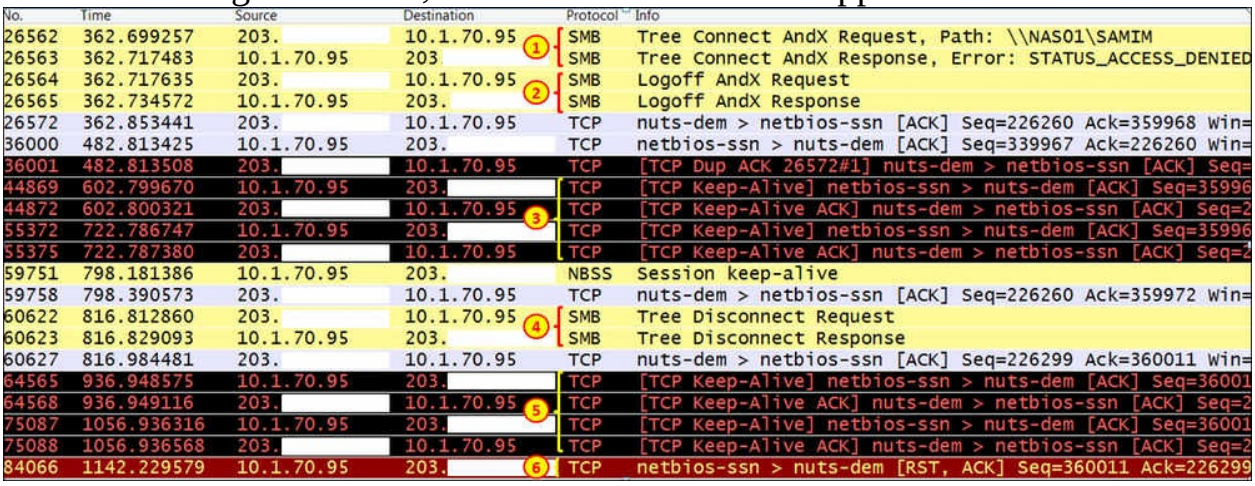
There are additional protocols such as **Server Message Block** (**SMB**) that run over NBSS for transaction operations and over NBDS for service announcement, **SPOOLS** for printer requests, and several others. To get to the details of NetBIOS is beyond the scope of this book. In the case that you are required to troubleshoot NetBIOS protocols, follow the instructions in this section—pay special attention to error messages and notes.

There's more…

In this section, I would like to show some examples to get a better understanding of the NetBIOS protocols.

Example 1 – application freezing

In the following screenshot, we see the reason for an application freeze:



No.	Time	Source	Destination	Protocol	Info
26562	362.699257	203.	10.1.70.95	SMB	Tree Connect AndX Request, Path: \\NAS01\SAMIM
26563	362.717483	10.1.70.95	203	SMB	Tree Connect AndX Response, Error: STATUS_ACCESS_DENIED
26564	362.717635	203.	10.1.70.95	SMB	Logoff AndX Request
26565	362.734572	10.1.70.95	203.	SMB	Logoff AndX Response
26572	362.853441	203.	10.1.70.95	TCP	nuts-dem > netbios-ssn [ACK] Seq=226260 Ack=359968 Win=
36000	482.813425	10.1.70.95	203.	TCP	netbios-ssn > nuts-dem [ACK] Seq=339967 Ack=226260 Win=
36001	482.813508	203.	10.1.70.95	TCP	[TCP Dup ACK 26572#1] nuts-dem > netbios-ssn [ACK] Seq=
44869	602.799670	10.1.70.95	203.	TCP	[TCP Keep-Alive] netbios-ssn > nuts-dem [ACK] Seq=35996
44872	602.800321	203.	10.1.70.95	TCP	[TCP Keep-Alive ACK] nuts-dem > netbios-ssn [ACK] Seq=2
55372	722.786747	10.1.70.95	203.	TCP	[TCP Keep-Alive] netbios-ssn > nuts-dem [ACK] Seq=35996
55375	722.787380	203.	10.1.70.95	TCP	[TCP Keep-Alive ACK] nuts-dem > netbios-ssn [ACK] Seq=2
59751	798.181386	10.1.70.95	203.	NBSS	Session keep-alive
59758	798.390573	203.	10.1.70.95	TCP	nuts-dem > netbios-ssn [ACK] Seq=226260 Ack=359972 Win=
60622	816.812860	203.	10.1.70.95	SMB	Tree Disconnect Request
60623	816.829093	10.1.70.95	203.	SMB	Tree Disconnect Response
60627	816.984481	203.	10.1.70.95	TCP	nuts-dem > netbios-ssn [ACK] Seq=226299 Ack=360011 Win=
64565	936.948575	10.1.70.95	203.	TCP	[TCP Keep-Alive] netbios-ssn > nuts-dem [ACK] Seq=36001
64568	936.949116	203.	10.1.70.95	TCP	[TCP Keep-Alive ACK] nuts-dem > netbios-ssn [ACK] Seq=2
75087	1056.936316	10.1.70.95	203.	TCP	[TCP Keep-Alive] netbios-ssn > nuts-dem [ACK] Seq=36001
75088	1056.936568	203.	10.1.70.95	TCP	[TCP Keep-Alive ACK] nuts-dem > netbios-ssn [ACK] Seq=2
84066	1142.229579	10.1.70.95	203.	TCP	netbios-ssn > nuts-dem [RST, ACK] Seq=360011 Ack=226299

In the example, we make the following observations:

1. A client with IP address that starts with `203` is trying to connect to `\\NAS01\SAMIM` on a server with an IP address `10.1.70.95`, and gets back a `STATUS_ACCESS_DENIED` error.
2. The client logs off and the server confirms it.
3. Since the applications waits, TCP is holding the connection with keep-alive messages.
4. After a while, the client sends disconnect requested that is approved by the server.
5. The application waits and TCP maintains the connection with keep-alives.
6. TCP closes the connection with `RST` (Reset).

What the customer saw here was an application freeze.

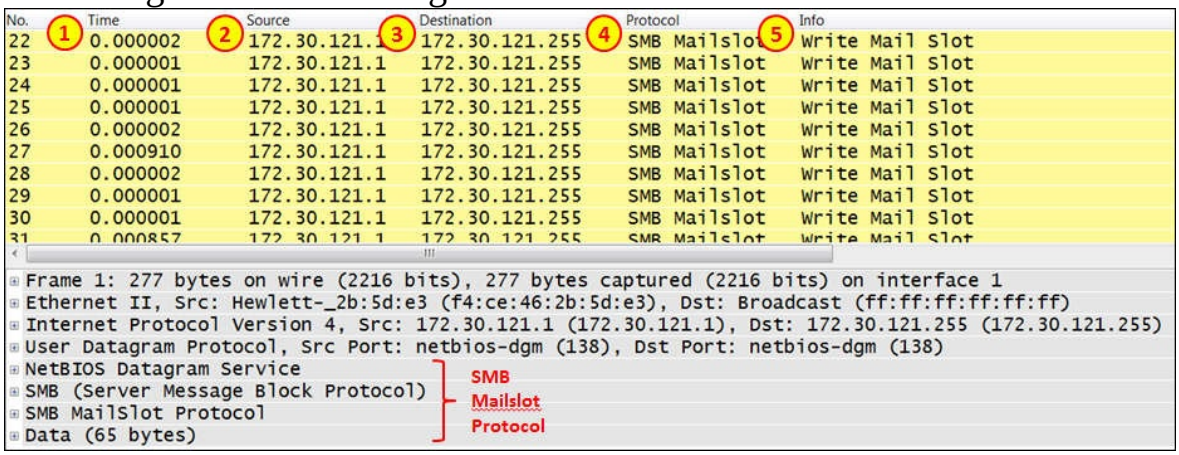
Example 2 – broadcast storm caused by SMB

In one of my client's networks, I got an urgent call that a remote office was disconnected from the HQ. Some network details are as follows:

- The remote office addresses are on subnet `172.30.121.0/24`, with a default gateway `172.30.121.254`.
- The HQ addresses are on subnet `172.30.0.0/24`. The connections between the remote offices and the centre are with L3 IP-VPNs over MPLS network.

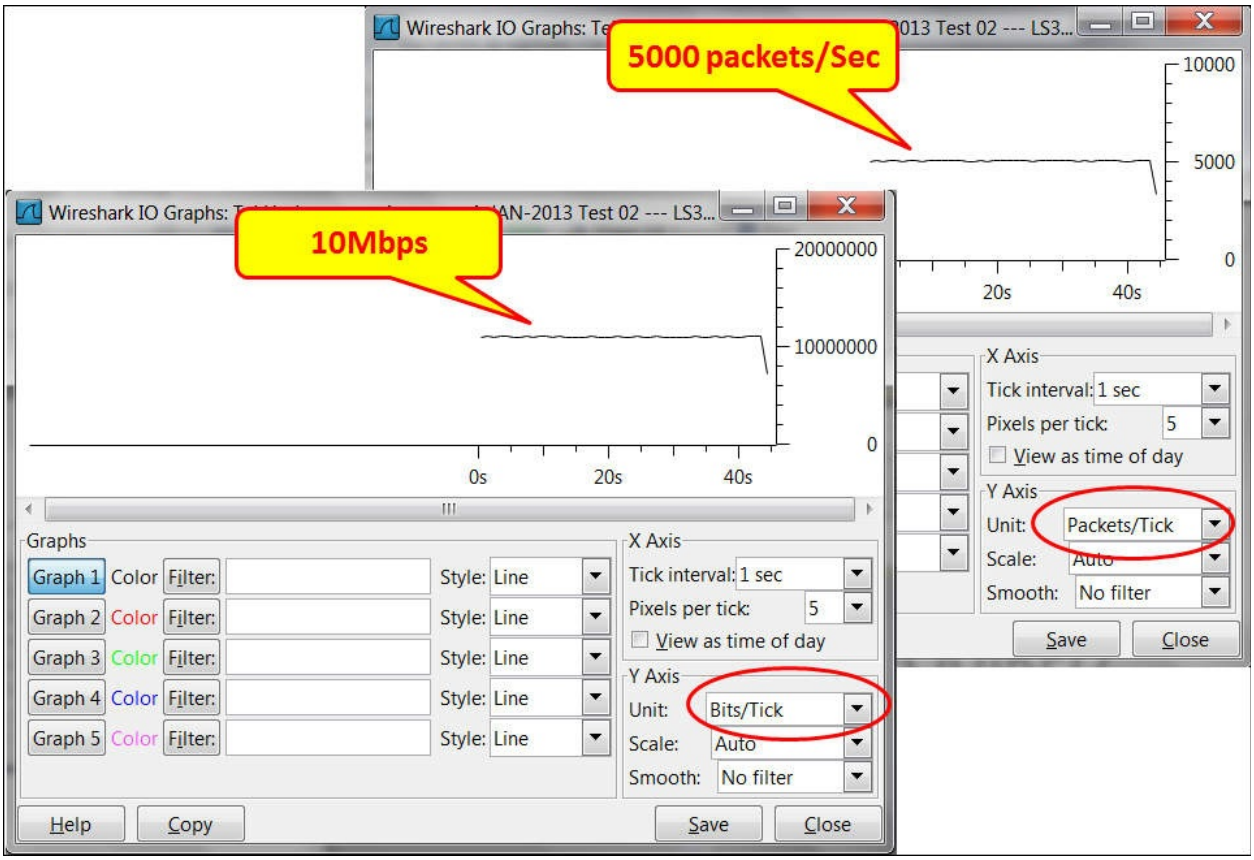
To solve the problem, I did the following:

1. I tried to ping the servers in the HQ. I got no response.
2. I called the service provider that provides the lines to the centre, and they said that on their monitoring system they don't see any load on the line.
3. I pinged the local router, `172.30.121.254`, and got no response. The meaning is that PCs on the LAN couldn't get to their local router, which is the default gateway.
4. I connected a Wireshark with port mirror to the router port, and I saw something like the following screenshot:



- I saw that a huge amount of packets are generated within microseconds (1) by a host with IP address `172.30.121.1`. The packets are broadcast (3), and the service that generated them is Write Mail Slot (5), which is sent by the SMB Mailslot protocol (4).
- To get the picture of the number of packets, I used the IO Graphs feature. I got 5000 packets per second, that generated 10 Mbps that block the poor old router port (changing the router port to 100 Mbps or 1 Gbps wouldn't help. It would have been blocked too).

- When I didn't find anything about it on Google or Microsoft, I started to stop services that I don't know, keeping track of what happened with the broadcast. Eventually, the service that caused the problem was called `LS3Bcast.exe`. I stopped it, made sure it didn't come back and that was it.



Analyzing database traffic and common problems

Some of you may wonder why I have added this section here. After all, databases are considered to be a completely different branch in the IT environment. There are databases and applications on one side and the network and infrastructure on the other side. It is correct since we are not supposed to debug databases; there are DBAs for this. But through the information that runs over the network, we can see some issues that can help the DBAs with solving the problem.

In most cases, the IT staff will come to us first because people blame the network for everything. We will have to make sure that the problems are not coming from the network and that's it. In a minority of the cases, we will see some details on the capture file that can help the DBAs with what they are doing.

Getting ready

When the IT team come to us complaining about the "slow network", there are some things to do just to verify that it is not the case. Follow the instructions in the following section to make sure we avoid the "slow network" issue.

How to do it...

In the case of database problems, follow these steps:

1. When you get complaints about the "slow network responses" start asking these questions:
    - Is the problem local or global? Does it occur only in the remote offices, or also in the center? When the problem occurs in the entire network, it is not a WAN bandwidth issue.
    - Does it happen the same for all clients? If not, there might be a specific problem that happens only with some users because only these users are running a specific application that causes the problem.
    - Is the communication line between the clients and the server loaded? What is the application that loads them?
    - Do all applications work slowly, or is it only the application that works with the specific database? Maybe some PCs are old and tired, or is it a server that runs out of resources?

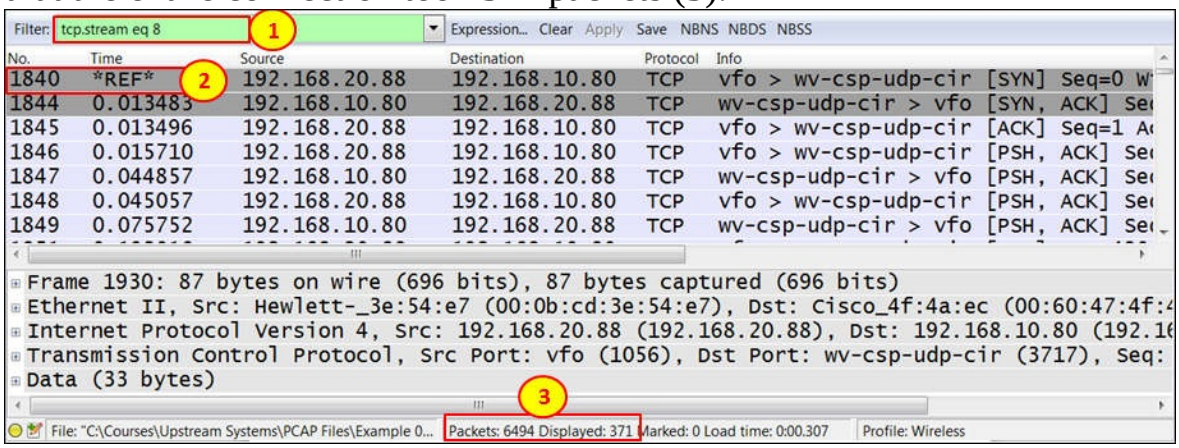
- When we are done with the questionnaire, let's start our work:
    1. Open Wireshark and start capturing packets. You can configure port mirror to a specific PC, to the server, to a VLAN, or to a router that connects to a remote office in which you have the clients.
    2. Look at TCP events (expert info). Do they happen on the entire communication link, on specific IP address/addresses, or on specific TCP port number/numbers? This will help you isolate the problem and verify whether it is on a specific link, server, or application.

        Tip

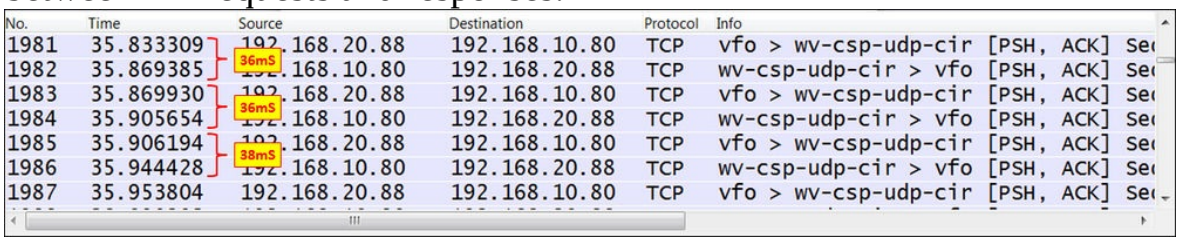
        When measuring traffic on a connection to the Internet, you will get many retransmissions and duplicate ACKs to websites, mail servers, and so on. This is the Internet. In an organization, you should expect 0.1 to 0.5 percent retransmissions. When connecting to the Internet, you can expect much higher numbers.

- If you see problems in the network, solve them as we learned in previous chapters. But, there are some network issues that can influence database behavior. In the following example, we see the behavior of a client that works with the server over a communication line with a roundtrip delay of 35 to 40 ms.

1. We are looking at the TCP stream number 8 (**1**), and the connection started with TCP SYN/SYN-ACK/ACK. I've set this as a reference (**2**). We can see that the entire connection took 371 packets (**3**).

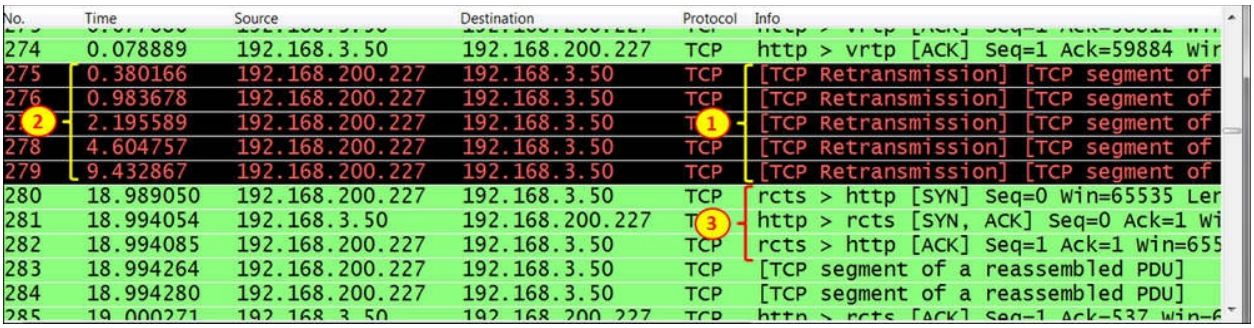


2. The connection continues, and we see time intervals of around 35 ms between DB requests and responses.



3. Since we have 371 packets travelling back and forth, 371*35 ms gives us around 13 seconds. Add to this some retransmissions that might happen and some inefficiencies, and this leads to a user waiting for 10 to 15 seconds and more for a database query.
4. In this case, you should consult with the DBA on how to significantly reduce the number of packets that run over the network; or you can move to another way of access, for example, terminal server or web access.

- Another problem that can happen is that you will have a software issue that will reflect in the capture file. If you have a look at the following screenshot, you will see that there are five retransmissions (**1**), and then a new connection is opened from the client side (**3**). It looks like a TCP problem but it occurs only in a specific window in the software. It is simply a software procedure that stopped processing, and this stopped the TCP from responding to the client (**2**).


No. Time Source Destination Protocol Info
274 0.078889 192.168.3.50 192.168.200.227 TCP http > vrtp [ACK] Seq=1 Ack=59884 Wir
275 0.380166 192.168.200.227 192.168.3.50 TCP [TCP Retransmission] [TCP segment of
276 0.983678 192.168.200.227 192.168.3.50 TCP [TCP Retransmission] [TCP segment of
2 2.195589 192.168.200.227 192.168.3.50 T 1 [TCP Retransmission] [TCP segment of
278 4.604757 192.168.200.227 192.168.3.50 TCP [TCP Retransmission] [TCP segment of
279 9.432867 192.168.200.227 192.168.3.50 TCP [TCP Retransmission] [TCP segment of
280 18.989050 192.168.200.227 192.168.3.50 TCP rcts > http [SYN] Seq=0 Win=65535 Ler
281 18.994054 192.168.3.50 192.168.200.227 T 3 http > rcts [SYN, ACK] Seq=0 Ack=1 Wi
282 18.994085 192.168.200.227 192.168.3.50 TCP rcts > http [ACK] Seq=1 Ack=1 Win=655
283 18.994264 192.168.200.227 192.168.3.50 TCP [TCP segment of a reassembled PDU]
284 18.994280 192.168.200.227 192.168.3.50 TCP [TCP segment of a reassembled PDU]

How it works...

Well, how databases work was always a miracle to me. Our task is to find how they influence the network, and this is what we've learned in this section.

There's more...

When you right-click on one of the packets in the database client to the server session, a window with the conversation will open. It can be helpful to the DBA to see what is running over the network.

When you are facing delay problems, for example, when working over cellular lines over the Internet or over international connections, the database client to the server will not always be efficient enough. You might need to move to web or terminal access to the database.

An important issue is how the database works. If the client is accessing the database server, and the database server is using files shared from another server, it can be that the client-server works great; but the problems come from the database server to the shared files on the file server. Make sure you that know all these dependencies before starting with your tests.

And most importantly, make sure you have very professional DBAs in your friends. One day you will need them.

Chapter 12. SIP, Multimedia, and IP Telephony

In this chapter, we will learn how to use Wireshark in order to resolve and troubleshoot IP telephony, voice and video calls, video streams, and other types of multimedia sessions. In this chapter we have the following recipes:

- Using Wireshark's features for telephony and multimedia analysis
- Analyzing SIP connectivity
- Analyzing RTP/RTCP connectivity
- Troubleshooting scenarios for video and surveillance applications
- Troubleshooting scenarios for IPTV applications
- Troubleshooting scenarios for video-conferencing applications
- Troubleshooting RTSP

Introduction

Various types of multimedia applications take up a significant part of modern communication networks. Among these applications, we have telephony, video conferencing, surveillance systems, distance-learning systems, and many more.

In multimedia applications, the requirements from the network are different from the requirements in other enterprise applications. While applications such as HTTP, e-mail, and file sharing require high bandwidth, a telephone call, for example, requires less than 100 Kbps, but is sensitive to delays, and very sensitive to jitter (delay variations). While most applications require high downstream to clients in remote offices, surveillance systems require the upstream direction as they are watched from monitors in the HQ, so the monitors actually download the video from the remote site.

In this chapter, we will focus on these voice, video, and multimedia applications, how they behave over network connections, and how to use Wireshark to troubleshoot problems when they don't work properly.

In this chapter, we will focus on **Session Initiation Protocol** (**SIP**), **Real Time Protocol** / **Real-time Transport Control Protocol** (**RTP**/**RTCP**), **Real Time Streaming Protocol** (**RTSP**), and other common multimedia protocols.

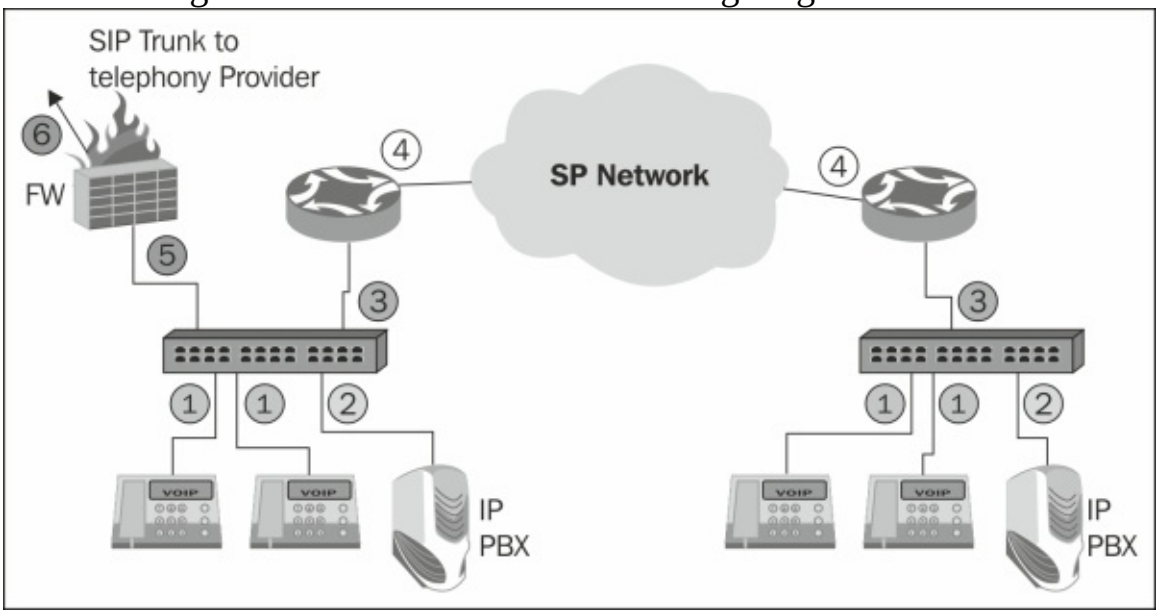
We will start this chapter by presenting the available tools that Wireshark provides us for troubleshooting voice and multimedia sessions. We will focus on how to resolve SIP problems and how to troubleshoot RTP/RTCP sessions, then we will learn how to troubleshoot video systems, including video conferencing and surveillance systems.

Using Wireshark's features for telephony and multimedia analysis

First, let's see what tools are provided by Wireshark for monitoring voice, video, and multimedia.

Getting ready

While facing problems with voice calls, video-conference calls, or other multimedia sessions, connect your laptop with Wireshark and port mirror one of the following devices as shown in the following diagram:

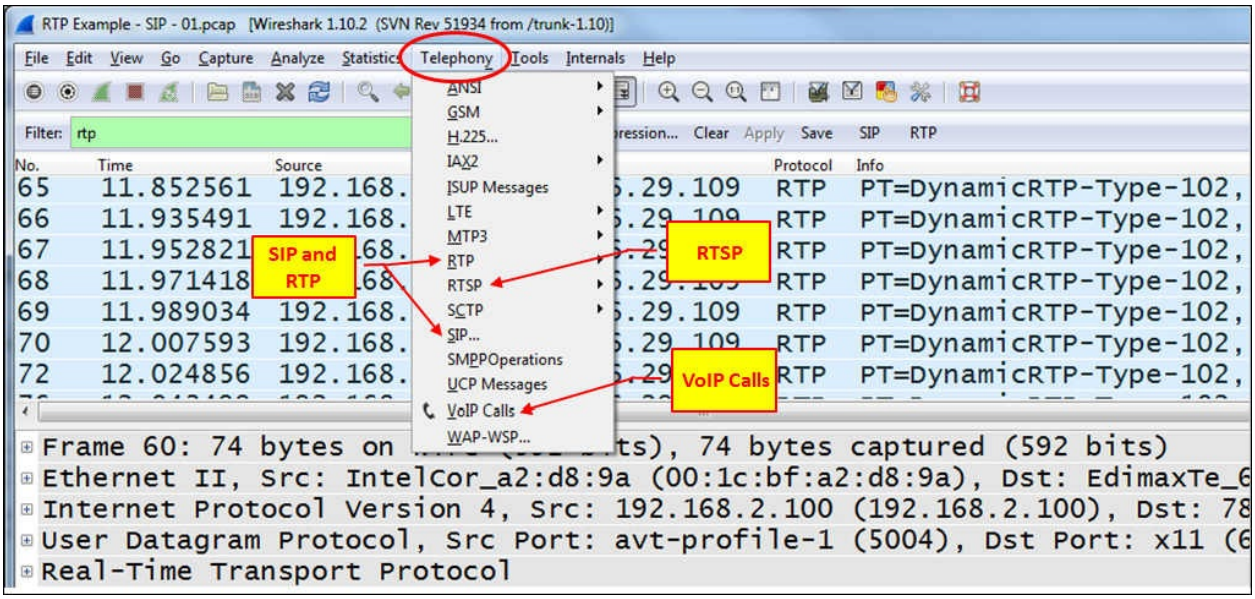


Follow these steps to use Wireshark's features for telephony and multimedia analysis:

1. Monitor the local or remote clients (**1**) in cases where you suspect a specific client problem.
2. Monitor the local or remote **IP PBX** system (**2**) when you suspect a central problem that influences the entire IP Telephony network.
3. Monitor the connections to the router (**3** and **4**) while suspecting an interoffice connectivity problem.
4. Monitor the firewall on the LAN port (**5**) or on the connection to the service provider (**6**). This connection will usually be over the Internet, but can also be on a direct line to the provider.

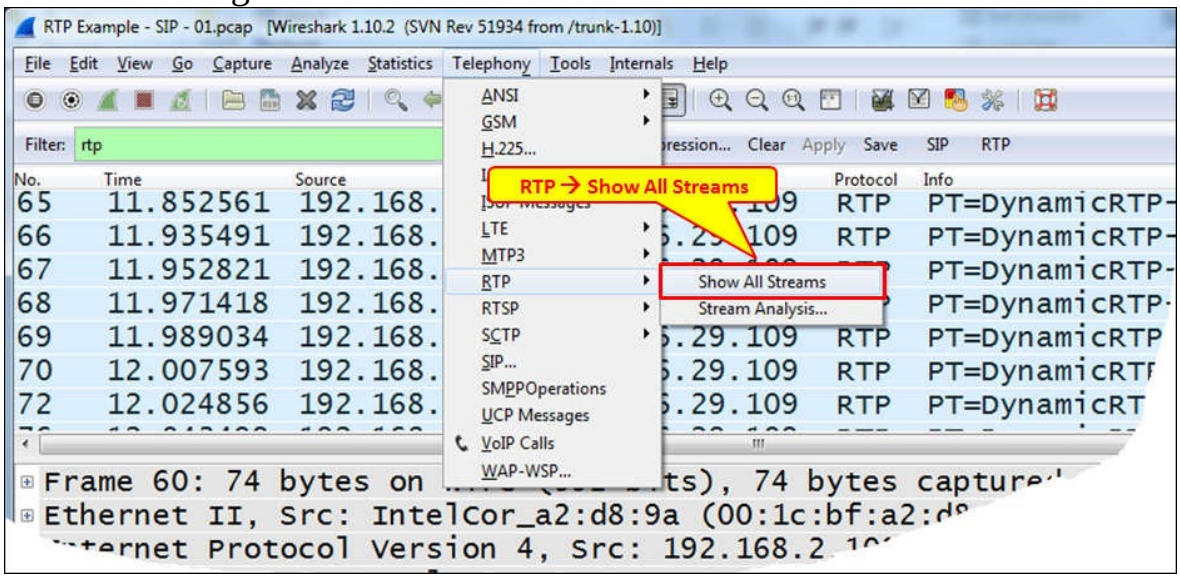
How to do it...

In the Wireshark window, open the **Telephony** menu, as shown in the following screenshot:

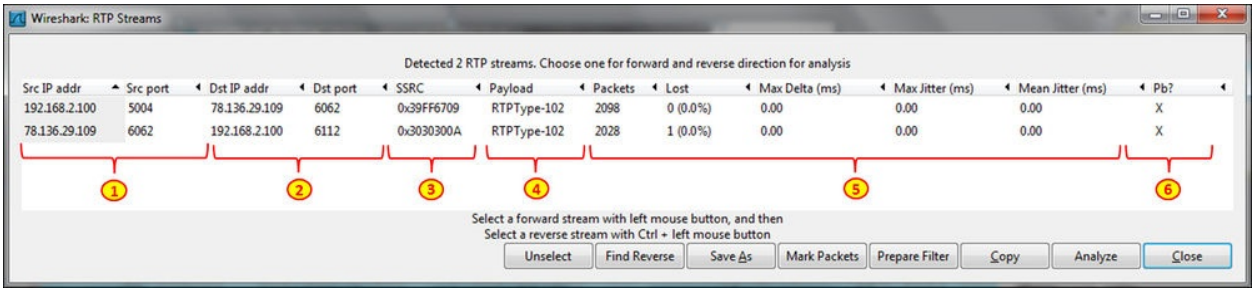


For telephony networks, use the following menus:

1. To view RTP information, navigate to **RTP** | **Show All Streams** as shown in the following screenshot:



- The following window will open:

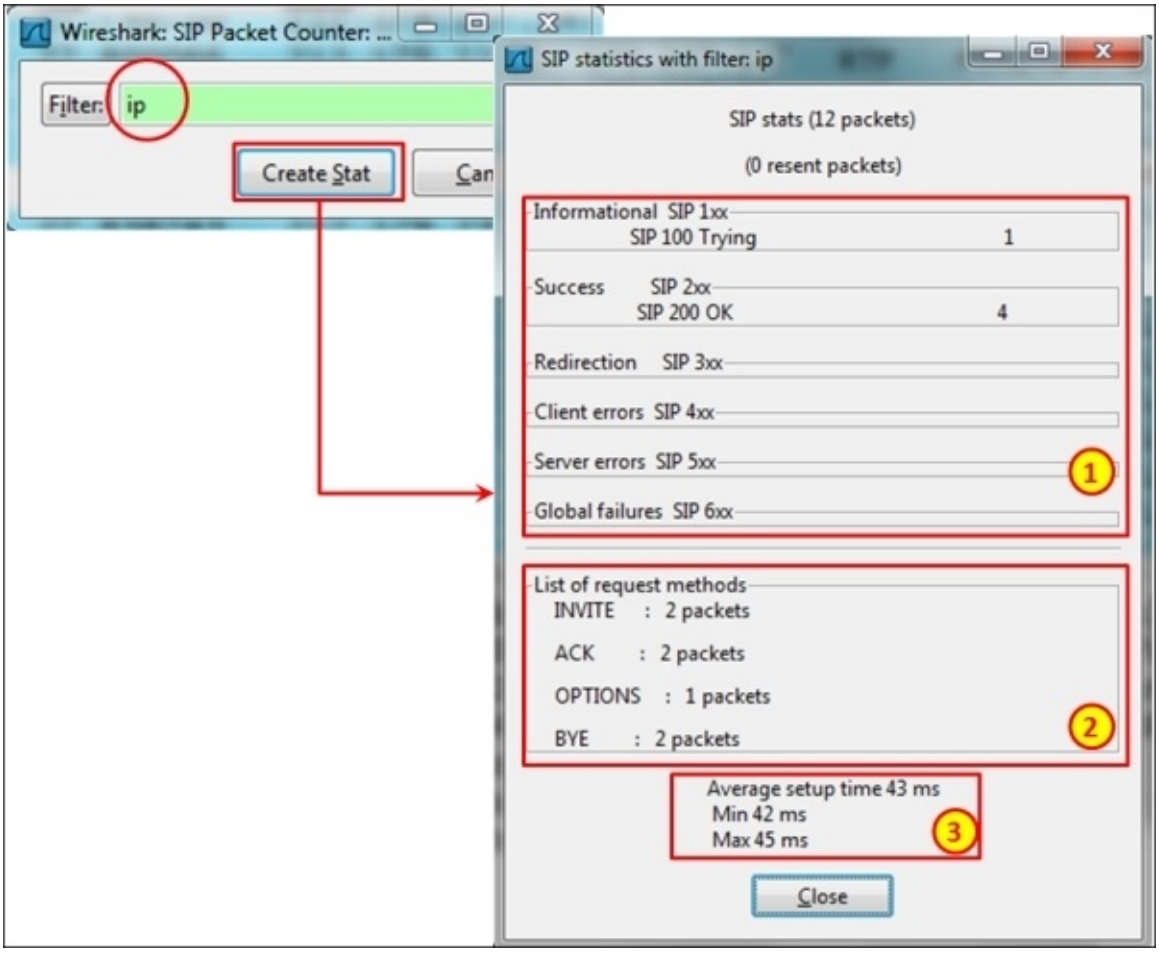


- In the RTP Streams window, you will see the following details:
  - The source IP address and UDP port
  - The destination IP address and UDP port
  - Synchronization Source (**SSRC**), which is an RTP stream identifier
  - RTP payload type (usually codec type)
  - Stream data, which is the total amount of captured packets, packet loss, maximum time between packets, maximum, and mean jitter
  - **Pb?** that indicates a general problem in the stream

In the lower part of the window, you have the following buttons:

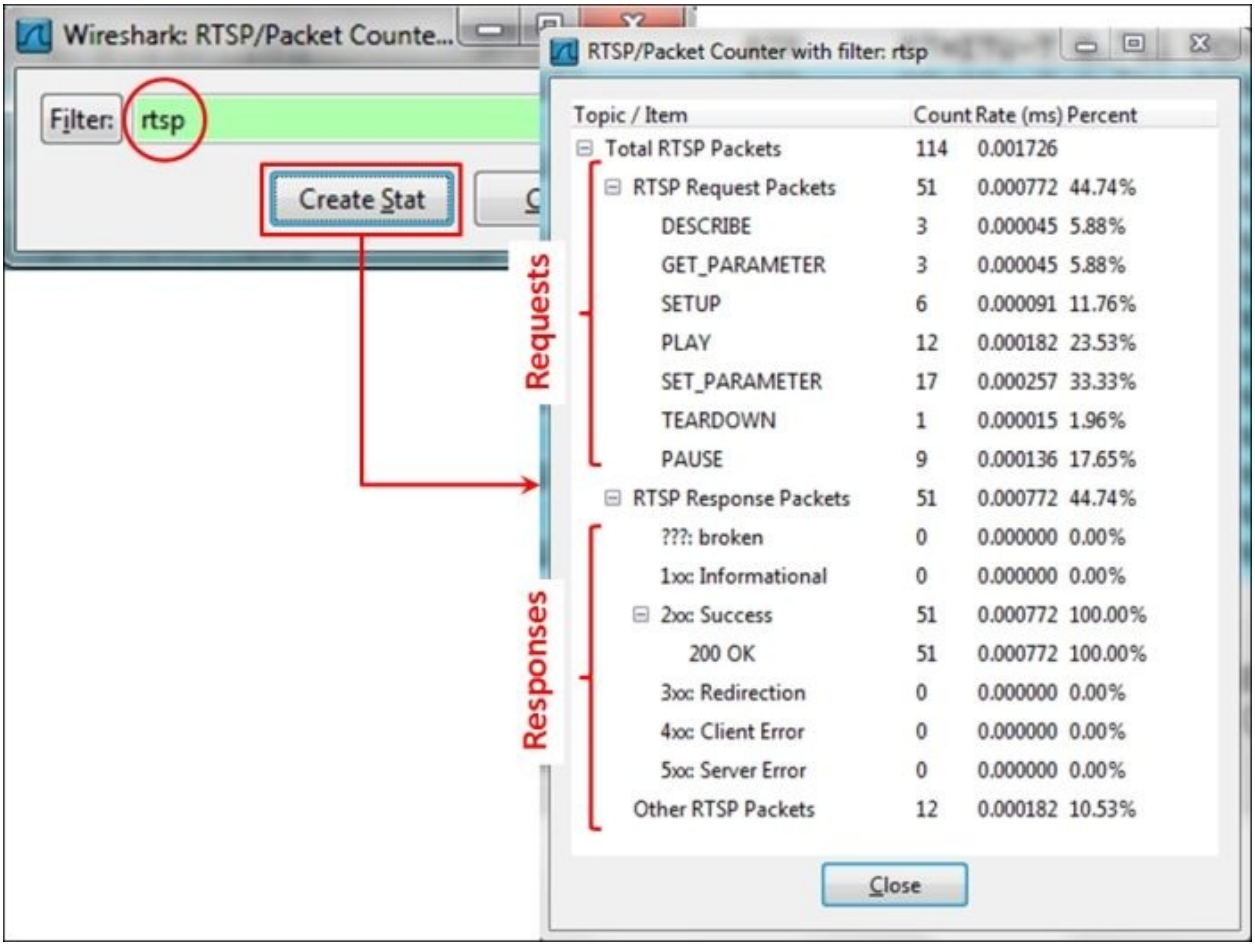
- **Unselect**: When you select a stream by clicking on its line, the **Unselect** button will cancel the selection.
- **Find reverse**: On a voice or multimedia call, a reverse stream is the stream in the opposite direction (which will be highlighted in light gray).
- **Save as**: This button can be used to save a stream in the `rtpdump` format. For information about the format, go to http://www.cs.columbia.edu/irt/software/rtptools/.
- **Mark packets**: not functioning.
- **Prepare filter**: This prepares a display filter in the display filter window.
- **Copy**: This option copies the RTP streams to a text file. For doing so, click on **Copy**, open a text editor, and paste the content to the text file.
- **Analyze**: When you click on a stream, and then click on the **Analyze** button, it opens the **RTP Stream Analyze** window. The same window can be opened by clicking on an RTP packet and navigating to **Telephony** | **RTP** | **Stream Analysis** from the menu.
- **Close**: Clicking on this button closes the window.

- To view SIP information, navigate to **Telephony** | **SIP**. Enter `ip` (or `udp` or `sip`) in the **SIP Packet Counter** window that opens, and the window **SIP statistics with filter: ip** will open as shown in the following screenshot:

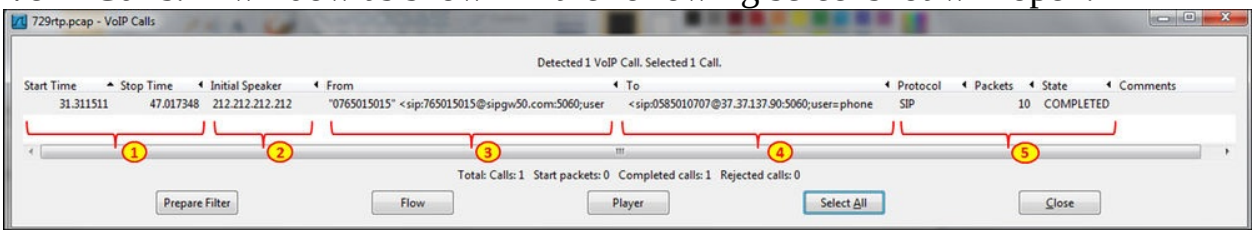


1. In the window, you will get the following SIP statistics:
    - Number of packets sent with SIP response codes (numbered as **1** in the preceding screenshot)
    - Total number of each one of the SIP methods (these are SIP commands) that were sent (numbered as **2** in the preceding screenshot)
    - Minimum, maximum, and average session setup times (numbered as **3** in the preceding screenshot)

- For RTSP statistics, navigate to **Telephony** | **RTSP** | **Packet Counter**, write `ip`, `rtsp`, or just leave it blank in the pop up that comes up, and then click on **Create Stat** that opens a window as shown in the following screenshot:



- For watching which telephone calls were captured, navigate to **Telephony** | **VoIP Calls**. A window as shown in the following screenshot will open:



- In the VoIP calls, you see the following parameters:
  - The start and end time that give the duration of the call (numbered as **1** in the preceding screenshot)
  - The IP address from where the session had started (numbered as **2** in the preceding screenshot)
  - The SIP address from where the session started (numbered as **3** in the preceding screenshot)
  - The SIP address for whom the session was intended (numbered as **4** in the

preceding screenshot)

- The protocol of the session (usually SIP), the number of protocol packets, and the session status (numbered as **5** in the preceding screenshot)

How it works...

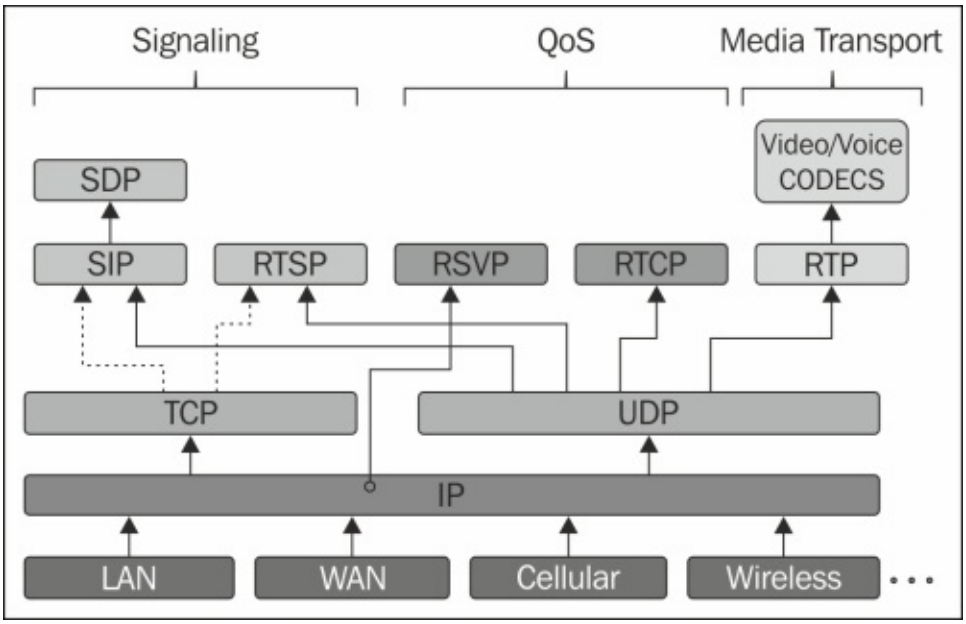
For transferring voice, video, or multimedia, we need two functions to be performed. The first one is to carry the media stream, which is mostly voice or video, and the second one is the signaling, which is to establish and terminate the call, to invite participants to the call, and so on. Two protocol suites were proposed over the years for the signaling:

- The **ITU-T** suite of protocols, including **H.323** as an umbrella protocol for the suite, **H.225** for registration and address resolution, **H.245** for control, and some others.
- The **IETF** suite of protocols including SIP as a signaling protocol (RFC 3261 with later updates) and **Session Description Protocol** (**SDP**) that describes the session parameters (RFC 4566).

The ITU-T set of protocols phased out in the last few years, and the majority of the applications today are using the IETF set of protocols, on which we will focus in this chapter. In the following diagram you see the structure of the IETF protocol suite.

For the stream transfer, both suites use RTP and RTCP (RFC 3550 with later updates). RTP is used for the media transfer, and RTCP for controlling the quality of the stream.

There are several protocols for transferring multimedia sessions over an IP-based network, as shown in the following diagram:



SIP is used for signaling the structured packets that are sent between end clients. **SDP** data is carried by SIP messages for the description of the session. RTSP is used for controlling streams, usually video transmissions (typically IPTV streams).

RTP is used for carrying the media. Above **RTP**, we have various types of codec for voice and video compression.

RTCP is used for controlling the quality of the stream, and RSVP is a protocol for establishing the quality of service through the network.

All these protocols are carried by the TCP/IP protocol suite, as shown in the preceding diagram. Later, you will find in this chapter a detailed description for most of them.

There's more...

SIP uses fixed port numbers; therefore, Wireshark will always refer to these ports as an SIP session—port 5060 for SIP and port 5061 for SIP over TLS (which is SIP secured with TLS). The standard allows using SIP over TCP or UDP, but in the majority of the cases, it will be used over UDP.

RTP and RTCP, on the other hand, do not use standard fixed ports. RTP uses even port numbers, and the corresponding RTCP stream uses the next higher odd port numbers. For example, if RTP uses port 5004 on one end, and port 2006 on the other, RTCP will use ports 5005 and 2007 respectively. This is why Wireshark, by default, will not resolve RTP, RTCP, and such others, but it will show you UDP traffic instead.

To resolve it, you can do the following:

1. Right-click on a packet in the RTP stream (which currently looks like UDP) and click on **Decode as**. In the window that opens, select **RTP**.
2. You can go to **Edit** | **Preferences**, and from the protocol list, choose **RTP**. In the RTP window, select **Try to decode RTP outside of conversations**, and in most cases, RTP will be decoded automatically in this manner. You can do the same for RTCP.

Analyzing SIP connectivity

As we learned in the previous recipe, **SIP** (RFC 3261 and various extensions) is a signaling protocol that is used for creating, modifying, and terminating user sessions between one or more participants. While sending SIP requests, the session parameters are sent via **SDP** (**SDP**, RFC 4566) which enable users to agree on a set of compatible media types between them. When sessions are created, the voice or video is carried by RTP and optionally controlled by **RTCP** (RTCP is optional, and can be used by multimedia applications, but it is not a mandatory protocol).

SIP defines endpoints as **User Agents** (**UAs**), and the process of creating a session involves UA negotiation in order to agree on a characterization of a session that they would like to create. For additional services such as locating session participants, registration, call forwarding, and others, SIP defines network hosts called **servers** to which UAs can send registrations, invitations to sessions, and other requests.

In this recipe, we will discuss the signaling part of the protocol suite, which is SIP, and how to use Wireshark in order to troubleshoot signaling problems, while in the next recipe, we will learn about **RTP** and **RTCP**.

Problems in voice and video over IP can be categorized in to two groups:

- **Problems of call establishment, modification, and termination**: These will be problems such as instances when you pick up the phone and you don't hear a dial tone, you hear the dial tone but cannot dial a number, you dial a number but the other side doesn't get the call, and so on. These types of problems are usually caused by signaling issues.
- **Problems of quality:** These are problems related to quality, such as voice quality, disturbances (like noise) during the call, video freezes, and so on, which are usually caused by networking problems, RTP problems, or various types of media issues.

In this recipe, we will discuss the first type, and in the next recipe the second type.

Tip

It is important to note that not all problems in this area are networking problems, and in many cases, they will be a result of bad configuration of the equipment (for example, telephony switches or end devices).

Getting ready

While facing problems of the first type, in most of the cases you are having signaling problems. Connect the Wireshark to the network, and follow the steps in this recipe in order to solve it.

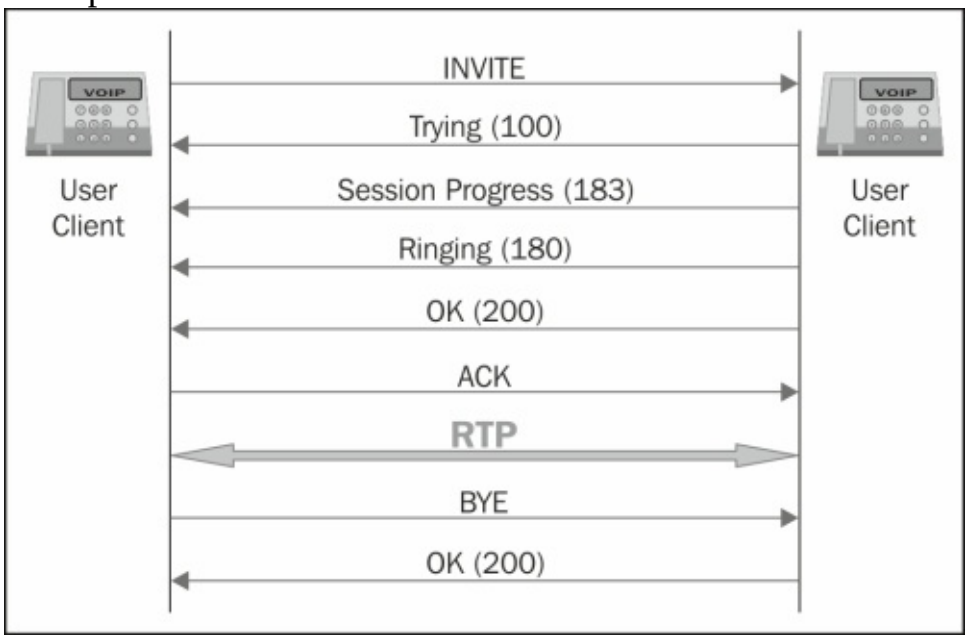
We have two major types of messages in SIP: **methods** and **responses**. Methods are commands initiated by one side of the session, and responses are generated as a reply.

While troubleshooting an SIP session, keep track of the responses and what they say. Only the major types are brought here.

How to do it...

After you've connected Wireshark to the network, follow these steps:

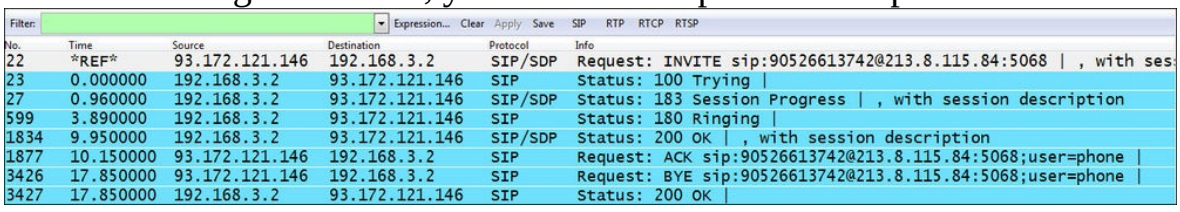
1. When a UA desires to establish a multimedia session, it sends an **INVITE** method to the remote UA. In the following diagram, you can see an example for a basic call flow.



Tip

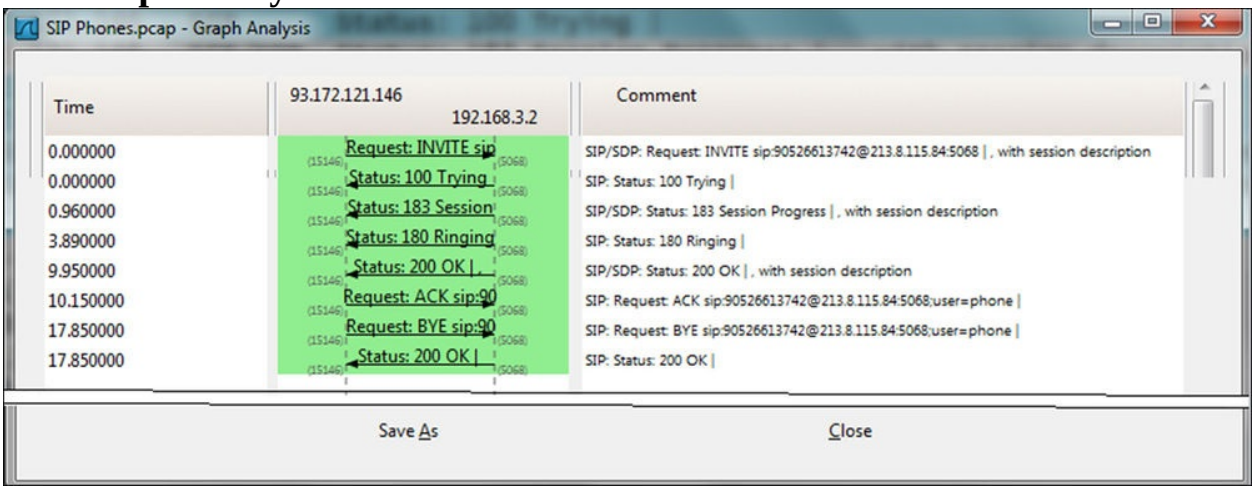
An end device in SIP is called **User Agent** (**UA**). A user agent can initiate or receive a call. A UA can be an IP phone, video camera, software client, or any device or software that participates in an SIP session.

2. In the following screenshot, you see an example for a telephone call flow:



- To see the detailed call flow, navigate to **Statistics** | **Flow Graph**, and mark the following:
    - Displayed packets
    - General flow
    - Standard source/destination addresses

The **Graph Analysis** window is shown for reference as follows:



- After **INVITE**, you should see **Trying**, **Session Progress**, **Ringing**, or a combination of them coming from the other side.
- We can see here how the session progresses between the initiator on **93.172.121.146** to the responder on **192.168.3.2**:
    1. The **INVITE** method is sent from the session initiator; this will always be the first packet that starts the conversation.
    2. The responder answers with **Trying** (code **100**), **Session progress** (code **183**), and after three seconds with **Ringing** (code **180**). Then it answers with **OK** (code **200**), meaning that the handset was picked up.

        ## Tip

        Not all these messages should be there, and in some cases, you will see only some of them, and it is still okay. Later in this chapter, we will see how to find out whether it is a problem or just standard protocol behavior.

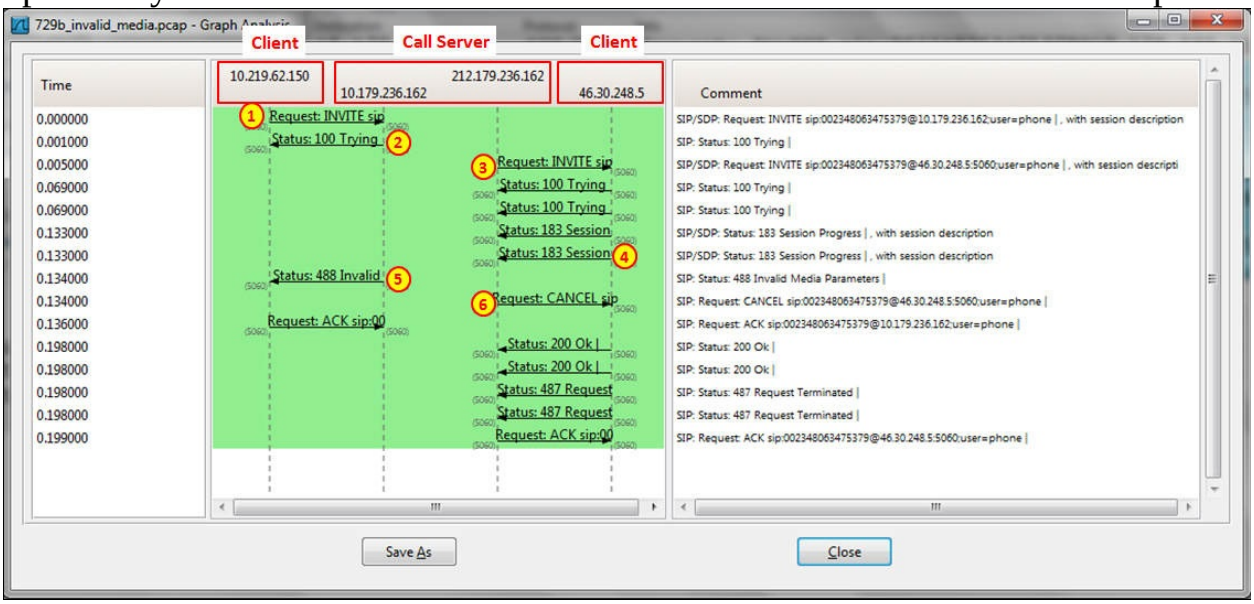
    3. The initiator sends ACK, and the session is established.
- If an error message is received at this stage, the connection will not be

established.

Tip

Don't forget that SIP works over UDP, and since UDP does not open any connection to the other side before sending the request, it can be possible that a request will not arrive to the destination simply because of a network-reachability problem. For this reason, when you don't get a response, it can be that **INVITE** simply didn't get to the destination because of a network problem.

- When there is a telephone switch between the user clients, the session will look like the one shown in the following screenshot. You will hear the term IP telephone switch, Call manager, IP PBX, and others. They all mean a telephony switch that handles the signaling between devices. The SIP terms together make up a Proxy that we will talk about in the *How it works...* section of this recipe.



- Here you see that the switch has two interfaces—the first one on the internal network (**10.179.236.162**), and the second one on the external network (**212.179.236.162**):
  1. The client on the left, **10.219.62.150**, sends an **INVITE** request to the switch (numbered as **1** in the preceding screenshot).
  2. The switch replies by saying that it is **Trying** (numbered as **2** in the preceding screenshot).
  3. The switch sends **INVITE** to the client on the right (numbered as **3** in the

preceding screenshot).

4. The client sends **Trying** (code **100**), and then the session progresses (code **183**) to the switch (numbered as **4** in the preceding screenshot).
5. After a while, the switch sees that the client has not responded, and to notify the initiator, it sends them code **488**, which means invalid or not acceptable, with an explanation as to why it was not accepted (numbered as **5** in the preceding screenshot).
6. The switch sends a **Cancel** message to the client on the right (numbered as **6** in the preceding screenshot).

- To allocate problems in SIP, do the following:
  1. Draw the network with all of its components.
  2. Check for the error codes.
  3. Figure out the reason for the errors.

- SIP error codes are listed in the following table, along with their possible reasons. Unless mentioned otherwise, codes are defined in RFC 3261.

1xx codes – provisional/informational

The 1xx codes or provisional/informational codes are those where the received request is still in process, and the receiver notifies the sender about it. They are described in detail in the following table:

Code	Event Name	Reason
100	Trying	The request has been received and accepted by the server, and an action is being taken for this call.
180	Ringing	The UA that received the call is alerting the end user. This is the message that is sent back to the client while doing so.
181	Call forward	The call is being forwarded to another destination.
182	Queued	The called party is temporarily unavailable, and the server saves the message for later delivery.
183	Session	The session is being handled by the receiving server. Additional details on the call

	progress	progress can be conveyed in the message header.

2xx codes – success

The 2xx codes or the success codes indicate that the action was successfully received, understood, and accepted. They are described in detail in the following table:

Code	Event Name	Reason
200	Ok	The request has been accepted, processed, and it succeeded.
202	Accepted	The request has been accepted for processing, but the processing of it has not been completed (RFC 3265).

3xx codes – redirection

The 3xx codes indicate that a redirection action needs to be taken in order to complete the request. They are described in detail in the following table:

Code	Event Name	Reason
300	Multiple choices	The address in the request was resolved to several choices, and the accepting server can forward it to one of them. The UA can use the addresses in the contact header field for automatic redirection, or confirm it with the sender before redirecting the message.
301	Moved permanently	The user could not be located at the address in the Request-URI, and the requesting client should try at the address provided in the contact header field. The sender should update its local directories with the change.
302	Moved temporarily	The requesting client should retry the request at the new address/addresses provided in the contact header field.
305	Use proxy	The requested resource must be accessed through the proxy, whose address is given by the contact field.
380	Alternative	The call was not successful, so the recipient sends this response for alternative

	service	services to be made available on the receiver. These services are described in the message body.

4xx codes – client error

The 4xx codes or client error indicate that the request contains bad syntax or cannot be fulfilled in this server. They are described in detail in the following table:

Code	Event Name	Reason
400	Bad request	The request couldn't be processed due to syntax error.
401	Unauthorized	The request that was received requires user authentication. Usually the client will ask the user for it.
402	Payment required	This is reserved for future use.
403	Forbidden	The server has understood the request, but is refusing to perform it. The client should not try it again.
404	Not found	The server notifies the client that the user does not exist in the domain specified in the Request URI.
405	Method not allowed	A method sent by the client is not allowed to be used by it. The response will include an `allow` header field to notify the sender which methods he is allowed to use.
406	Not acceptable	The resource identified by the request is only capable of generating response entities that have content characteristics not acceptable according to the `accept` header field sent in the request.
407	Proxy authentication required	The client must authenticate with a proxy server.
408	Request timeout	The server couldn't respond during the expected time. The client may send the request again after a while.

410	Gone	The requested resource is no longer available at the server, and the forwarding address is not known. This condition is considered to be permanent.
413	Request entity too large	The server is refusing to process a request because the request entity's body is larger than what the server is able or willing to process.
414	Request-URI too long	The server is refusing to service the request because the Request URI is longer than what the server is able or willing to interpret.
415	Unsupported media type	The server is refusing to process the request because the message body of the request is in a format that is not supported by the server.
416	Unsupported URI scheme	Request URI is unknown to the server, and therefore, the server cannot process the request.
420	Bad extension	The server did not understand the protocol extension received from the client.
421	Extension required	The UA that received the request requires a particular extension in order to process it, but this extension is not listed in the supported header field of the request.
423	Interval too brief	The server is rejecting the request because the expiration time of the resource refreshed by the request is too short.
424	Bad location information	This response code indicates a rejection of the request due to its location contents. This indicates malformed or not satisfactory location information (RFC6442).
428	Use Identity header	It is sent when a verifier receives an SIP request that lacks an Identity header in order to indicate that the request should be re-sent with an Identity header (RFC4474).
429	Provide referrer identity	This provides referrer identity (RFC3892).
433	Anonymity disallowed	This indicates that the server refused to satisfy the request because the requestor was anonymous (RFC5079).
436	Bad identity info	This response is used when there is bad information in the Identity-Info header (RFC4474).
437	Unsupported	This is used when the verifier cannot validate the certificate referenced by the URI

	certificate	in the Identity-Info header (RFC4474).
438	Invalid identity header	This is used when the verifier (the receiver UA) receives a message with an Identity signature that does not correspond to the digest-string calculated by the verifier (RFC4474).
470	Consent needed	This is the response to a request that contained a URI list in which at least one URI was such that the relay had no access permissions (RFC5360).
480	Temporarily unavailable	The callee's end system was contacted successfully, but the callee is currently unavailable.
481	Call/transaction does not exist	The receiving UA received a request that does not match any existing transaction or dialog.
482	Loop detected	The server has detected a loop.
483	Too many hops	The server received a request that contains a Max-Forwards header field that equals zero.
484	Address incomplete	The server received a request with an incomplete Request-URI.
485	Ambiguous	The Request-URI was unclear. The response may contain a listing of possible addresses in the Contact header fields.
486	Busy here	The callee's end system was contacted successfully, but the callee is currently unable or unwilling to take additional calls by this end system.
487	Request terminated	The request was terminated by a BYE or CANCEL request.
488	Not acceptable here	Specific resources addressed by the Request-URI are not accepted.
491	Request pending	The receiving UA had a pending request.
493	Undecipherable	The request contains an encrypted MIME body, which cannot be decrypted by the recipient.

5xx codes – server error

The 5xx codes or server error codes indicate that the server failed to fulfill an apparently valid request. They are described in detail in the following table:

Code	Event Name	Reason
500	Server internal error	An unexpected condition prevented the server from fulfilling the request.
501	Not implemented	The functionality that requested to fulfill the request is not supported by the server.
502	Bad gateway	A gateway or proxy received an invalid response from the downstream server it accessed while attempting to fulfill the request.
503	Service unavailable	The server is temporarily unable to process the request due to temporary overloading or maintenance of the server.
504	Server time out	The server processing the request has sent the request to another server in order to process it, and the response did not arrive on time.
505	Version not supported	The server does not support the SIP protocol version that is used in the request.
513	Message too large	The server was unable to process the request since the message length is too long.

6xx codes – global failure

The 6xx codes or global failure codes indicate that the request cannot be fulfilled at any server. They are described in detail in the following table:

Code	Event Name	Reason
600	Busy everywhere	The recipient's end system was contacted successfully, but the user is busy and does not wish to take the call at this moment.

603	Decline	The receiving UA was successfully contacted, but the user explicitly does not wish to or cannot participate.
604	Does not exist anywhere	The server has authoritative information that the user indicated in the Request URI, which does not exist anywhere.
606	Not acceptable	The US was contacted successfully, but some aspects of the session description described by SDP were not acceptable.

How it works...

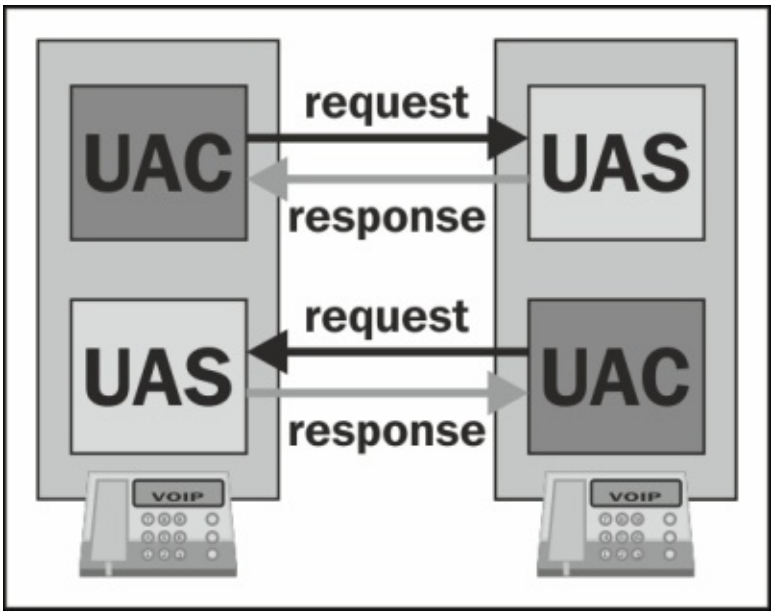
SIP is an application-layer control protocol that is used to establish, maintain, and terminate calls between two or more end nodes.

SIP defines two basic classes of network entities—clients and servers:

- A client is an entity (or application) that sends SIP requests
- A server is an entity (or application) that responds to those requests

For connectivity to other network types, we have gateways. A gateway connects between SIP and **Public Switched Telephone Networks** (**PSTN**), or SIP and H.323.

As illustrated in the following diagram, a client is made of **User Agent Client** (**UAC**) and **User Agent Server** (**UAS**), and each client can initiate or respond to requests.

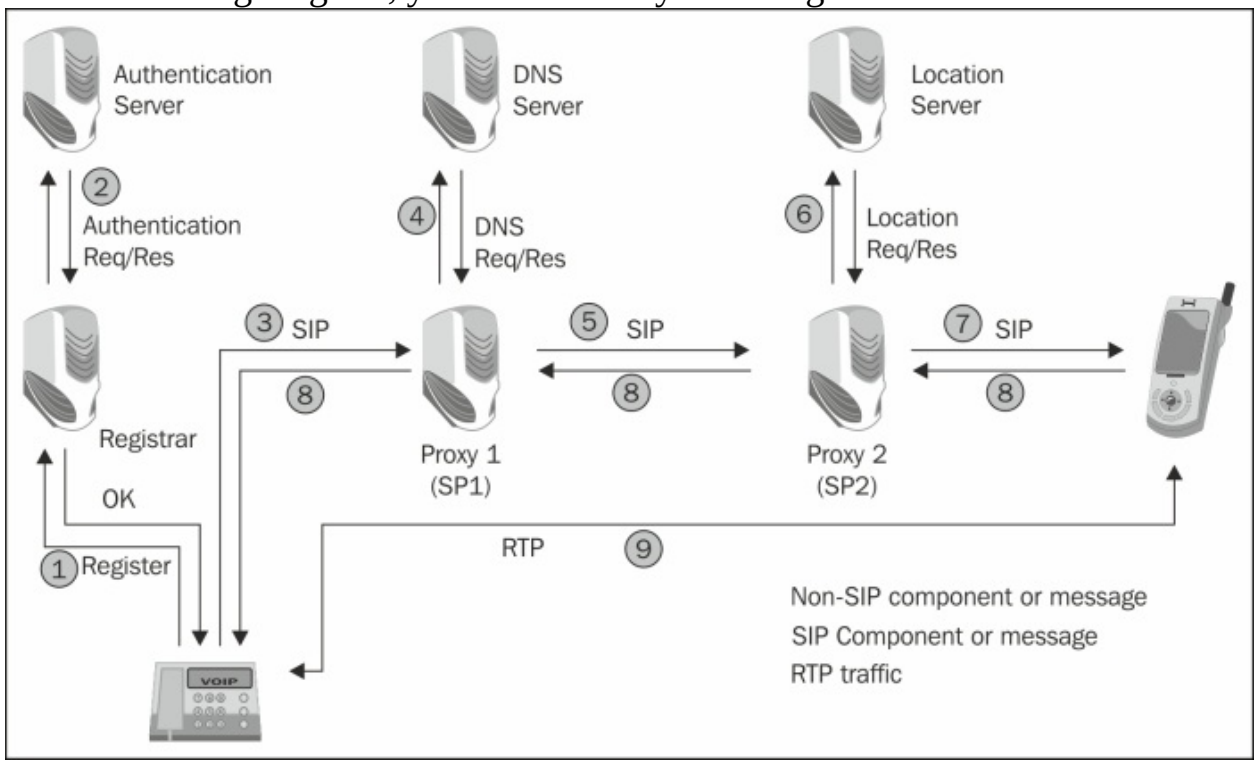


SIP servers can be of various types:

- **Proxy Server**: This receives SIP requests from a user agent or another proxy and forwards or proxies the request to another location

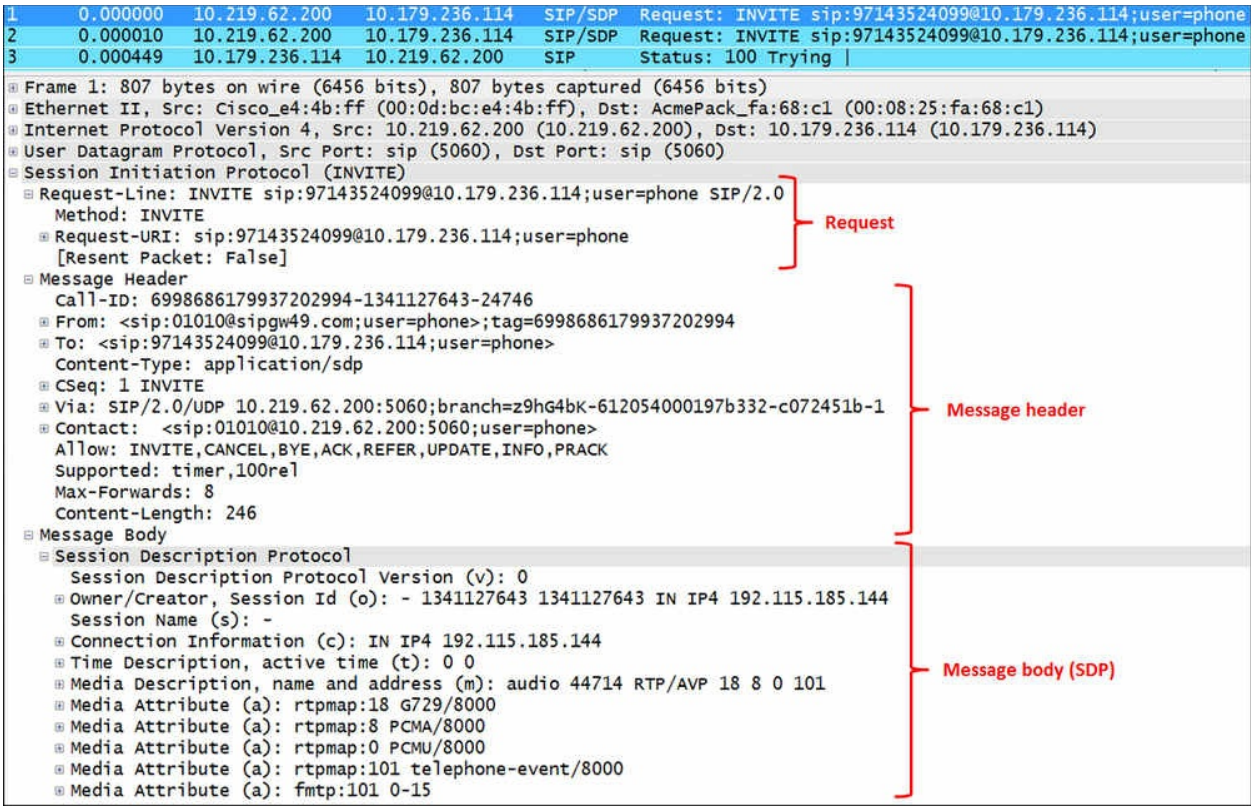
- **Redirect Server**: This receives requests from a user agent or proxy, and returns a redirection response (3xx), indicating where the request should be present
- **Registrar**: This receives SIP registration requests and updates the user agent information to a location service or other database

In the following diagram, you see how they all fit together:



An IP phone registers to the registrar (**1**). The registrar checks with the organization server (**2**), and if it's all OK, it sends an SIP request to the provider's proxy (**3**). The provider's proxy checks with the DNS server for the IP address of the requested client's domain (**4**), and then it forwards the requests to the destination proxy (**5**). The destination proxy sees that the client is not in its place and checks with the location server for its location (**6**). When found, the SIP request is forwarded to the destination client (**7**). The destination client confirms the acceptance of the request to the sender (**8**). When all is okay, an RTP session is opened on the UDP ports described in SDP when opening the session (**9**).

The SIP message is built as you see in the following diagram:



The first part is the **Request** line in which we have:

- The method; **INVITE** in this case
- The requested URI

The second part is **Message Header**, in which we have:

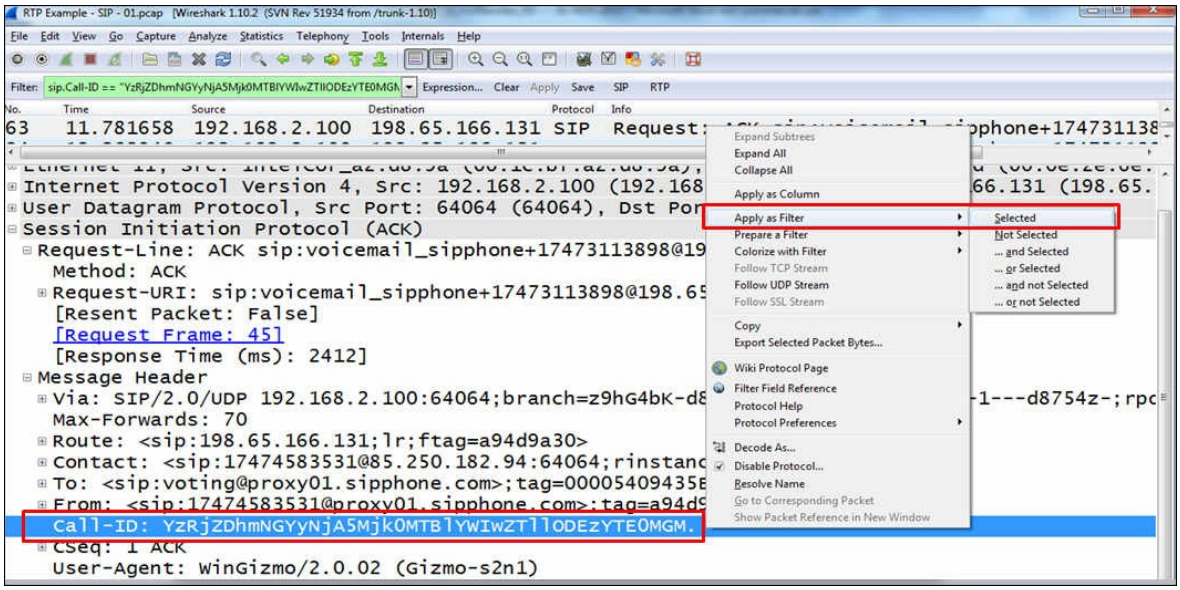
- **Call-ID**: This provides a unique ID for the call
- **From**: This indicates the initiator of the call
- **To**: This indicates the destination of the call
- **CSeq**: This contains the sequence number, which contains an integer followed by the request method. Each successive request during the call will have a higher CSeq number, and the caller and called parties each maintain their own separate CSeq counts.
- **Via**: This indicates the path taken by the request so far and indicates the path that should be followed in routing responses.
- **Contact**: This contains one or more SIP URIs that provide the other party in the session with information for contacting the initiating user

- **Allow**: This indicates which methods are allowed
- **Supported**: This indicates whether parameters such as timers are supported
- **Max-Forwards**: This is the maximum number of hops to pass to the destination
- **Content-Length**: This is the byte count of the message body
- **Message Body**: This contains information on the codecs that are supported by the sender; for example, timers supported

There's more...

When debugging a phone call, first filter the call with the **Call-ID** parameter. To do so, you can do one of the following:

- Look for the **Call-ID** parameter in the **Message Header** field in the SIP header, right-click on it, and select **Apply as Filter**, as illustrated in the following screenshot
- Use the **VoIP Calls** feature from the Wireshark menu



When debugging an SIP trunk that is signaling between IP PBXs, try to figure out whether there is there a specific call that doesn't work or whether all calls have the same problem.

To troubleshoot VoIP calls, the best way is to read the SIP messages. They will tell you what to do.

Analyzing RTP/RTCP connectivity

In the previous recipe, we talked about signaling, that is, SIP and RDP. In this recipe, we will see how to use the voice or video call itself and see what might go wrong with it.

It will always start with a user complaining about voice or video quality, low speech quality, noises, and so on.

Also, don't forget that it might look like everything works fine in Wireshark, but further tuning of the IP PBX should be done say to increase the transmit volume.

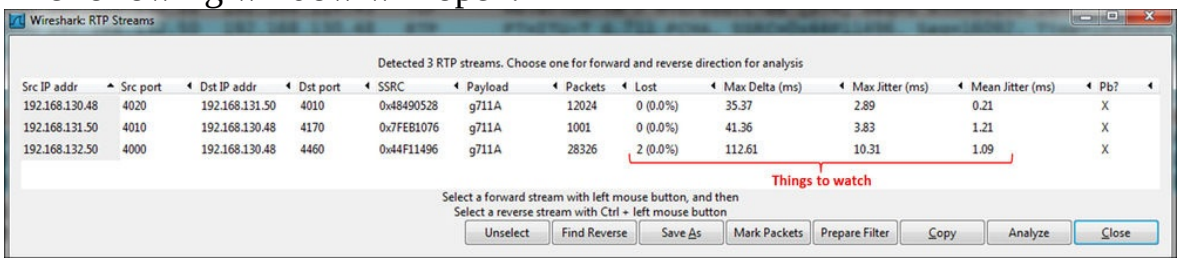
Getting ready

When facing problems on a specific client, connect Wireshark to the client port with a port mirror. When facing problems with all clients connected to the same link, connect Wireshark to the link with a port mirror.

How to do it...

To locate performance problems, follow these steps:

1. After you connect Wireshark with a port mirror, start the capture.
2. Make sure there are calls running.
3. From the **Telephony** menu, navigate to **RTP** | **Show All Streams**. This will show you all RTP streams running on the port that you are monitoring.
4. The following window will open:



- Parameters that are important to watch are the packets lost, **Max Delta (ms)**, **Max Jitter (ms)**, and **Mean Jitter (ms)**.

Tip

Delay values higher than 300 ms (RTT) and Jitter values higher than 50 ms are considered to be problematic for interactive voice and video over IP. Further discussion on this subject is in the *How it works...* section in this recipe.

In the case of delay, follow these steps to locate the problem:

1. Use a simple ping test to check the delay between the two ends of the network:
    1. When you see a high delay, check if it is typical to the communications line that you are measuring (see a list of typical delays in the *There's more...* section in this recipe).
    2. If it is a typical delay, you don't have anything to do here. Check with the phone and switch providers for tuning solutions for their equipment.
    3. If you have a longer delay than expected, ping the two phones from your laptop and check where the delay came from.
    4. When you locate the link with the higher delay, ping in a step-by-step

manner along the link to see where the delay came from.
5. In parallel, use Wireshark to check the load on the line. The delay can come from there, and in most of these cases, you will have Jitter coming with it.

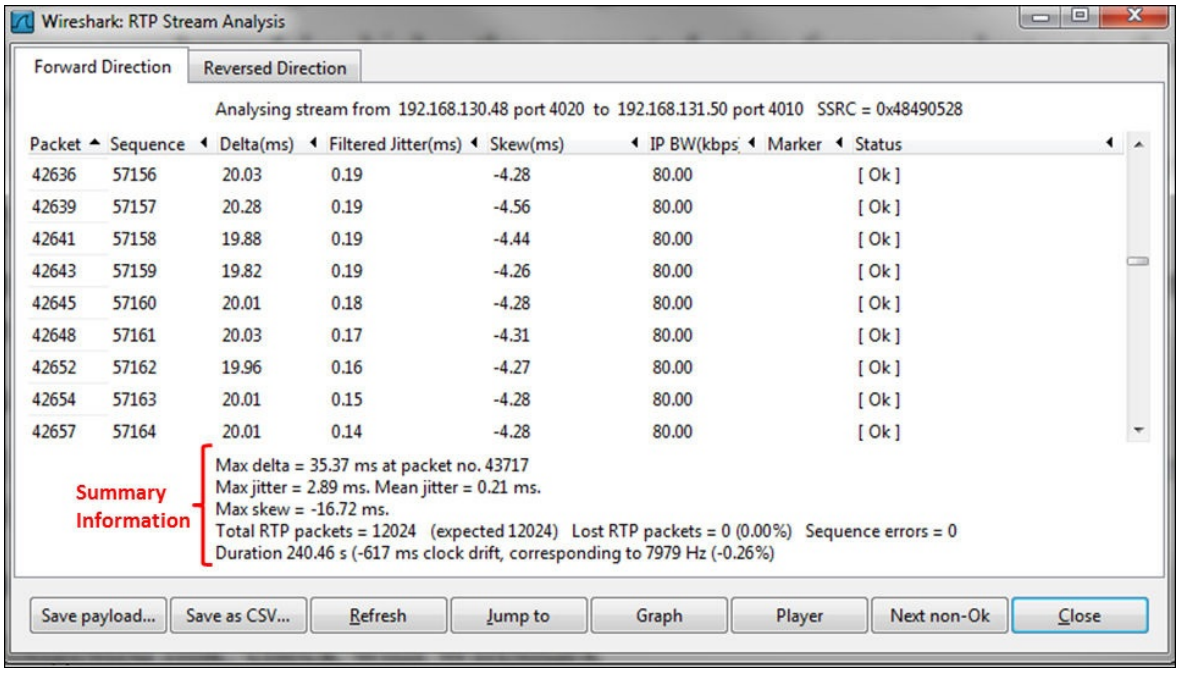
2. Delay can come from the following sources as well:
    - **Congested link**: Check the case using Wireshark.
    - **Load on a router**: Use provider tools, SNMP tools, or router CLI to measure load on these devices (use the provider manuals). It can be CPU load, memory load, and so on.
    - **Queuing delay on routers buffers**: Check the vendor manuals.

In the case of Jitter, follow these steps to locate the problem:

1. Use the same methodology as in the delay measurement and try to figure out where the Jitter comes from.
2. Jitter can come from several sources:
    - **Congested line**: Check the line with a `ping` command to see if you have any problems here
    - **Load on a router (CPU/memory)**: Check the vendor manuals to see how you can monitor these parameters

In the case of packet loss, follow these steps to locate the problem:

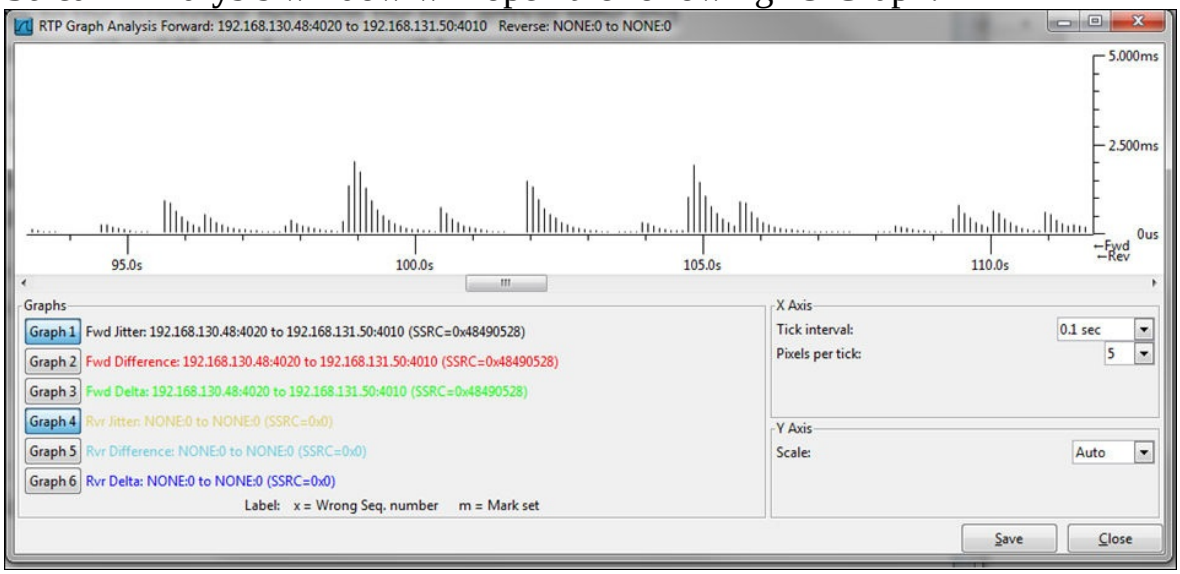
1. Check using the `ping` command to see if there is a packet loss across the link.
2. If so, check the equipment along the way to see for packet losses.
3. You can also click on a specific RTP packet in the packet list window and then navigate to **RTP** | **Stream Analysis**. It will show you the parameters on the stream that the packet is a part of. The following window will open:



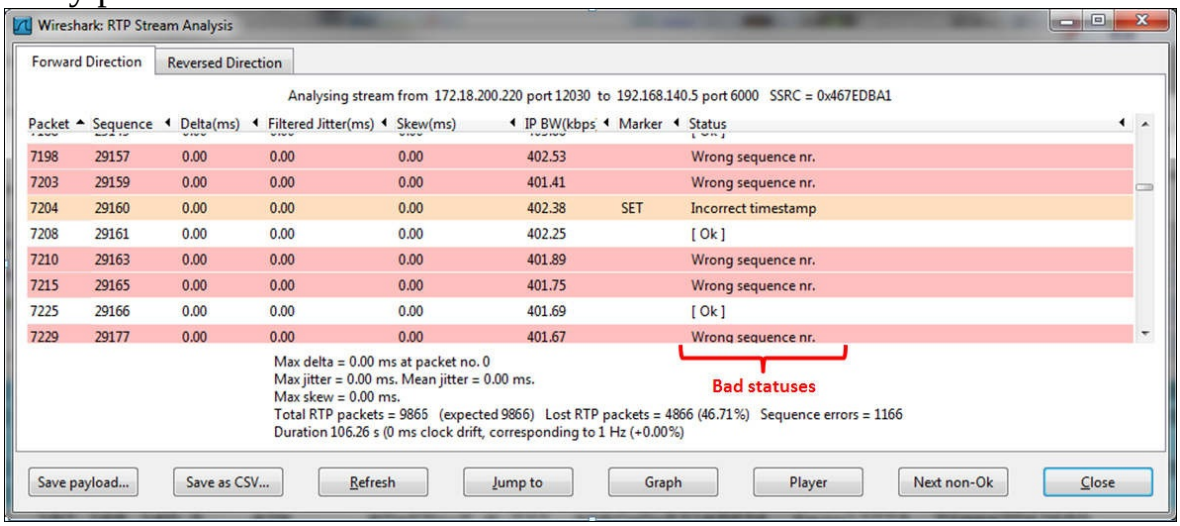
4. In the window, you will see the following parameters on the stream that you've opened:
    - **Packet**: This parameter denotes the number of packets in the captured file.
    - **Sequence**: This parameter denotes the RTP sequence number.
    - **Delta (ms)**: This is the time difference between the current and previous packet in the stream.
    - **Filtered Jitter (ms)**: This parameter refers to the difference between the real arrival time and the RTP timestamp parameter. It should be as low as possible and preferably zero.
    - **Skew (ms)**: This parameter denotes how early (or late) the packet is in relation to where it was supposed to be. For example, if we have a packet rate of 20 packets per second, we should have 50 ms between packets, and if a packet arrives 49 ms after the previous one, it will be a skew of -1 ms.
    - **IP BW (kbps)**: This parameter refers to the bandwidth consumption at the IP level that is with all headers down to layer 3.
    - **Marker**: This parameter denotes whether the marker is SET (SET=1, UNSET=0). A marker indicates various phenomena such as end of silence period and end of video frame, and is added by the application.
    - **Status**: This parameter lets you check whether the status is **OK**.
5. From the **Summary Information** window, we can see a maximum Jitter of

2.89 ms; this is very low, so we should not expect any problems here.

6. Clicking on the **Graph** button in the middle of the lower part of the **RTP Stream Analysis** window will open the following IO Graph:



7. In this case, we see that the network looks great—the Jitter is less than **2.500 ms**.
8. When we use Wireshark and it looks like all the parameters measured in the previous points are OK, the problem in connectivity is a probably configuration problem in the equipment itself.
9. You might see some problems such as wrong sequences and timestamps (see the following screenshot). These errors usually occur due to Jitter and delay problems.



How it works...

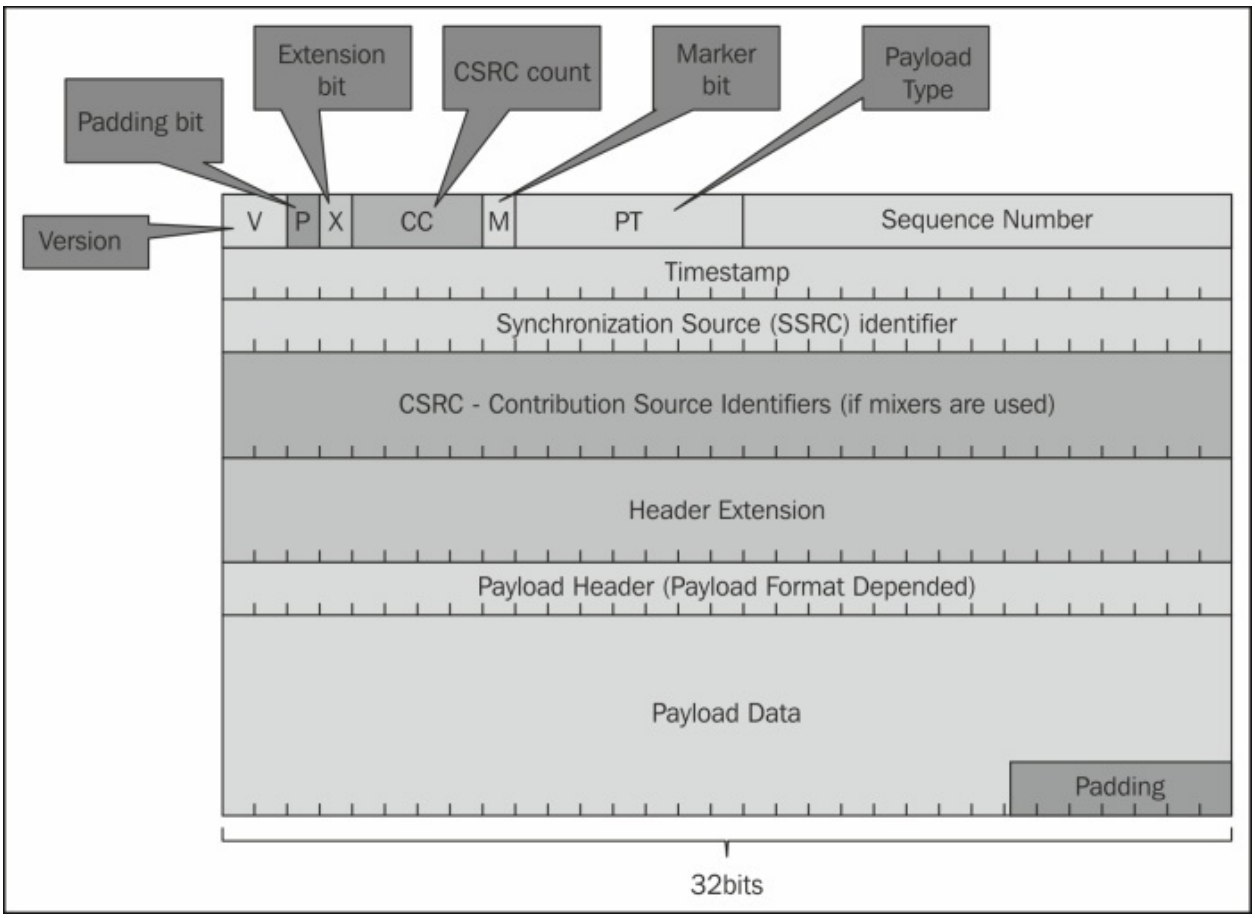
RTP is used in conjunction with RTCP (both were first standardized in RFC 3550). RTP is used to carry the media streams (audio and video), and RTCP is used to monitor transmission statistics and quality of service. While establishing a session, RTP uses even port numbers, whereas RTCP uses the next corresponding odd port number (higher by one).

RTP provides mechanisms for timing recovery, loss detection and correction, payload and source identification, and media synchronization.

RTCP specifies reports that are exchanged between the source and destination of the session. Reports contain statistics such as the number of RTP-PDUs sent, the number of RTP-PDUs lost, inter-arrival Jitter, and so on. These reports can be used by applications to modify the sender's transmission rates and for diagnostic purposes.

RTP principles of operation

RTP lies over UDP, which lies over IP. In the following diagram, you see the RTP packet structure:

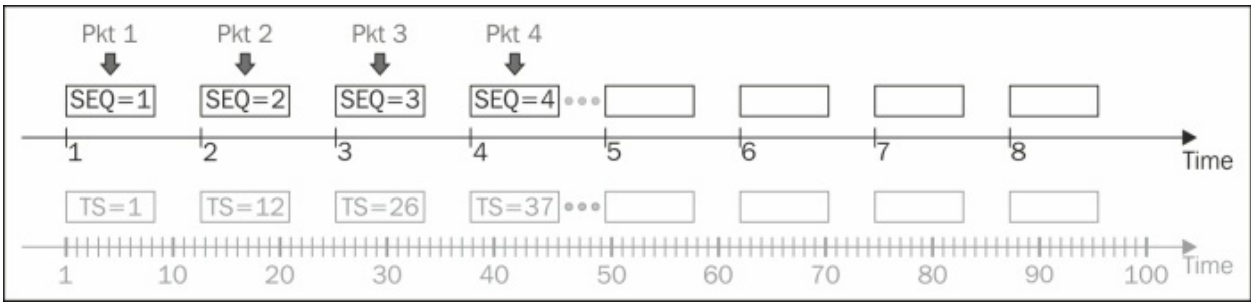


The fields in the header are as follows:

- **Version (V)**: This field indicates the RTP version
- **Padding (P)**: This field indicates that the packet contains one or more additional padding bytes at the end that are not part of the payload
- **Extension bit (X)**: This field indicates a fixed header that follows the standard header
- **CSRC count (CC)**: This field contains the number of CSRC fields that follow the fixed header
- **Marker (M)**: This field is used to indicate application events, for example video frame boundaries
- **Payload type**: This field identifies the format of the RTP payload to be interpreted by the receiving application
- **Sequence number**: This field is incremented by one for each RTP packet sent. Used by the receiver to detect packet losses

- **Timestamp**: This field reflects the sampling rate of octets in the RTP data stream
- **Synchronization source (SSRC)**: This field is the stream identifier that is chosen randomly, so that no two synchronization sources within the same RTP session will have the same SSRC identifier
- **Contributing source identifiers list (CSRC)**: This field identifies the contributing sources (that is, the stream source) for the payload contained in this packet

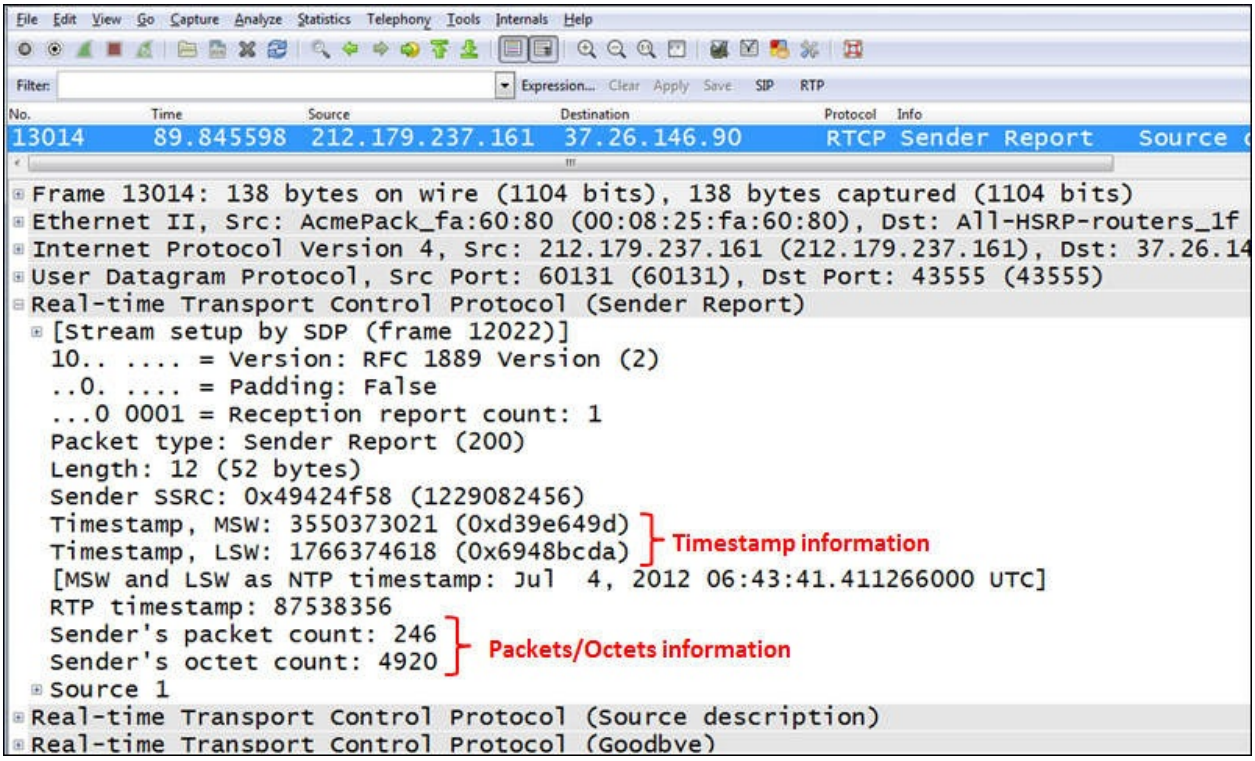
In the following diagram, you can see how the sequence and timestamps mechanisms work:



As we can see in the diagram, the sequence numbers are increased by one for each RTP packet transmitted, while timestamps increase by the time covered by a packet. Packet number 1, for example, will have both set to 1; packet 2 will have a sequence number of 2 and a timestamp of 12; it goes on in this manner for the other packets. The receiver will receive the sequence numbers that tell him the order of the packets, and timestamps that tell him the time at which they left the receiver. The receiver will use both to play back the received data.

The RTCP principle of operation

RTCP has several report types, in which the sender and receiver update each other on the data that was sent and received. In the following diagram, you can see an example of this, in which we see a sender report that tells the receiver how many packets and octets were sent, timestamp information, and other parameters that can be used by the receiver.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save SIP RTP
No. Time Source Destination Protocol Info
13014 89.845598 212.179.237.161 37.26.146.90 RTCP Sender Report Source
Frame 13014: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits)
Ethernet II, Src: AcmePack_fa:60:80 (00:08:25:fa:60:80), Dst: All-HSRP-routers_1f
Internet Protocol Version 4, Src: 212.179.237.161 (212.179.237.161), Dst: 37.26.14
User Datagram Protocol, Src Port: 60131 (60131), Dst Port: 43555 (43555)
Real-time Transport Control Protocol (Sender Report)
[Stream setup by SDP (frame 12022)]
10.. .... = Version: RFC 1889 Version (2)
..0. .... = Padding: False
...0 0001 = Reception report count: 1
Packet type: Sender Report (200)
Length: 12 (52 bytes)
Sender SSRC: 0x49424f58 (1229082456)
Timestamp, MSW: 3550373021 (0xd39e649d)
Timestamp, LSW: 1766374618 (0x6948bcda)
Timestamp information
[MSW and LSW as NTP timestamp: Jul 4, 2012 06:43:41.411266000 UTC]
RTP timestamp: 87538356
Sender's packet count: 246
Sender's octet count: 4920
Packets/Octets information
Source 1
Real-time Transport Control Protocol (Source description)
Real-time Transport Control Protocol (Goodbye)

There's more...

Delay can come from several sources:

- **Coding delay**: This is the delay that comes from the digital processing of the voice signals.
- **Handling delay (packetization)**: This delay is the time that it takes to build packets and insert voice information into them.
- **Serialization delay**: This is the fixed delay that occurs when sending packets over the communication line. This delay depends on packet size and line speed.
- **Typical delays (round trip)**: This is the delay that you can expect when pinging over a communication line (all the following points refer to unloaded lines):
  - **Over a LAN**: The delay is less than 1 ms.
  - **Over a WAN connection**: The delay is 1-2 ms in a short-range connection (up to 250-300 km / 150-190 miles) and about 15-20 ms in long range connections (for example, US coast to coast). In older networks you can add several tens of milliseconds to these numbers.
  - **For home connections, usually xDSL or CaTV**: The delay is somewhere between 10 and 25 ms.
  - **For inter-continent connections**: The delay is somewhere between 100 and 200 ms.
  - **For cellular connections**: The delay ranges from 300 ms to 600 ms for old 2.5G networks (GPRS or CDMA 1X), 120 to 150 ms for 3.0G (UMTS or EVDO), 60 to 100 ms for HSDPA, HSUPA, and HSPA+, and goes down to 20 to 50 ms for LTE networks.
  - **For satellite communications**: The delay is 500 to 600 ms.

    ### Tip

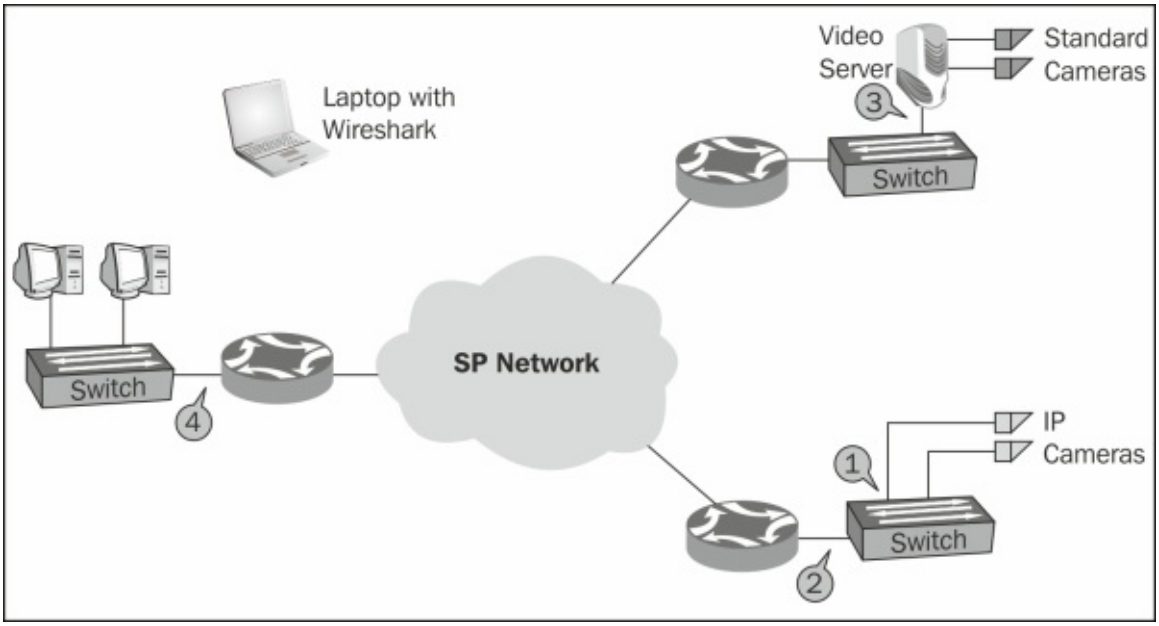
    The delay over a communication line is the sum of the time that it takes the light signal to cross the distance and time consumed by switching or routing delays on the service provider network. While technologies since the early 2000s (for example, MPLS or Carrier Ethernet) are implemented fast switches and routers, technologies older than 2000 (such as Frame Relay or ATM) have slower switching times and therefore will have higher delays.

Troubleshooting scenarios for video and surveillance applications

In the last 10 to 12 years, security and surveillance systems have taken on a larger and more important role in communications networks. The problems we might see in these types of networks will usually start from video freezes due to lack of bandwidth but can also be much more complicated as will be discussed in this recipe. In this recipe we will discuss some of the problems with these systems and how to approach and solve them.

Getting ready

Usually, you will be called to solve problems that users experience when watching security cameras. In this case, you can port mirror the specific camera (**1**), the communication line in the remote site (**2**), and a camera server (**3**), or you can monitor the central line with a filter to the remote network (**4**).



How to do it...

To identify problems in this network, follow these steps:

1. First, if possible, port mirror a connection between the viewer and a locally connected camera (over the LAN). When doing so, you will be able to note the required bandwidth for every picture resolution you try.

   Tip

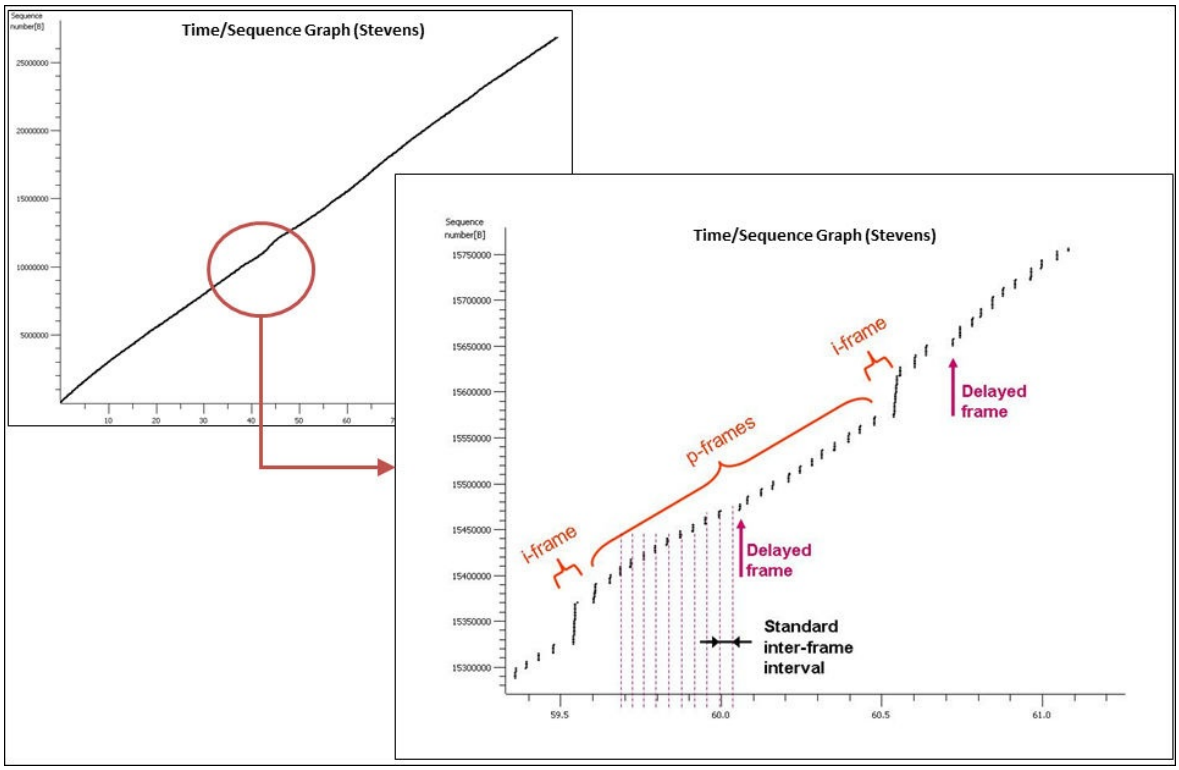
   When watching a video, the bandwidth can start at 128 Kbps for a very basic black-and-white movie at a low resolution, average around 0.5 to 1.0 mbps for a black-and-white or colored video stream at a reasonable resolution, and go up to several megabits per second for high definition streaming (usually at 6 to 8 mbps).

2. As a basic—make sure your bandwidth is sufficient.
   - When viewing freezes, use IO graphs to monitor the bandwidth. Make sure you have enough bandwidth and the line is not completely loaded.
   - To make sure you are watching only the bandwidth consumed by the camera or camera server, configure a filter to its IP address.

     Tip

     Video streaming can be transferred over UDP and RTP or over TCP. UDP is mostly used for interactive applications, while TCP is mostly used for watching remote cameras.

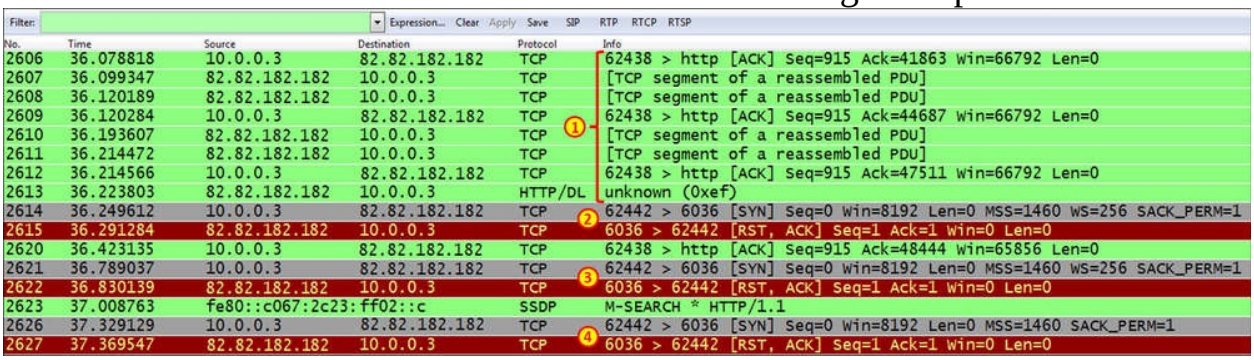
3. Make sure you don't have any packet losses or significant delays or Jitter.
   - For packet losses, log in to the communications equipment or use SNMP
   - For delay and Jitter, you can use the `ping` command or graphical utilities (many of them are free, for example, from Colasoft)
4. While monitoring a remote camera feed, if you have short freezes, navigate to **Statistics** | **TCP Stream Graph** | **Time Sequence (Stevens)**. Make sure all I and P frames are received at constant intervals.



5. The problem here is that there were cameras in the customer sites that transmitted the video to a central server and the central server transmits it to the monitors. What we see in the preceding graph is the server delayed some of the **p-frames** and that was the reason for the short freezes in that case. It turned out to be a software problem on the server.
6. When trying to log in to a camera server, several ports may be in use and you may not get the picture; or it may so happen that you get the picture but something else does not appear. To verify that all TCP port numbers are open, you can do this:
    - Look at the firewall (if there is one between you and the camera server) if connections were blocked
    - In Wireshark, make sure you don't get any **triple SYN**, which indicates that something is blocking your access to the server

- In the following screenshot, you see how the HTTP session (**1**) is running between the internal office address **10.0.0.3** and the external address of the web camera **82.82.182.182** (don't use these; they are just sample values). In the line **2614**, you see a **SYN** packet is sent from **10.0.0.3** to **82.82.182.182**; this packet is blocked in packet **2615** via the **TCP RST** (reset) PDU (**2**). The same event

occurs twice more (**3** and **4**). The fact that you see one established connection does not mean that there are no other connections being attempted.



No.	Time	Source	Destination	Protocol	Info
2606	36.078818	10.0.0.3	82.82.182.182	TCP	62438 > http [ACK] Seq=915 Ack=41863 Win=66792 Len=0
2607	36.099347	82.82.182.182	10.0.0.3	TCP	[TCP segment of a reassembled PDU]
2608	36.120189	82.82.182.182	10.0.0.3	TCP	[TCP segment of a reassembled PDU]
2609	36.120284	10.0.0.3	82.82.182.182	TCP	62438 > http [ACK] Seq=915 Ack=44687 Win=66792 Len=0
2610	36.193607	82.82.182.182	10.0.0.3	TCP	[TCP segment of a reassembled PDU]
2611	36.214472	82.82.182.182	10.0.0.3	TCP	[TCP segment of a reassembled PDU]
2612	36.214566	10.0.0.3	82.82.182.182	TCP	62438 > http [ACK] Seq=915 Ack=47511 Win=66792 Len=0
2613	36.223803	82.82.182.182	10.0.0.3	HTTP/DL	unknown (0xef)
2614	36.249612	10.0.0.3	82.82.182.182	TCP	62442 > 6036 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
2615	36.291284	82.82.182.182	10.0.0.3	TCP	6036 > 62442 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2620	36.423135	10.0.0.3	82.82.182.182	TCP	62438 > http [ACK] Seq=915 Ack=48444 Win=65856 Len=0
2621	36.789037	10.0.0.3	82.82.182.182	TCP	62442 > 6036 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
2622	36.830139	82.82.182.182	10.0.0.3	TCP	6036 > 62442 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
2623	37.008763	fe80::c067:2c23:	ff02::c	SSDP	M-SEARCH * HTTP/1.1
2626	37.329129	10.0.0.3	82.82.182.182	TCP	62442 > 6036 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
2627	37.369547	82.82.182.182	10.0.0.3	TCP	6036 > 62442 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

- In this case, the HTTP connection had connectivity to the web server. To log in, another connection was opened. Since the log in connection was blocked, it was possible to see the camera server but not to log in to it and watch the video.

How it works...

Video streams are made of **I-frames** (Intra-coded frames), **P-frames** (Predicted frames), and **B-frames** (Bi-predictive frames). I-frames are frames that contain the full picture, while P-frames contain changes from the previous one. There are also B-frames, which also use prediction mechanism for the next frame.

In TCP, each video frame, I, P, or B, is divided between several TCP packets; therefore, when you have TCP problems (retransmissions and others) it can directly influence the video stream.

There's more...

The quality of video transmission depends on the codec that you are using, number of frames per second that are transmitted, time interval between frames, and more parameters that can be configured in the camera or on the camera server. Make sure you've set all parameters correctly to get a good picture.

Troubleshooting scenarios for IPTV applications

IPTV applications have become more and more popular over the last few years, while more and more TV stations are moving to the Internet. E-learning applications are also more popular along with various types of other applications.

Basically, IPTV applications use TCP, and the problems you will face are mostly TCP problems, such as retransmissions. In this recipe we will see some examples.

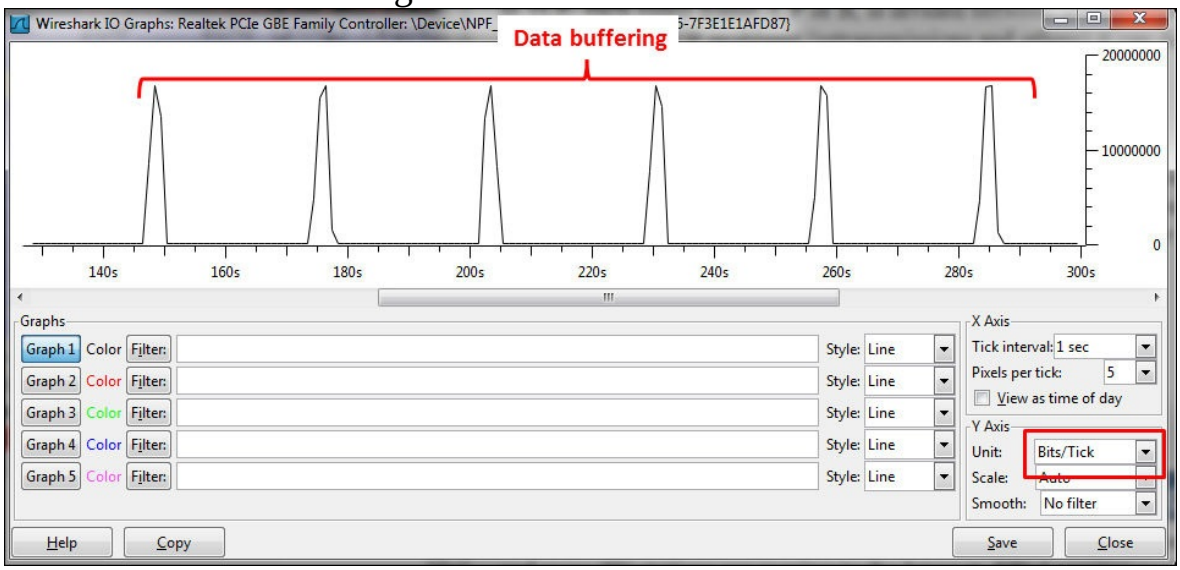
Getting ready

When getting complaints about quality of video, freezes, and so on, connect the instance of Wireshark that has a port mirror to the device or the link that connects you to the network.

How to do it...

Start the capture and go through these steps:

1. Open the IO graph and verify that you have buffering type of traffic, as illustrated in the following screenshot:



- If you see that the line is blocked at the top, check your bandwidth to the Internet and tune the viewer accordingly (it's usually done automatically).
- Check for TCP retransmissions, duplicate ACKs, and TCP window problems, and if you find any, go through it to find what is disturbing the transmission.

How it works...

From the network point of view, IPTV is not more than a simple application that runs over a TCP connection.

There's more...

When troubleshooting problems on it, go through the regular TCP troubleshooting procedures described in Chapter 9, *UDP/TCP Analysis*.

Troubleshooting scenarios for video conferencing applications

Video conferencing uses the same protocols as standard telephony, but there is a difference: while in telephony we have one stream of data in each direction, we have a stream of data and a stream of video in video conferencing. When you capture data on the end device, you will see four streams of data: two streams that you send to the other side and two streams of data that are sent back to you.

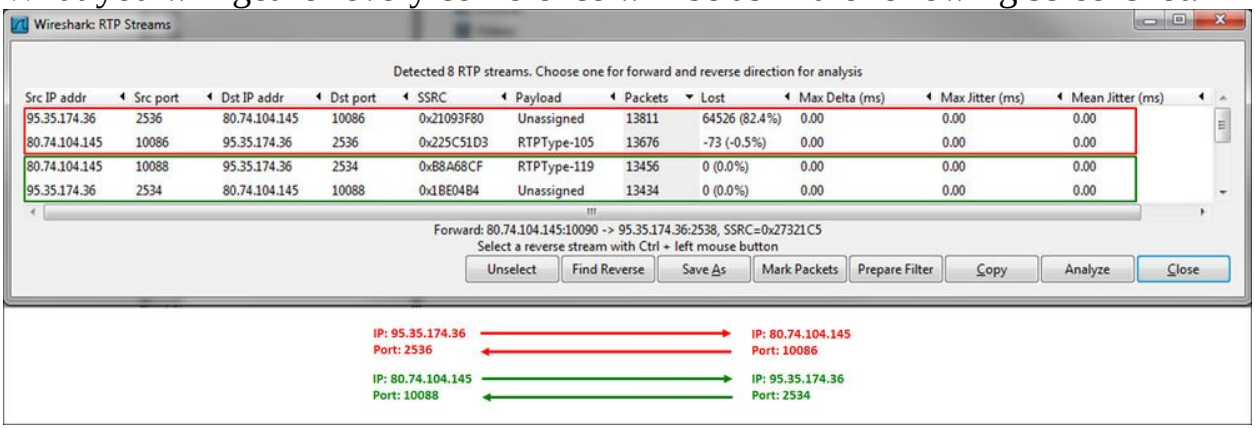
Another difference is that some video conference applications are still using the H.323 protocol suite, so instead of troubleshooting SIP problems, you will have to troubleshoot H.225 and H.245 connectivity issues. Due to the fact that most applications use SIP and the IETF protocol stack, we will focus on them only.

Getting ready

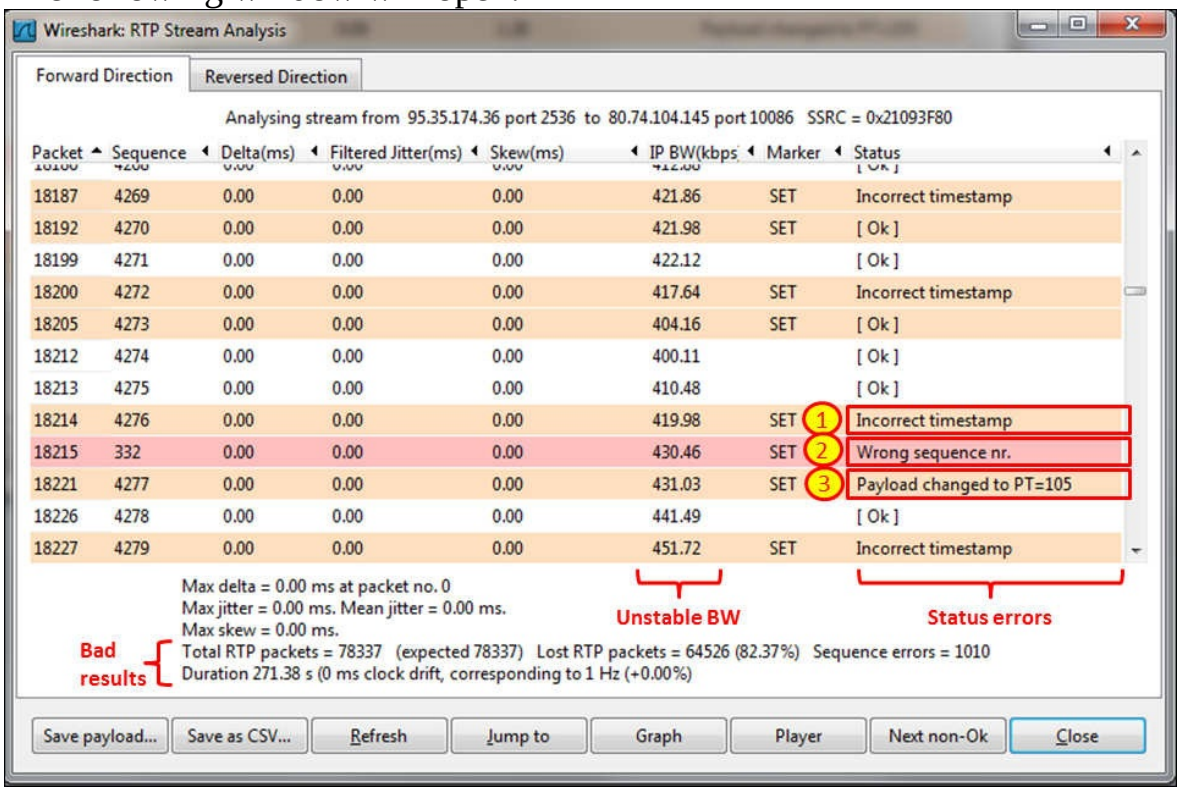
To troubleshoot a problem in your video conference system, connect the instance of Wireshark with port mirror to the device or to the link to the devices that are functioning badly.

How to do it...

What you will get for every conference will be as in the following screenshot:



1. As you can see in the screenshot, there are two streams of data in each direction.
2. On one of them (the first one), we see massive degradation in performance.
3. To focus on it, click on the stream, and then click on **Analyze**.
4. The following window will open:



- In the preceding screenshot, you see that there are many errors, the bandwidth is unstable, and there are many error statuses:
  - Incorrect timestamp (**1**) and wrong sequence number (**2**) are caused by a communication line with high Jitter
  - Payload change (**3**) occurs when the system on the sender's side changes the codec to a better one to fit into the channel
- There was a problem here simply because this was a video conference call over an unstable cellular connection.

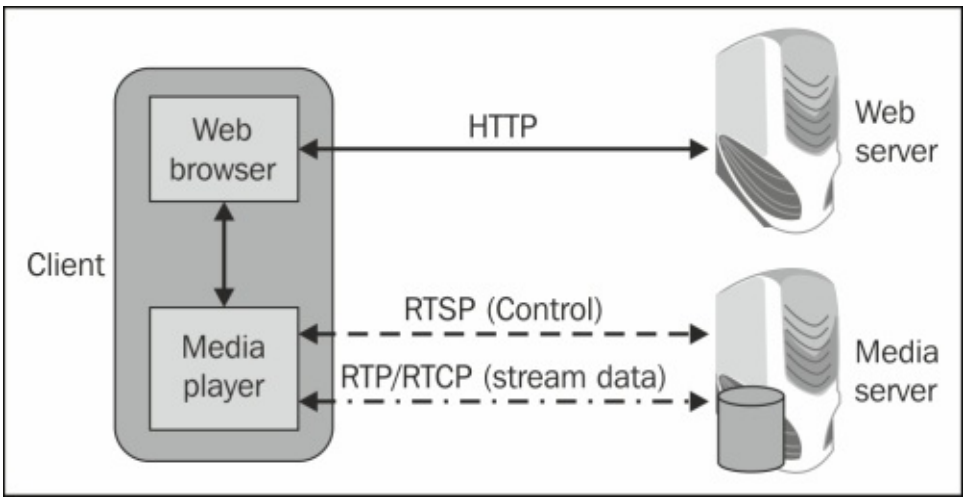
Troubleshooting RTSP

RTSP is an application-layer control protocol that is used for the control of a single or multiple time-synchronized streams of continuous media such as audio and video. The purpose of RTSP is to provide control over remote media servers. It is used when we click on **Play**, **Pause**, and so on, and can be used also to invite a new media server for viewing on the screen, for example, for a conference. While RTSP is the control protocol, the streaming itself is usually carried out by RTP—which carries the data—and RTCP —used for the monitoring of the data transfer.

The RTSP standard (RFC2326) does not define any transport protocol, but most implementations use TCP. RTSP is commonly used while watching IPTV. In this recipe we will learn how to monitor and troubleshoot these streams.

Getting ready

RTSP monitoring should be used in cases in which you experience transmission disturbances; for example, problems with the media player control or cases with connectivity problems to a server. RTSP works as illustrated in the following diagram:



When monitoring a stream, we can have problems with RTP/RTCP (discussed earlier in the chapter), HTTP (discussed in Chapter 10, *HTTP and DNS*), or even TCP problems (discussed in Chapter 9, *UDP/TCP Analysis*). In this recipe we will talk about RTSP (the center line with long dashes in the preceding diagram).

The web server and the media servers can be on single or multiple physical servers, or on different virtual machines. The functionality in any case is as presented.

How to do it...

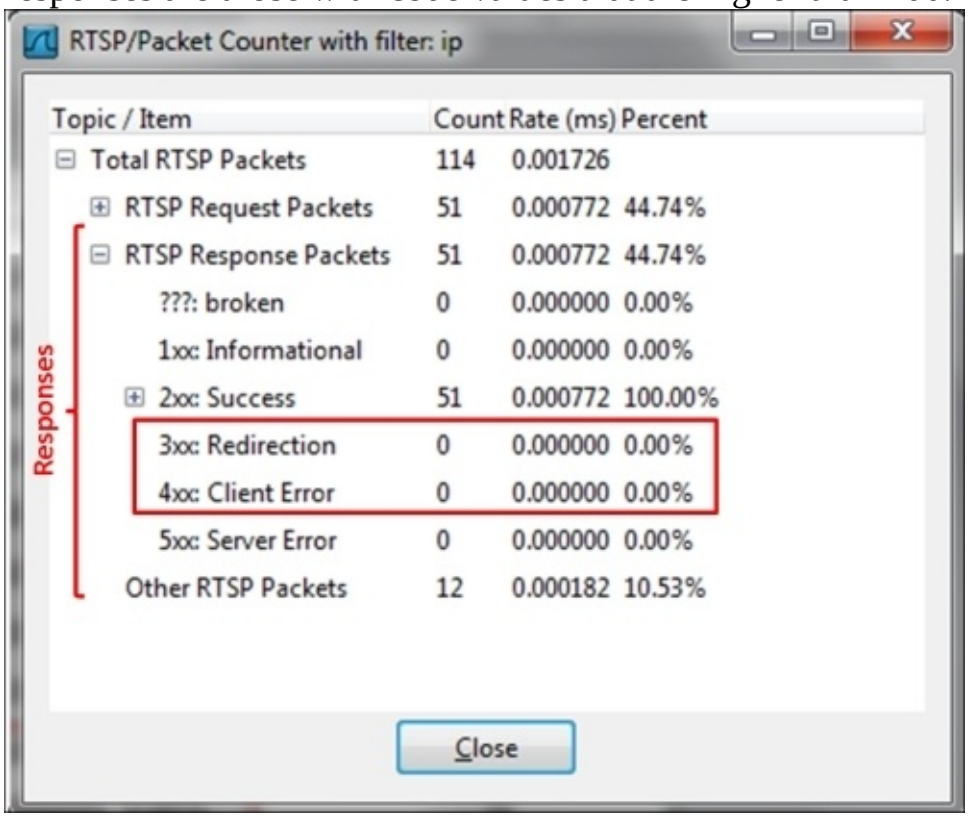
To find problems with RTSP, connect the instance of Wireshark with port mirror to the client experiencing the problems, and in the case of multiple clients, connect it to a mutual link or to the server.

1. To view all RTSP traffic, filter the packets with TCP port 554; the filter for this is `tcp.port==554`.

   Tip

   The filter `tcp.port == 554` gives us all traffic over this port, while filter `rtsp` only gives us packets in which Wireshark is recognized as an RTSP header.

2. To view RTSP requests and responses, navigate to **RTSP** | **Packet Counter** from the **Statistics** menu as described earlier in this chapter. Error responses are those with code values that are higher than 400.



3. Look for RTSP response codes that are 4xx or higher. To do so, you can configure the display filter `rtsp.status >= 400`.

How it works...

As with SIP (which is used for signaling, while RTP is used for the transport of the media), the streams controlled by RTSP may use any transport protocol; in many cases, they also use RTP. The protocol is intentionally similar in syntax and operation to HTTP and uses the same syntax.

The most common RTSP methods (commands) are (C-Client, S-Server):

Command	Direction	Function
OPTIONS	C to S or S to C	Determines capabilities of server/client
DESCRIBE	C to S	Gets description of media stream
ANNOUNCE	C to S or S to C	Announces a new session's description
SETUP	C to S	Creates a media session
PLAY	C to S	Starts media delivery
RECORD	C to S	Starts media recording
PAUSE	C to S	Pauses media delivery
REDIRECT	S to C	Redirects to another server
TEARDOWN	S to C	Performs immediate teardown

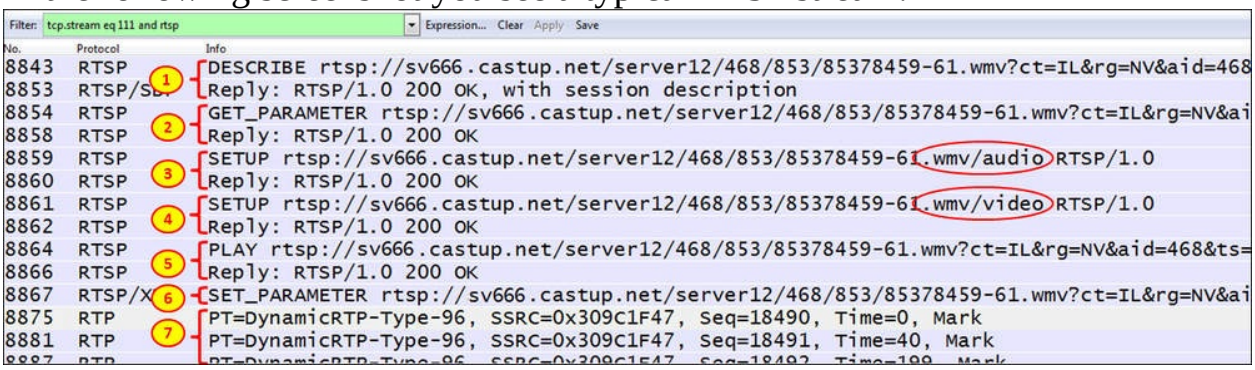
The response categories are:

Code series	Type	Meaning
1xx	Informational	Request received, continue with processing
2xx	Success	The action was successfully received, understood, and accepted

3xx	Redirection	Further action must be taken in order to complete the request
4xx	Client error	The request contains bad syntax or cannot be fulfilled
5xx	Server error	The server failed to fulfill an apparently valid request

There's more...

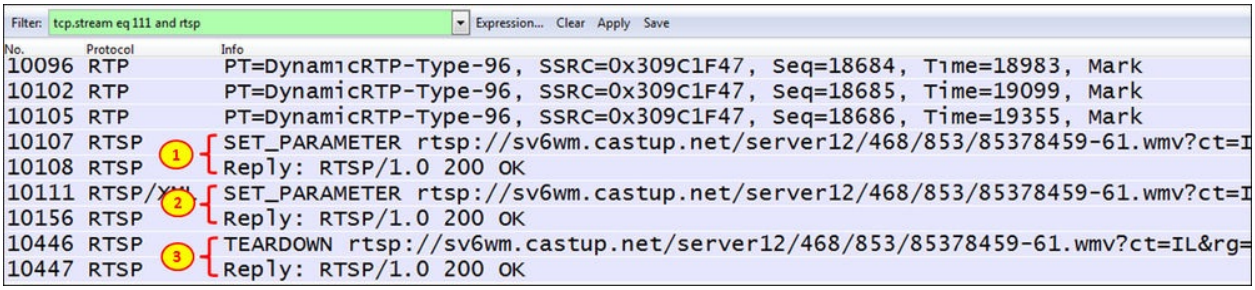
In the following screenshot you see a typical RTSP stream:



The typical RTSP stream is processed in the following order:

1. A **DESCRIBE** request is sent to the server, asking to retrieve the description of a presentation or media object identified by the request URL from that server, and the server replies with **200 OK**.
2. A **GET_PARAMETER** request retrieves the parameter value of a presentation or stream specified in the URI.
3. A **SETUP** request is sent to open the audio stream and is confirmed with **200 OK**.
4. A **SETUP** request is sent to open the audio stream and is confirmed with **200 OK**.
5. A **PLAY** request is sent to the server to start playing the stream.
6. A **SET_PARAMETER** request is sent to the server to set a parameter value for a presentation or stream specified by the URI.
7. The stream starts to play with RTP.

In the following screenshot, we see how the stream is broken down:



The process for the breakdown of the stream is as follows:

1. **SET_PARAMETER** is sent to the server to set a parameter value for a presentation or stream specified by the URI.
2. A second **SET_PARAMETER** request is sent to the server.
3. The **TEARDOWN** command is sent to close the connection.

Chapter 13. Troubleshooting Bandwidth and Delay Problems

In this chapter we have the following recipes:

- Measuring total bandwidth on a communication link
- Measuring bandwidth and throughput per user and per application over a network connection
- Monitoring jitter and delay using Wireshark
- Discovering delay/jitter-related application problems

Introduction

When measuring communication lines, there are four major parameters that we should be aware of: **bandwidth**, **delay**, **jitter**, and **packet loss**. While there are applications that require high bandwidth, there are other applications that are more sensitive to delay and jitter. Packet loss can influence all types of applications, but there are applications that are more sensitive to it and some that are less.

In this chapter we will learn how to measure these parameters, how to check for network problems caused by it, and how to solve them when possible.

Measuring total bandwidth on a communication link

In this recipe, we will see how to measure the total bandwidth over a communication line. The first thing of course is to verify the communication line with the service provider. Check whether it is a symmetric or an asymmetric line, and if it is asymmetric, check what the bandwidth is in both directions.

Getting ready

There are two cases that you might need to test:

- When you measure a communication line between two offices: in this case connect your laptop (or any PC on the network) to the LAN, and verify whether you have a server or another PC on the other side of the line
- When you measure a communication line to the Internet, make sure you have a testing server on the **Service Provider** (**SP**) side or on the **Internet Service Provider** (**ISP**) side

How to do it...

To check the bandwidth on a communication line, follow these steps:

1. Ask for the following details:
    1. Ask the SP what the line bandwidth is.
    2. If it is a line to the Internet, in addition to the preceding step ask the ISP what is the bandwidth to the Internet.

- Locate a server, a PC, or a laptop on the remote location.

Tip

When using a PC or laptop for the test, don't forget that the PC itself should be strong enough to generate the traffic. A standard Windows 7 is able to generate around 200 Mbps per TCP connection, and when opening several connections, you can get into other limitations such as disk performance and so on. Therefore, it is recommended to try the transfer first on a LAN, where there are no bandwidth limits (practically), and only then to test the SP or the ISP lines. If you are using FTP, use an efficient one (FileZilla, for example). The best way of course is to use test equipment, if it's available. Dedicated test equipments are available from many vendors such as VeEX, Fluke Networks, and IXIA.

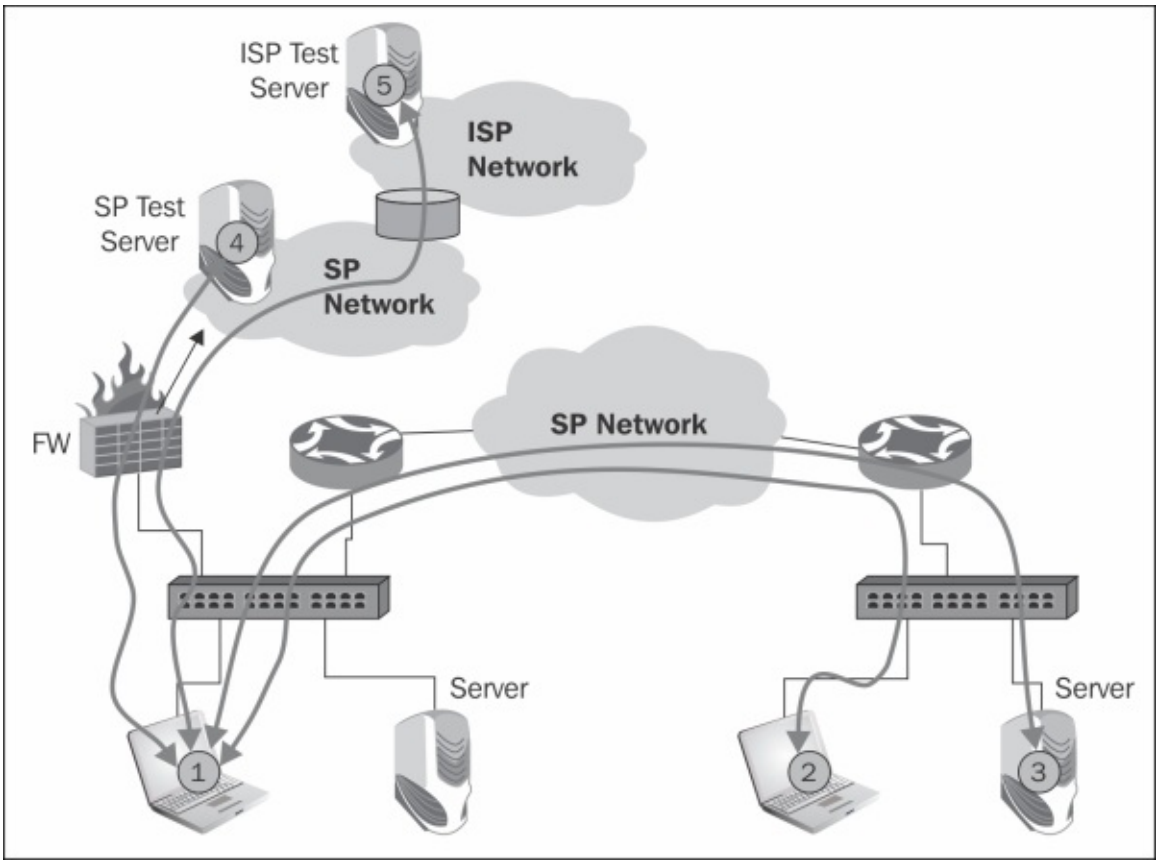
- In case you want to test the bandwidth between two sites, download and then upload a big file between nodes numbered as **1** and **2** or between nodes numbered as **1** and **3**. A file big enough should load the line for a significant amount of time, that is, a minute or more. For example, if you want to test a 10 Mbps (Megabits per second) line, use a file of at least 10/8 = 1.25 MB (Megabytes).
- In case you want to test your connection to the Internet, usually you can perform the test on your service provider (numbered as **1** to **4** in the following diagram), and then to your Internet service provider (numbered as **1** to **5** in the following diagram).

    ### Tip

    If possible, it is better to use the IP or UDP test, since when you copy a file, it is done over TCP, so you can get into TCP issues that influence the test. For this purpose, use **Iperf** or another testing tool that can generate IP or

UDP traffic.

In the following illustration, you can see two local networks connected via a **Service Provider** (**SP**) line. The site on the left is connected to the Internet through a firewall. The connection to the Internet goes through the Service Provider (**SP**, **Server 4**) to the **Internet Service Provider** (**ISP**, **Server 5**).



Follow these steps to measure the bandwidth over the communication lines:

1. Use Wireshark **Statistics** | **IO Graphs** for the test.

Tip

Don't forget that Wireshark has its own limitations when working with high bandwidth lines. In this case, you can configure it to use multiples files. Personally, I prefer to use other tools (**Omnipeek**, for example) when

monitoring lines of 200-300 Mbps and higher.

2. When testing your enterprise network, you can use software tools such as Iperf (http://sourceforge.net/projects/iperf/).

Following are the steps to measure network bandwidth with IPerf:

1. Install Iperf on both ends of the connection.
2. Configure one side as a client, and the other side as a server.
3. Start the test and use I/O Graphs to verify that you have a stable bandwidth.

Tip

When downloading or uploading a file, do it with a single large file and not a directory of multiple files. When transferring many small-sized files, it will take time to open and transfer each one of them, so the test will not give good results.

When getting less bandwidth than expected, perform the following steps:

1. When getting a value up to around 5% more or less than expected, it can be due to the reasons mentioned in the *There's more...* section in this recipe. Check the configurations and the technology that the line is running on (**SDH/SONet**, **Carrier Ethernet**, and so on)
2. If you test the line with file copy, and in the IO graphs see sawtooth, there might be errors on the line. Check TCP retransmissions, and then check for errors in the switch/router port connected to the service provider.

Tip

To check switch or router port statistics, you can use console or telnet to connect to it and use the switch or router commands (for example, **show interface** commands in Cisco). You can also use SNMP management software or any MIB browser and browse the `IfInErrors` and `InOutErrors` objects.

3. If you see a degradation of 80 to 90 percent of what you had expected (for example, you test a line of 100 Mbps and get 10 to 20 Mbps); in most of the cases, it is a **duplex-mismatch** problem. As shown in the *How it works...*

section of this recipe.

It isn't common, but it can also be that your service provider has a configuration problem. Check it with them. If none of the preceding cases are true, it can be that this is the reason.

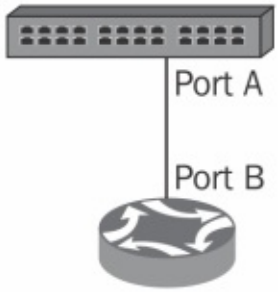
How it works...

First, there are two different definitions; it is important to distinguish between:

- **Bandwidth**: This is the total bits per second that can be transferred over a communications line
- **Throughput**: This is the effective application bytes per second that is transferred between the two ends of a connection

To check the bandwidth of a communication line, you can ask the service provider for the line details, or you can simply transfer some traffic over it, use Wireshark or SNMP tool, and see what you get.

Most of the cases in which a duplex mismatch problem occurs is when you connect using Ethernet on one side with 100 Mbps full duplex, and the other side configured to auto-negotiate.



No.	Setup		What You Get		Status
	Port A	Port B	Port A	Port B	
1	100FD	100FD	100FD	100FD	V
2	100FD	AUTO	100FD	100HD	X
3	AUTO	100FD	100HD	100FD	X
4	AUTO	AUTO	1000FD	1000FD	V

V: OK, X: Mismatch

As you see in the diagram, when you connect a device (a router in this example) to a switch, when both sides are manually configured, for example, to 100 Mbps **Full Duplex** (**FDX**), the intended configuration will take place (numbered **1** in the preceding diagram).

When you configure both sides to auto-negotiation (numbered **4** in the preceding diagram), it will also be fine, and will be automatically set to 1 Gbps (in the case of gigabit adapters).

In the case when one side is configured to 100 FDX and the other side to auto negotiate, the auto negotiate will be automatically set to 100 Mbps **Half-Duplex**

(**HDX**). In this case, when one side is set to HD and the other to FD, many packets will be lost, and you will experience significant degradation in performance (numbered **2** and **3** in the preceding diagram).

There's more...

When we buy a line at a certain bandwidth, it can be that we'll get a little bit more or less of what we've bought. For example, when we buy 10 Mbps line, and the line runs over the **Synchronous Digital Hierarchy** (**SDH**) or **Synchronous Optical Network** (**SONet**) line; the 10 Mbps is made of 5 VC-12s, which is 5*2.176 Mbps, so the total bandwidth will be 10.88 Mbps.

On the other hand if, for example, we use site-to-site VPN over the Internet, and the line is 10 Mbps, even if we have a very good Internet connection (for example, when the two ends are connected to the same ISP), the encryption mechanisms of the VPN itself can take 5 to 10 percent of the line, and when measuring it, you will get somewhere between 9.0 to 9.5 Mbps. In this case, for example, when you transfer a file over the line, you will see that the line is loaded with 10 Mbps (that is, the bandwidth), while what is left for the file copy is usually between 9.0 to 9.5 Mbps (that is, the throughput).

Measuring bandwidth and throughput per user and per application over a network connection

In many cases, we need to know not only the total bandwidth of a connection, (communication line or on a server port), but also who exactly are the consumers, that is from which IP addresses and port numbers the traffic is coming. In this recipe, we will see how to measure it.

In order to see this, you can use proprietary tools that collect the data from the switch (**RMON1**, **RMON2**, **sFlow**) or router (**Cisco Netflow** or **Juniper Jflow**), or to use Wireshark with port mirror to the communication link, and this is what we'll learn in this recipe.

Getting ready

For using Wireshark to get traffic distribution, connect a laptop with a port mirror to the link you wish to monitor and start packet capture. You can also use the `Tshark` command from the CLI.

How to do it...

For basic statistics on users and applications that are using the communications link, perform the following steps:

- For general statistics:
    1. From the **Statistics** menu, choose **Conversations**.
    2. In the **Conversations** window, you see the statistics on the total number of packets captured until now.
    3. You can also use graphical tools such as **Compass** (Chapter 11, *Analyzing Enterprise Applications, Behavior*).

- For flow analysis, use IO graphs with filters on IP addresses and/or port numbers:
    1. From the **Statistics** menu, select **IO Graphs**.
    2. In the **IO graphs** window (Chapter 5, *Using Advanced Statistics Tools*), configure IP and port numbers and display filters for the applications that you wish to monitor.

- For continuous monitoring, use Wireshark with multiple files with ring buffer, or use tools such as Netflow or Jflow for router monitoring.

How it works...

With Wireshark, like we learned in Chapter 1, *Introducing Wireshark*, we capture data and analyze it.

In Netflow, Jflow, and applications that collect data from the router, the router periodically sends the collected data to the management console that analyzes it.

In **Remote Monitoring 1** (**RMON1**) and **Remote Monitoring 2** (**RMON2**), when the end switch supports it, you access the data with the SNMP software that reads from the RMON1/RMON2 MIB. While RMON1 provides you layer 1 to 2 statistics, RMON2, when implemented provides you layer 3 to 4 statistics. The main standards of RMON were published in RFCs 2613, 2819, 3577, and 4502. In various applications and devices such as firewalls, **Intrusion Detection Systems** (**IDS**), **Deep Packet Inspection** (**DPI**) devices, and WAN Accelerators, you will get the data from the monitored device.

See also

Additional data on these applications can be found at:

Cisco Netflow:
http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_hon

http://www.ietf.org/rfc/rfc3954.txt

For Juniper Jflow:

http://www.juniper.net/techpubs/software/erx/junose82/swconfig-ip-services/html/ip-jflow-stats-config2.html

sFlow:

http://www.ietf.org/rfc/rfc3176.txt

Various applications can be located in:

For switch monitoring:

http://www.sflow.org/index.php

http://tools.ietf.org/html/rfc3176

Monitoring jitter and delay using Wireshark

Jitter and delay are characteristics that can significantly influence various network applications. For monitoring jitter and delay on a communication line, you can use simple or graphical Ping tools that will show you the line characteristics. Wireshark on the other hand does not measure the end-to-end delay but the influence that it has on the network traffic, that is inter-frame delay and how it influences applications.

In this recipe, we will see how to use Wireshark tools for monitoring these parameters, and in the next recipe we will see how to discover problems caused by them.

Getting ready

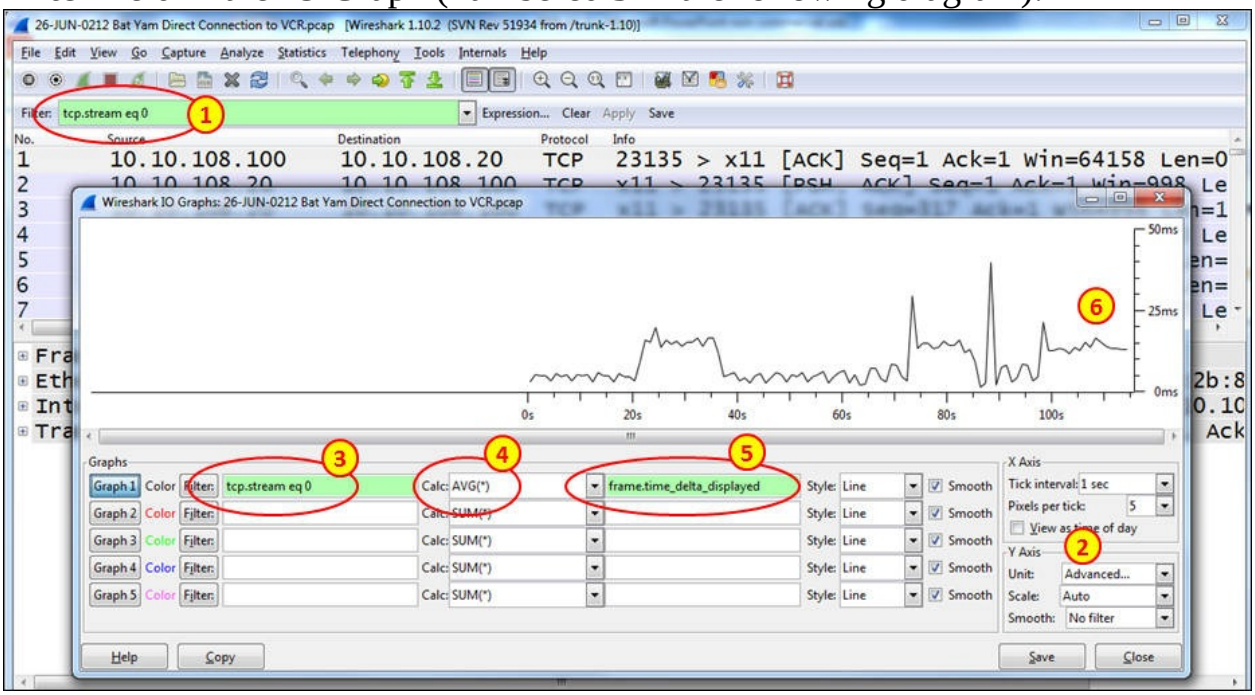
For monitoring delay on a communication line, first use the `ping` command to get the feeling of the line, and then configure port mirror to the port you want to monitor.

How to do it...

To monitor inter-frame delay:

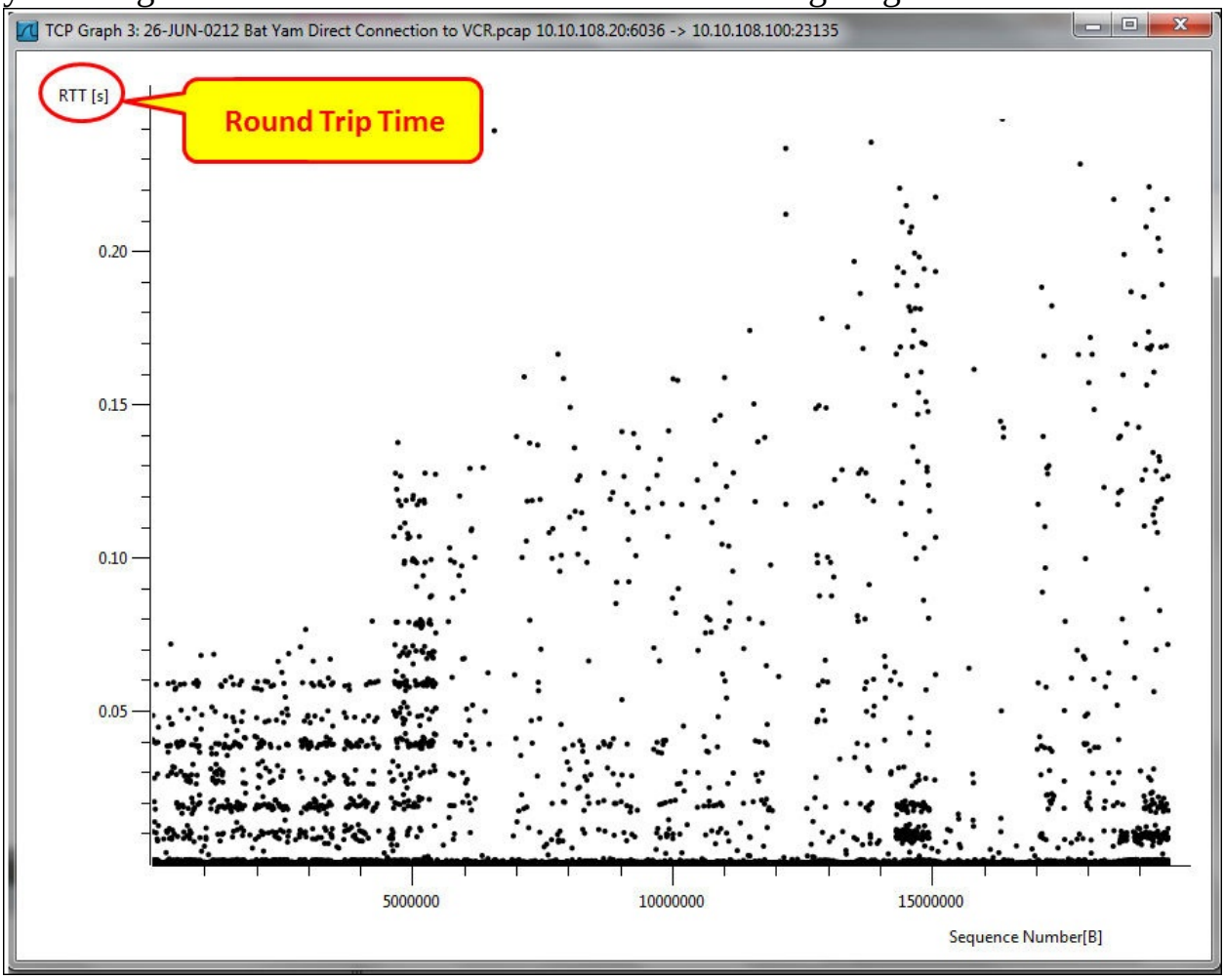
1. From **Statistics**, select **IO Graph**.
2. For monitoring time between frames in a specific stream of data:
    1. Click on a packet in the TCP or UDP stream.
    2. Click on **Follow TCP Stream** or **Follow UDP stream**.
    3. Copy the displayed filter string that showed up (numbered **1** in the next screenshot).

- From statistics open **IO Graph**.
- In **IO Graph**, in the **Y Axis** part (bottom-right side of the window), select **Advanced...** (numbered **2** in the following diagram).
- Copy the **TCP stream number** (numbered **1** in the following diagram) to the **Filter** field in the IO Graph (numbered **3** in the following diagram).



- Select **AVG(*)** (numbered **4** in the preceding diagram).
- Configure the filter `frame.time_delta_displayed` (numbered **5** in the preceding diagram).
- In the graph (numbered **6** in the preceding diagram), you see the time between frames in milliseconds.

- By navigating to **Statistics** | **TCP Stream Graph** | **Round Trip Time Graph**, you will get the same results as shown in the following diagram:



- In the diagram, we see that the **Round Trip Time** (**RTT**) varies between values that are lower than 10 ms and up to 200-300 ms.
- To measure delays in layer 4, use the TCP filter `tcp.analysis.ack_rtt` that will give you the time that it takes to acknowledge every received packet.

How it works...

The software simply captures packets over the line, and shows you the time difference between them. It is important to notice that there is a delay or jitter, but we will not see where it is coming from.

Delay is the time that it takes a packet to get from one end of the network to the other. It is usually referred to as RTT. Delay can be measured with simple Ping or graphical Ping tools. Delay is measured in seconds – milliseconds (ms), microseconds (µs), and so on.

Jitter in IP networks measure the variations in delay. For example, if we have an average delay of 100 ms, and it varies between 80 ms and 120 ms, the jitter is 20 percent.

There's more...

Graphical Ping tools are available for free on many websites. You can use, for example, http://www.colasoft.com/download/products/download_ping_tool.php.

Discovering delay/jitter-related application problems

Jitter and delay can influence various types of applications. In applications that run over TCP, high delay reduces the effective throughput that can be sent and high jitter can cause packet losses and retransmissions. In multimedia applications that run over RTP, which runs over UDP, high jitter and delay can cause severe disturbances in the voice and video quality.

In this recipe, we will get into the details: the influence of behavior on TCP, and how it can influence the application behavior. RTP over UDP behavior was discussed in Chapter 12, *SIP, Multimedia, and IP Telephony*.

Getting ready

When you ping a remote site and get high delays, and in the Wireshark you see many retransmissions, it can be because of high network or applications delay. Connect the Wireshark to the network and configure port mirror to the link that you test.

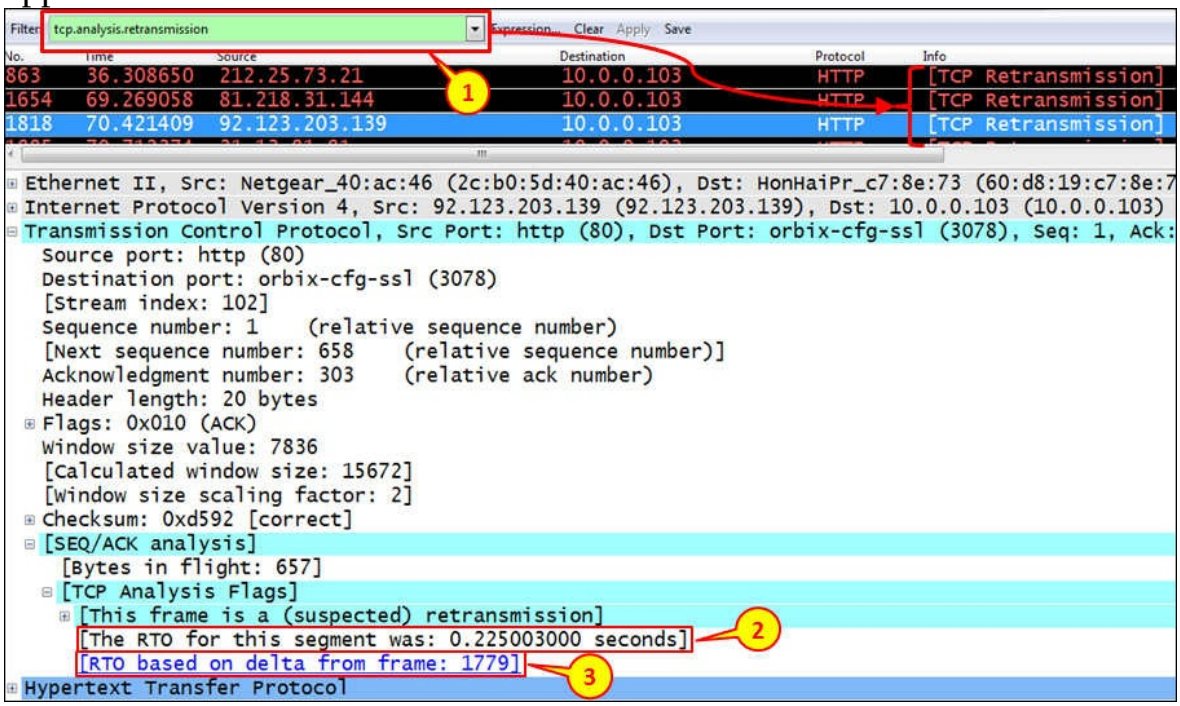
The purpose of this recipe is to check whether the TCP retransmissions and duplicate ACKs are due to delay and jitter or other problems.

How to do it...

When experiencing many TCP retransmissions, perform the following tests:

1. Check whether retransmissions are coming from the same application or from the same IP address. In this case, it is a slow application or a slow device and probably not a network delay issue.

   If retransmissions are distributed between various applications and devices, it can be because of unstable line that causes network delays.
2. Configure a display filter `tcp.analysis.retransmissions` (numbered **1** in the following diagram). A list of all retransmissions in the packet list will appear.

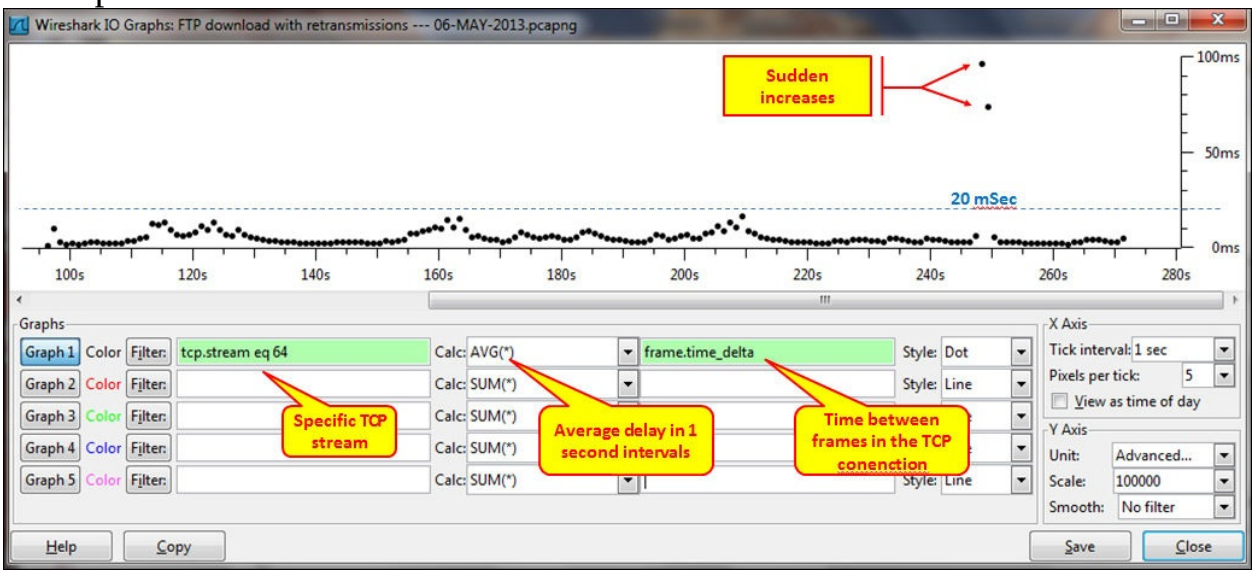


3. Down the packet details pane, expand the **TCP Analysis Flags**, and you will get:
   - The time since the original packet is retransmitted (numbered **2** in the preceding diagram). In this case, **0.225003000** seconds.
   - The packet that is retransmitted (numbered **3** in the preceding diagram). In this case, packet number **1779**.

- Usually the **Retransmission Time Out** (**RTO**) timer will be around 0.2 seconds for local connections, and up to 0.3 to 0.4 seconds for international

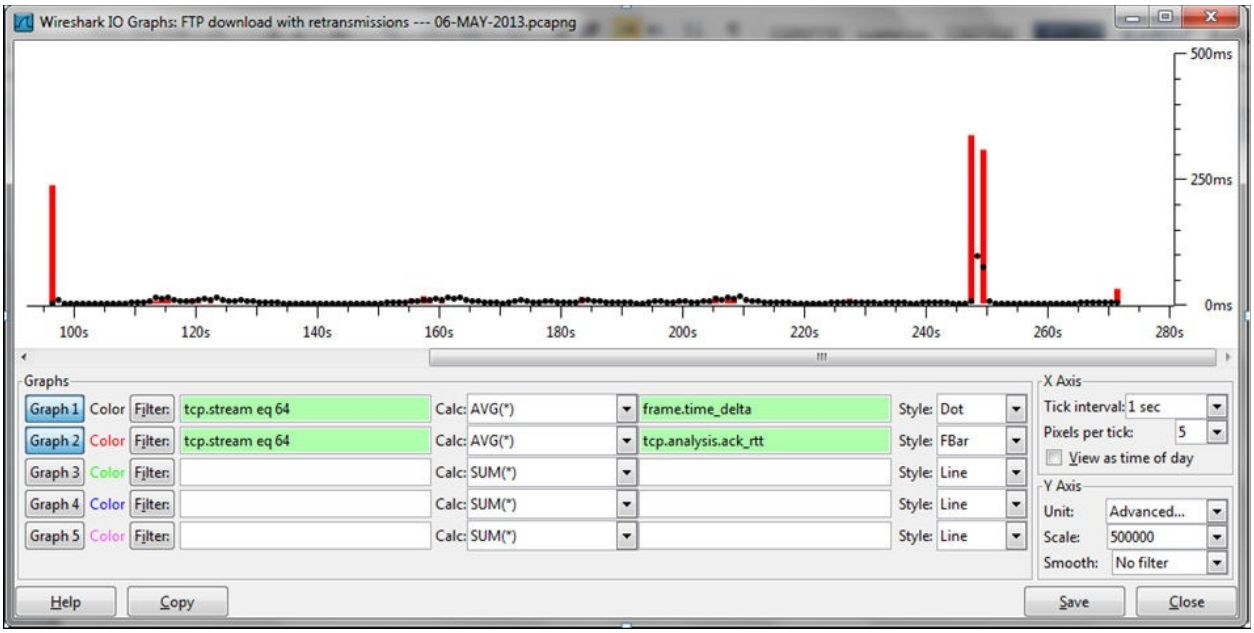
connections. Start with assuming 0.2 seconds. Refer to the *How it works...* section in this recipe for explanation about the RTO mechanism.

- To check TCP delay over a connection, use **IO Graphs** with the following filters, as presented in the next diagram:
  - `tcp.stream eq <the stream number>` to get to the stream number right-click on a packet and select **Follow TCP stream**.
  - `frame.time_delta` to see the time difference between frames in the TCP stream. This parameter actually shows inter-frame delta in layer 2, but since it is shown only for the stream, it will show us inter-frame deltas in a specific TCP stream.

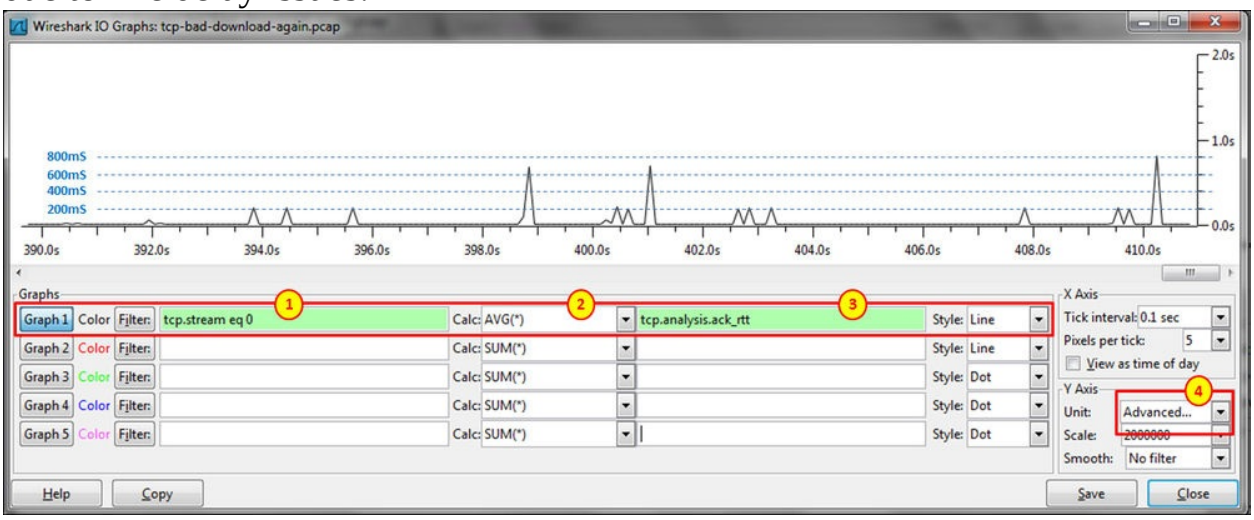


You get a graph that shows a very stable connection, with delays that are lower than 20 ms, except for the two increases in delay in time **250s** (250 seconds since the beginning of the capture), that causes retransmissions.

- When we add the `tcp.analysis.ack_rtt` filter on the same connection, we see the delays between TCP packets and ACKs.



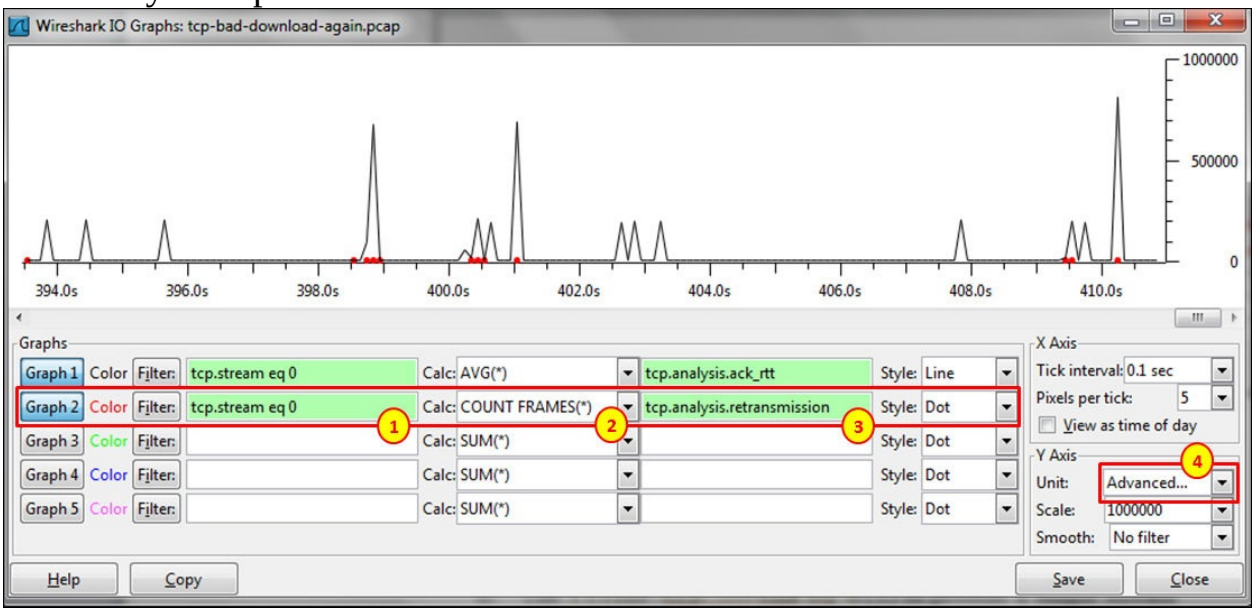
In the following diagram, you see a graph which is an example for delays not due to line delay issues:



- You can see here that I've configured:
  - The **Advance** option in the **Y Axis**
  - The **Filter** field: **tcp.stream eq 0** (numbered **1** in the preceding diagram) to present a single stream
  - The calculation **AVG(*)** to see the average (**2**)
  - You can also configure **MAX(*)** and the filter `tcp.analysis.ack_rtt` to see the time to acknowledge every TCP sequence

What we've got is the time that it took to acknowledge every TCP packet.

- Now, let's configure **IO Graphs** to see if there are TCP retransmissions, and why they happen:
  - Use the same IO Graph with **Advance** in **Y Axis**, and configure the second line.
  - The **Filter** field: **tcp.stream eq 0** (numbered **1** in the preceding diagram) to present a single stream.
  - The calculation **COUNT FRAMES(*)** to see the average.
  - The filter `tcp.analysis.retransmissions` to see the time to acknowledge every TCP packet.



- We can see from here that all retransmissions happened when there was a significant increase in the delay, so it is a delay problem. We cannot say from here if the delay is from the network, from the end device or from the application, so for isolating the problem check how retransmissions are distributed (see Chapter 9, *UDP/TCP Analysis*).
- Retransmissions that are not due to increase in RTT are probably due to packet losses.

How it works...

TCP uses the retransmission mechanism to ensure data delivery in the absence of any feedback from the remote data receiver. The duration of this timer is referred to as **Retransmission Time Out** (**RTO**). This mechanism was first standardized in RFC1122 that specified that the RTO should be calculated as outlined in the *Jacobson V. and M. Karels, Congestion Avoidance and Control,* article from 1988. An update to this RFC came out in RFC 2988 in November 2000, and later in RFC 6298 *Computing TCP's Retransmission Timer*, June 2011.

There's more...

For delay variations, you can also navigate to **Statistics** | **TCP Stream Graph** | **Round Trip Time Graph**.

When experiencing high delays, it also influences the throughput you can get from the network. This is what is called as the bandwidth delay product as shown in the following figure:

$$\text{Throughput [Bytes/Sec]} = \frac{\text{Window Size [Bytes]}}{\text{RTT [Sec]}}$$

From here we can see that the higher the delay is, the lower the throughput becomes. In networks with high delays, for example, old cellular networks, satellite lines, and long distance international lines, we have several methods to improve the application's throughput. Among these methods are applications that use multiple connections per application, usage of the TCP increases the window size (comes as default in Window Vista and the higher versions, along with various Linux versions).

Chapter 14. Understanding Network Security

In this chapter, we will cover the following recipes:

- Discovering unusual traffic patterns
- Discovering MAC- and ARP-based attacks
- Discovering ICMP and TCP SYN/Port scans
- Discovering DoS and DDoS attacks
- Locating smart TCP attacks
- Discovering brute-force and application attacks

Introduction

Information security is one of the fascinating areas in information systems, and its purpose is to secure the organization's systems against internal and external attacks that can come in various patterns. These attacks can come from the Internet or from the internal network, and as such, they all come through the network and therefore, can be monitored with Wireshark (and other tools that will be mentioned later).

For monitoring the network against malicious traffic, we must first understand what constitutes normal traffic. We can then try to find out how malicious traffic is short of being normal traffic. Among unusual traffic, we might see an ARP, IP, or TCP scanning, DNS responses without queries, unusual TCP flags, unknown IP addresses or port numbers whose purpose is not known to us, and so on.

It is also important to understand the difference between security problems and networking problems, and distinguish between them. For example, ICMP scan can be a malicious software scanning the network but also a management software that discovers the network, while TCP SYN scan can be a worm but also a software bug. We will elaborate on these in each of the recipes.

In this chapter, we will start by differentiating between normal and unusual network traffic and then understand the various types of attacks, where they come from and how to isolate and solve them.

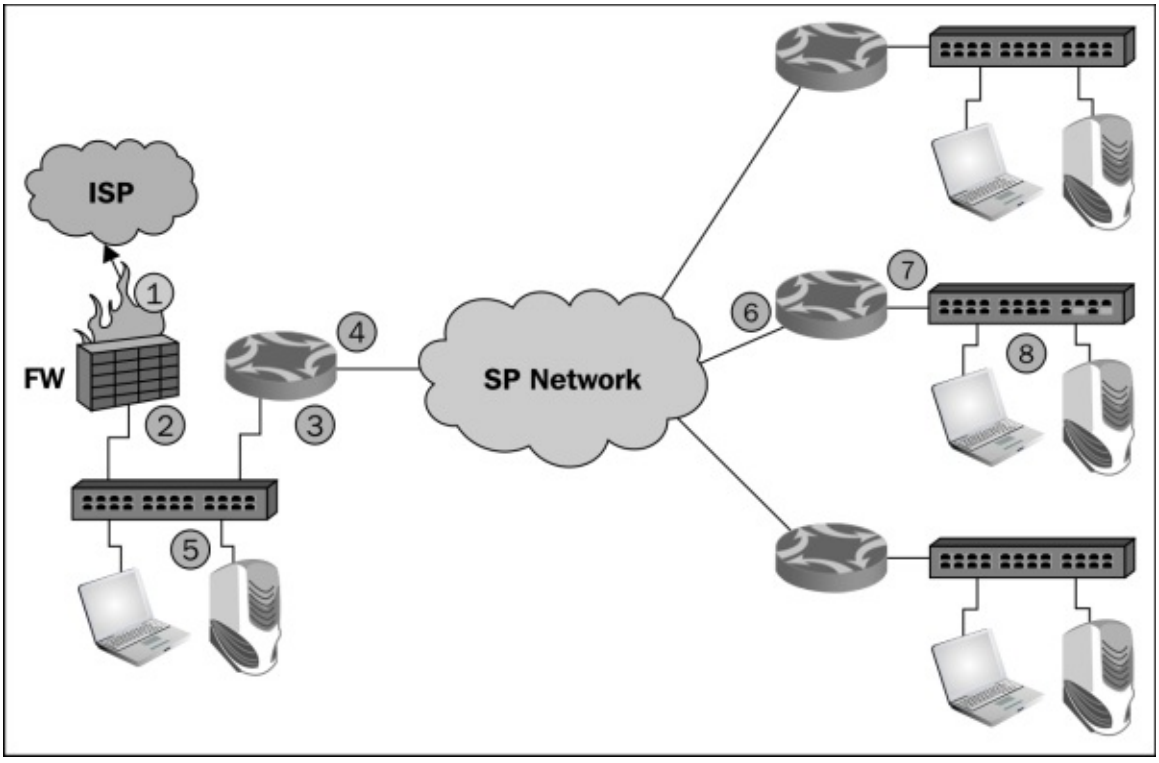
Discovering unusual traffic patterns

In this recipe, we will learn what are usual and unusual traffic patterns and how to distinguish between them.

Getting ready

The first thing is to locate Wireshark. There are several options for this (see the following diagram):

1. When you suspect an attack that comes from the Internet, locate Wireshark after the firewall (**1**), and when you suspect that it crosses the firewall, locate it before (**2**).
2. When you suspect malicious traffic coming from a remote office, port mirror the traffic coming on the central line before (**3**) or after (**4**) the router. In this case, you can filter the suspicious traffic with IP networks to see patterns from different offices in order to isolate the problematic office.
3. You can also port mirror the traffic in the remote office before (**7**) or after (**6**) the routers.
4. When a PC or a server is the suspect, port mirror its port on the switch (**5**) or (**8**).



Now, we will try to see what are the types of traffic that we should look out for,

what are the types of traffic that are normal, and what traffic should be followed.

Before starting with the tests, make sure that you have an updated topology of the network that includes:

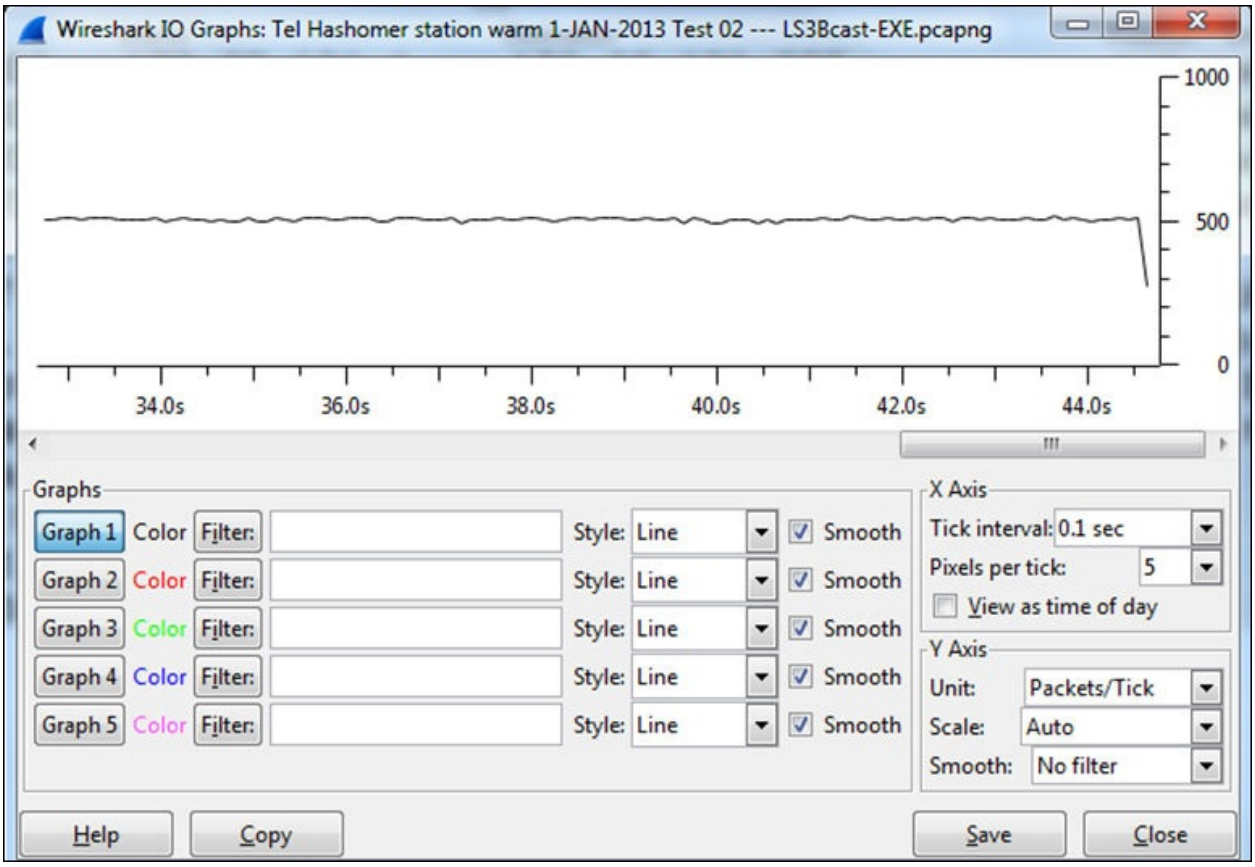
- Servers' IP addresses and LANs' IP address ranges
- Routers, switches, and other communications equipments' IP addresses and topology
- Security devices—firewalls, **Intrusion Detection Systems** / **Intrusion Prevention Systems** (**IDSs**/**IPSs**), **Web Application Firewalls** (**WAF**), database and application firewalls, antivirus systems, and any other device that has an IP address and generates, filters, or forwards network traffic
- What are the applications that work over the network including TCP/UDP port numbers and IP addresses of software

How to do it...

When you monitor internal traffic in your organization, the following things should be checked:

1. Traffic that is generated from known addresses (in the organization):
    - **Normal**: This is the traffic from known addresses and address ranges
    - **Suspicious**: This is the traffic from/to addresses that you don't know

- Applications and port numbers:
    - **Normal**: This includes standard port numbers, 80 (HTTP), 137/8/9 (NetBIOS), 3389 (RDP), 20/21 (FTP), 25,110 (Mail), 53 (DNS), and so on. Be sure of the applications that run over the network, and verify that these are the only port numbers that you see.
    - **Suspicious**: This includes unusual port numbers, that is, port numbers that do not belong to applications that run on server (for example, RDP packets to web server).
- TCP patterns:
    - **Normal**: TCP `SYN`/`SYN-ACK`/`ACK` that indicates a connection establishment, single reset (RST) that indicates a fast connection tear-down, `FIN`/`FIN-ACK` packets that indicate a regular tear-down of a connection, standard packets, and acknowledgments
    - **Suspicious**: Large amount of `SYN` packets that go to a single or multiple destinations or coming from multiple sources (usually in a scan pattern that will be described later in this chapter), unusual flags combination (`RST`/`FIN`, `URG`), and so on
- Massive traffic to a single or multiple sites that you don't know about:
    - **Normal**: Traffic patterns are usually not of fixed bandwidth. When you save or open files, browse the Internet, send or receive mails, or access a server with RDP, you see ups and downs.
    - **Suspicious** (in some cases): Fixed bandwidth patterns can indicate that someone is connected to your device, but it can also indicate that someone is listening to the radio over the Internet (100-150 Kbps), watching video (in some cases), and so on. When you see a fixed bandwidth pattern of traffic, check what it is. A fixed bandwidth pattern is illustrated in the following screenshot:



- Broadcasts:
  - **Normal**: NetBIOS broadcasts, ARP broadcasts (not too many), DHCP (not too many), application broadcasts (usually once every several seconds and more), and so on
  - **Suspicious**: Tens, hundreds, or thousands and more broadcasts per seconds per device
- DNS queries and responses:
  - **Normal**: A standard query-response pattern up to several tens per second per client, occasionally
  - **Suspicious**: Massive amount of DNS queries and/or responses, responses without queries, and so on

How it works...

Network forensics is quite similar to what you see in police dramas on television. Something is going wrong; so, you go to the crime scene (this is your network) and look for evidence (these are the traces that are left in the network).

What you look for are the things that do not match the crime scene (your network), things that are left behind (unusual traffic patterns), fingerprints, and DNA (patterns that can identify the attacker).

In the following recipes, we will get to the details of various types of attacks and abnormalities that can indicate that a crime was committed, and we will see how to isolate the problems and solve them.

Some common attacks that can come from the network are:

- **Viruses**: These are small programs that attack your computer and try to cause damage. Viruses should be discovered and fixed by antivirus software.
- **Worms**: These are usually programs that attempt to replicate themselves across the network. There is a major impact on resource consumption, for example, bandwidth consumption and CPU load. The important thing is that the moment you fix the problem, everything will go back to normal.
- **Denial of Service (DoS) and Distributed DoS (DDoS)**: These are attacks that deny access to network resources. These types of attacks are usually very easy to discover since they have a distinct behavior that can be located easily.
- **Man-in-the-middle attacks**: These are attacks in which the attacker intercepts messages and then retransmits them. In this way, the attacker can eavesdrop on the traffic or change it before it gets to the destination.
- **Scanning**: There are various types of scans ranging from simple ICMP scans that usually are a form of DDoS, TCP scans that send, for example, SYN requests on various port numbers in order to try and open connections to services running on a server, and also application scans that try to connect to applications running on your servers.
- **Application-layer attacks**: These are attacks that target applications on your servers by intentionally causing a fault in a server's operating system or applications.

In the following recipes, we will see each of them (and some more).

There's more...

An important indication that something went wrong is when a server, a PC, a communication link, or any other entity on the network becomes slow without any logical reason. For example:

- When a server becomes slow, check for hardware and software issues, check for network problems, but also check if someone is attacking it
- When a link from a remote office to the center becomes slow, it can be because of the load (constant or sudden), but it can also be because of an attack that blocks it (usually DOS/DDoS)
- When a PC becomes slow, it can be because it is doing something that you know about, but there is not just one possibility, check for the things you don't know

It is important to mention here that there are various systems that can protect us from attacks; a few of them are listed as follows:

- **Firewalls**: They protect unauthorized traffic from getting into specific areas. Firewalls can be located on the connection to the Internet, before the organization servers, between organization areas, and even as personal firewalls on every PC.
- **Network Access Control** (**NAC**): These systems allow only authorized users to connect to the network. When connecting an unauthorized device to the network, you will see that the link on the device will be turned on and immediately off, and the unauthorized device will be blocked on the MAC layer.
- **IDS/IPS**: These systems can identify intrusion patterns and block them. There are usually two lines of defense here—one at the ISP network and one at the customer premises. IDS/IPS can be a dedicated device located between the firewall and the Internet or an additional software on the firewall.
- **Web Application Firewalls** (**WAF**), **Application Firewalls**, **Database Firewalls, and other application protection devices**: This group of products are layer-7 protection devices that look inside the applications and forward or block application layer attacks.
- **Web Filters and Mail Filters**: These are devices that scan mail and/or web content and forward only those messages and traffic that are allowed.

The features mentioned above can come as different devices, software on Virtual Machines (VMs), or features on the same device.

See also

In this recipe, we talked about some security components. Some examples are:

- **Firewalls**: Checkpoint (www.checkpoint.com), Juniper SSG series (http://www.juniper.net/us/en/products-services/security/ssg-series/), Cisco ASA series (http://www.cisco.com/en/US/products/ps5708/Products_Sub_Category_Ho and many others.
- **NAC**: In this category, you have, for example, Forescout (http://www.forescout.com/solutions/network-access-control/) and Enterasys (http://www.enterasys.com/company/literature/nac-ds.pdf).
- **IDS/IPS**: In this category, we have, for example, the Juniper IDP device series (http://www.juniper.net/us/en/products-services/security/idp-series/) and the Check Point software blade for the firewall (http://www.checkpoint.com/products/ips-software-blade/).
- **WAF**: Here we have, for example, Imperva (http://www.imperva.com/products/wsc_web-application-firewall.html) and F5 (http://www.f5.com/glossary/web-application-firewall/). Database firewalls are available, for example, from Imperva (http://www.imperva.com/products/dsc_database-firewall.html) and Oracle (http://www.oracle.com/us/products/database/security/audit-vault-database-firewall/overview/index.html).
- **Web and mail filters**: Here we have, for example, McAfee (http://www.mcafee.com/au/products/email-and-web-security/index.aspx), Blue Coat (http://www.bluecoat.com/security-policy-enforcement-center), and Websense (http://www.websense.com/content/Home.aspx).

Discovering MAC- and ARP-based attacks

There are various types of layer-2 MAC-based attacks and layer-2/3 ARP attacks that can be easily discovered by Wireshark. These attacks are usually caused by scanners (described in the next recipe) and man-in-the-middle attacks (described in the *Analyzing connectivity problems with ARP* recipe in Chapter 8, *ARP and IP Analysis*). In this recipe, we will see some typical attack patterns and their meanings.

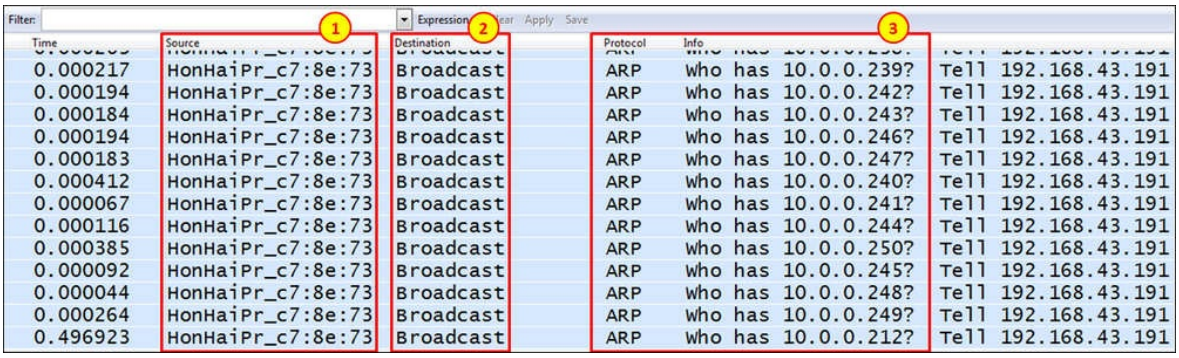
Getting ready

When viewing too many ARP requests on a network or when seeing non-standard MAC addresses in the network, connect Wireshark with port mirror to their source and start the capture.

How to do it...

To look for ARP/MAC-based attacks, follow these steps:

1. Connect Wireshark to any port on the network.
2. Look for massive ARP broadcasts. Since ARP requests are broadcasts, they will be distributed in the entire layer-2 network (that is, on a single VLAN). In the following screenshot, you can see a typical ARP-scan pattern. It's important to note that this ARP scan can be an application that works this way, for example, SNMP software that discovers the network and router that uses gratuitous ARP. It is a problem only if it comes from an unidentified source.



3. There are also some suspicious MAC patterns. You can identify them when you see:
    - Two identical MAC addresses with different IP addresses. It can be two IP addresses configured on the same network adapter, which is OK, but it can also be an attack pattern in which someone has changed its MAC address to the MAC address of a server (can be performed in every adapter).
    - The case mentioned above can also indicate a man-in-the-middle attack as mentioned in the *ARP poisoning and man-in-the-middle attacks* section in Chapter 8, *ARP and IP Analysis*.

How it works...

ARP sends broadcasts to the network asking for the MAC address of a specific IP destination. Anything that is not according to this pattern should be considered malicious.

There's more...

ARP requests can also come from the SNMP software that discovers the network (auto-discovery feature), the DHCP server that sends gratuitous ARP, and so on. Whenever you see ARP scanning something, it is not necessarily a problem; the question is who sends them. You can find more information on the ARP process in Chapter 8, *ARP and IP Analysis*.

Discovering ICMP and TCP SYN/Port scans

Scanning is the process of sending packets to network devices in order to see who is answering the ping requests, to look for listening TCP/UDP ports, and to find which types of resources are shared on the network including system and application resources.

Getting ready

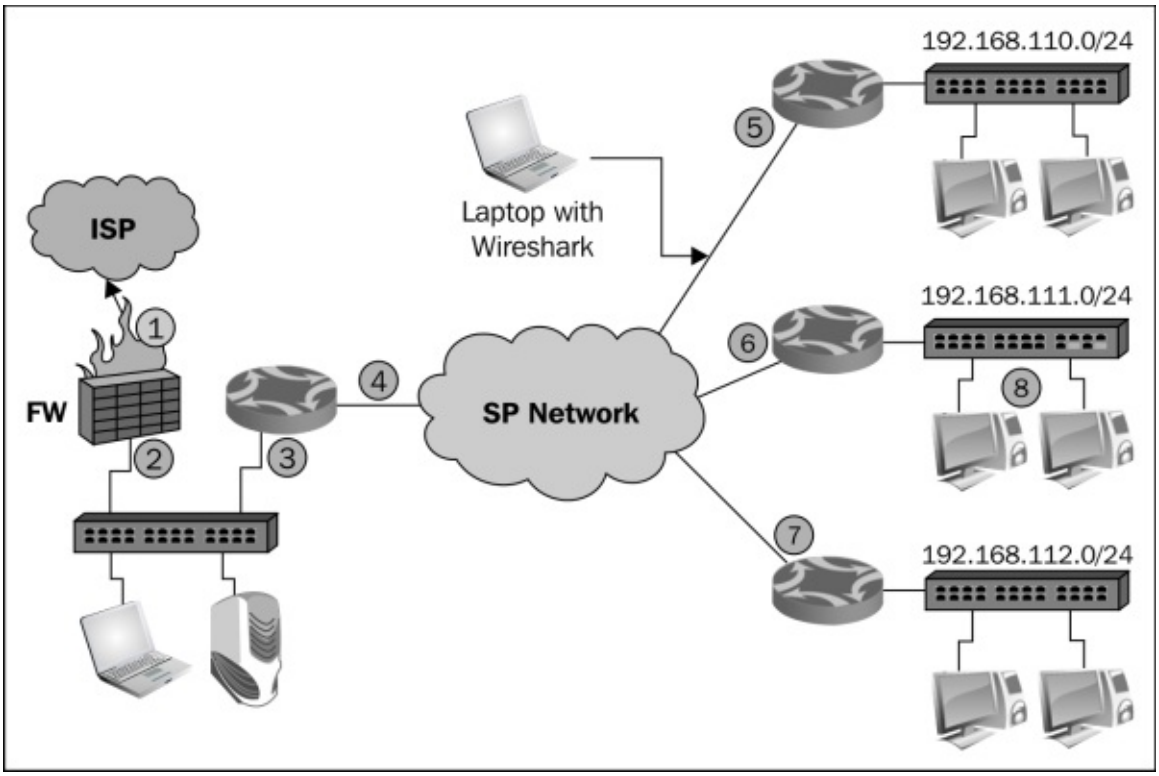
A scanning attack is usually detected by users complaining about slow network responses, management systems that discover unusual load on servers or communication lines, and when the attack is implemented also by **Security Information and Event Management Systems** (**SIEM**) that identifies suspicious usage patterns. In these cases, locate the Wireshark with port mirror as close as possible to the area that you suspect is infected, and start capture.

How to do it...

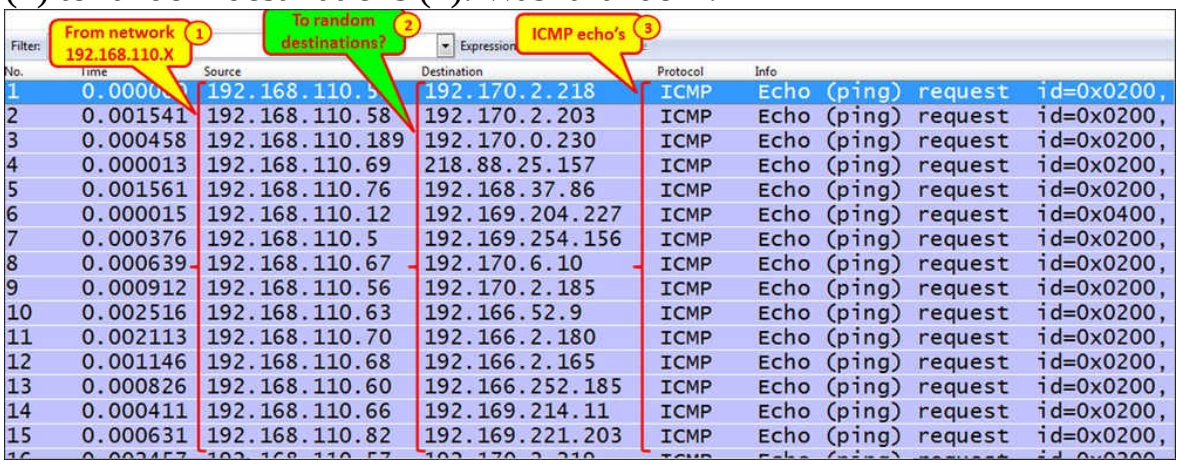
To discover the problem, follow these steps:

1. Start Wireshark with capture on the interface that is close to the problem:
    - If the line to the Internet becomes slow, port mirror the line
    - If a server becomes slow, port mirror the server
    - If remote offices become slow, port mirror the lines to them

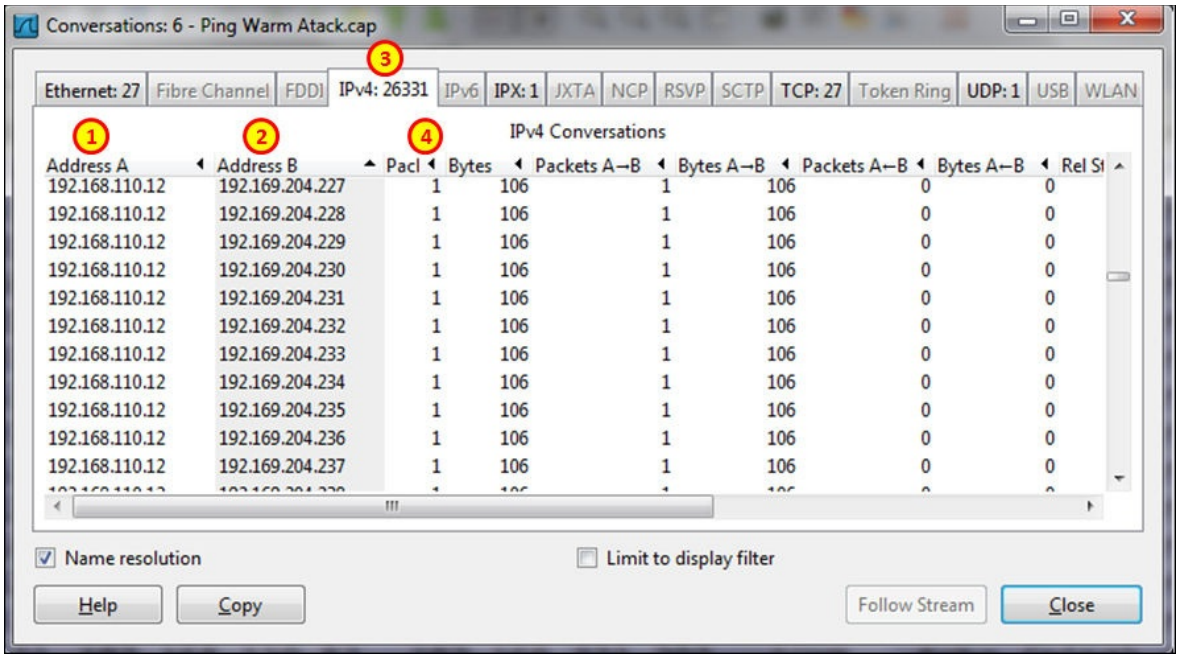
- If you see that Wireshark does not respond, it is probably because you have a very strong attack that generates thousands or more packets per second; so, Wireshark (or your laptop) cannot process them. In this case, stop Wireshark (with *Ctrl+Alt+Del* in Windows or with the `kill` command in Unix if necessary) and configure it to capture multiple files (described in the *Starting the capture of data* recipe in Chapter 1, *Introducing Wireshark*)
- There are various patterns that you might see, all of them with the same behavior— massive scanning, ICMP or TCP in most of the cases, but also other types. The best way to understand all is to see them with some examples.
- In the following diagram, you see a network that was under attack. Users from all the remote sites complained about a very slow network. They were all accessing servers on the center on the left-hand side of the diagram.



- What I got when I connected Wireshark to a remote site (as illustrated below) was many ICMP requests (3), coming from the LAN 192.168.110.0 (1) to random destinations (2). Was it random?



- Also, look at the time between packets. If scanned, it will usually be very short.
- When you go to **Statistics | Conversations**, you will see something interesting:



- When we sort the table by address A (**1**), we see a pattern of ICMP requests coming from various addresses on the network 192.168.110.0 (here, we see a very small part of it, that is, 192.168.110.12 scans the network).
- This worm simply scans the network with ICMP requests. The moment someone answers, the worm infects him/her also, and after a few minutes, all communication lines are blocked with ICMP requests going out of the remote offices.

Tip

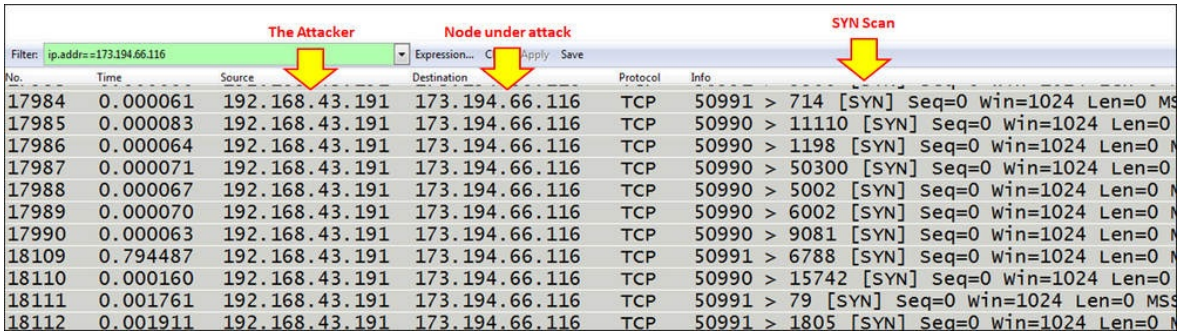
Conclusion

When you see a massive number of pings scanning on a communication channel or link, that is, thousands and more pings, check for the problem. It can be the SNMP software discovering the network, but it can also be a worm that will flood your communications line or server links (or both).

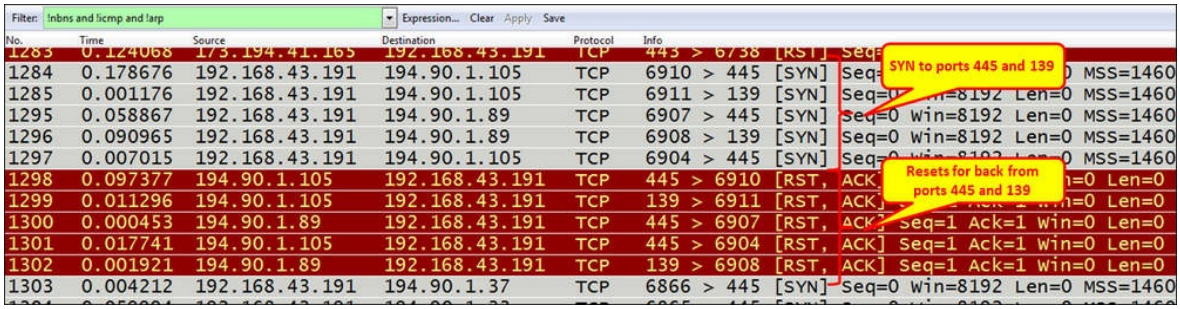
- Another common type of scan is the TCP-SYN scan. In this case, the attacker scans random TCP ports with TCP-SYN packets waiting for someone to answer with SYN-ACK. The moment it happens, there are two options:
  - The attacker will continue to send SYN packets and receive the SYN-ACKs, thus leaving many half-open connections on the device under attack
  - The attacker will answer with ACK, thus initiating the connection, and

leave it open as in DoS/DDoS attacks or try to harm the device under attack with this connection

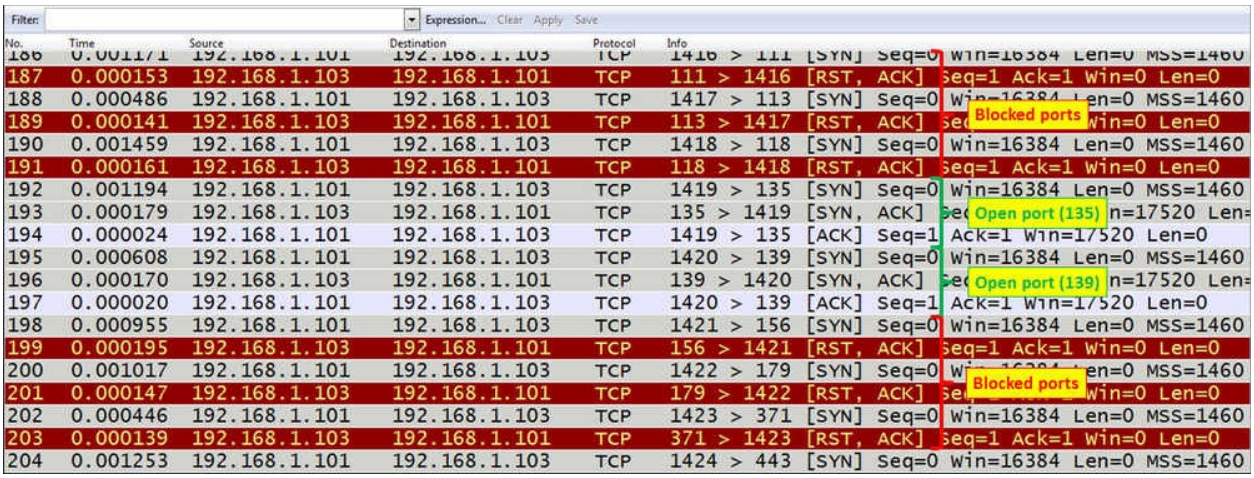
- The TCP-SYN scan will look like one of the patterns in the following screenshots:
  - You will see many SYN packets without any response from the node under attack.



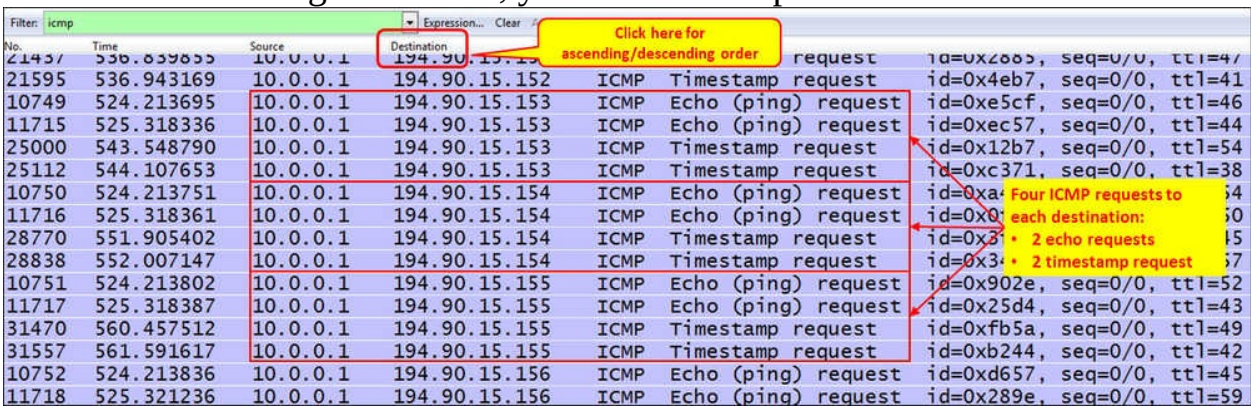
  - You will see many SYN packets when a TCP RST packet is sent as a response to each one of them. This is usually when you have a firewall on the device that is under attack or will be attacked.



  - You can also have consecutive SYN and RST packets. When there is a port number that is opened, you will see the complete SYN/SYN-ACK/ACK when the scanner opens connection to the victim. This is illustrated in the following screenshot:



- Always look for unusual traffic patterns. Too many ICMP requests, for example, are a good indication for scanning. Look for multiple ICMP requests to clients, ICMP timestamp request, ICMP in ascending or descending order, and so on. These patterns can indicate malicious scanning.
  - When you suspect a scan, click on the title of the destination (address), and you will get the packet list sorted by the destination address. In this way, it will be easier to see the scan patterns.
  - In the following screenshot, you see an example of this scenario:



- In the case of application scanning, you can have various types of scans:
  - **NetBIOS scans**: It looks for massive scanning of NetBIOS ports
  - **HTTP**: It looks for SYN requests to HTTP port `80` with HTTP requests later on
  - **SMTP**: It looks for massive scanning on the TCP port `25`
  - **SIP**: It looks for massive requests on port `5060`

Other types of applications are scanned according to their port numbers

How it works...

The majority of scanners work in several steps: ARP scanning, ICMP, and then TCP or UDP. The principle is simple:

- If the scanner is on the LAN, it sends an ARP broadcast to the entire LAN.
- The scanner sends ICMP requests. Some of the ICMP requests will be answered.
- When someone answers the ARP or ICMP request, it goes up to TCP and UDP and starts scanning the layer-4 ports. When the scanner finds out that a port is open, it starts with application scanning.
- In application scanning, the scanner sends commands to the applications, trying to get the application to answer, and in this way, try to break into it.

There's more...

Most of the modern intrusion detection/prevention systems (IDS/IPS) in the last several years know how to deal with ICMP scans, TCP SYN scans, and various types of scans that generate massive traffic of standard, well-known attack patterns. In case you have such a system and you connect to the Internet with an ISP that has their systems, you are probably protected from these simple types of attacks.

These systems usually work in two ways:

- NetFlow/Jflow-based IDS/IPS that identifies massive traffic coming from several sources; they neutralize it by blocking it or changing the routing tables to disable these packets from getting to the ISP network
- Content-based IDS/IPS that looks at the traffic patterns and accordingly decides whether to forward it or not

Attacks coming from the internal network are not filtered by the external devices, and therefore, are even more common. There are more sophisticated types of attacks that will be discussed in the *Locating smart TCP attacks* recipe later in this chapter.

The way to prevent attacks coming from the Internet is to connect through an ISP with efficient IDS/IPS systems along with using one of your own. The way to prevent attacks coming from the internal network is to implement organizational security policy along with appropriate protection software such as antivirus and personal firewalls.

See also

In the previous section, I've mentioned the issue of organizational security policy, that is, how to implement a set of rules for securing your organization. Further information on this subject is widely available on the Internet. Some interesting websites that cover this area are:

- http://www.cert.org/work/organizational_security.html
- http://www.praxiom.com/iso-17799-4.htm
- http://www.sans.org/reading-room/whitepapers/policyissues/1331.php
- http://www.sans.org/security-resources/policies/

Discovering DoS and DDoS attacks

Denial of Service (**DoS**) and **Distributed Denial of Service** (**DDoS**) are attacks that intend to deny users from accessing network services. Services that can be denied to users can be:

- **Communication lines**: This will usually be done by generating traffic that floods and blocks the communications line
- **Applications and services (web services, mail services, and so on)**: This will usually be done by loading a server to a point at which it will not be able to serve clients' requests

DoS/DDoS attacks can be a result of scanning that we talked about in the previous recipe. The difference is that DoS/DDoS is a scan that slows down a server or a network in a way that denies user access.

In this recipe, we will see some common DoS/DDoS patterns, and learn how to identify and block them.

Getting ready

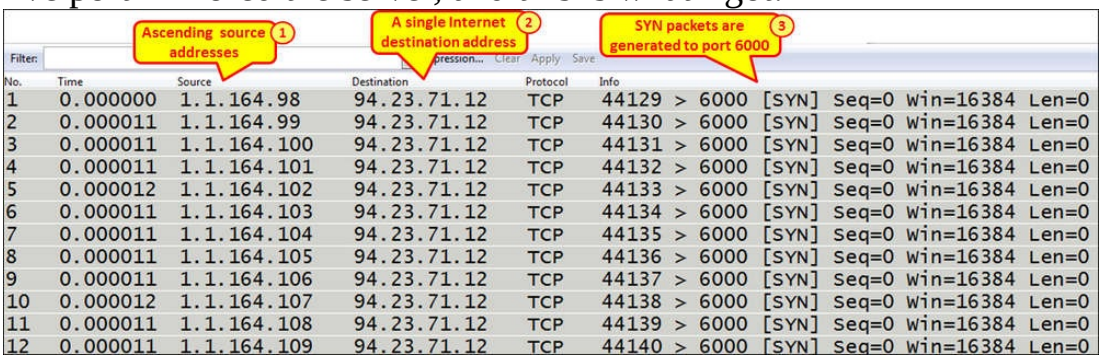
DoS/DDoS are usually discovered when one of the network resources, that is, communications lines or servers becomes very slow and is also not functioning.

When you identify such a resource, connect Wireshark with port mirror to this device and start packet capture. In this recipe, we will go through some common DoS/DDoS attacks and their signatures.

How to do it...

Connect Wireshark to the network with port mirror to the port of the resource that you suspect is exposed to DoS/DDoS. Usually, it will be a server that becomes very slow, a communication line that becomes very loaded, or any other resource that stops functioning or becomes very slow.

1. When a communication line becomes very slow, for example, a connection to the Internet, connect Wireshark with port mirror to this line.
    1. Try to locate where the traffic comes from.
    2. I've port mirrored the server, and this is what I got:

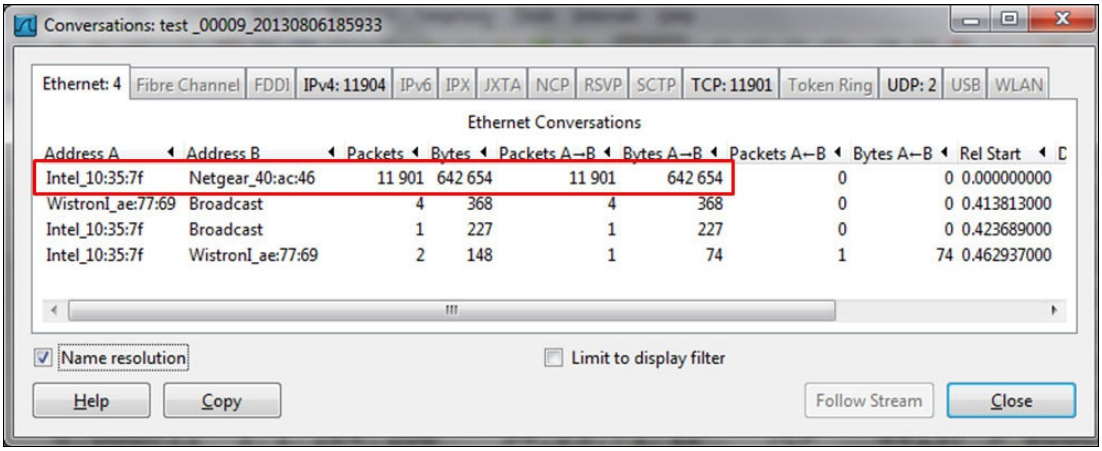


    3. We see source addresses in the ascending order, generating traffic to the Internet address **94.23.71.12**.

Tip

When you look at the time column that is configured with "time since the previously displayed packet", you see that there are 11-12 micro-seconds between frames. When you see TCP-SYN coming at this rate, something is wrong. Check what it is!

4. Since the source addresses are unknown, I've checked their MAC address. What I got was:



5. The problem was that all source addresses came from a single MAC address; so I checked their MAC addresses, and all IP addresses came from a single MAC address, the MAC address of the server.

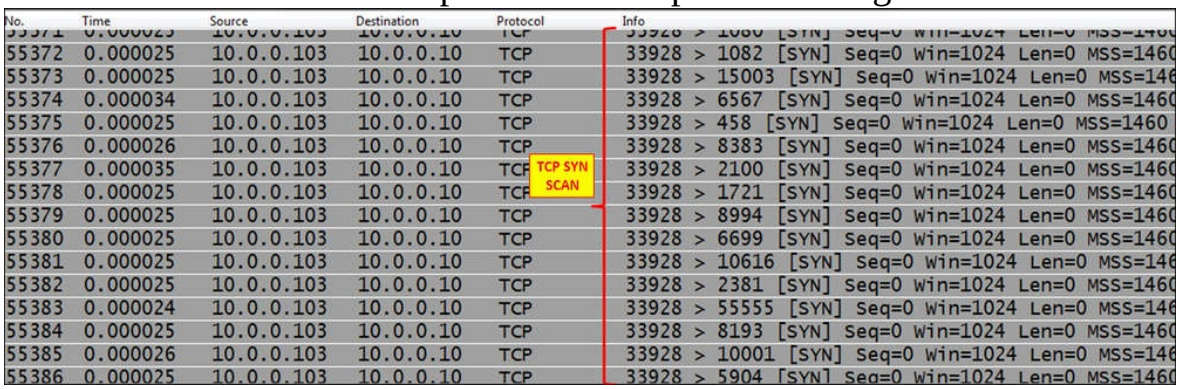
Tip

Conclusion

Check for SYN scans, and verify which IP and MAC addresses they are coming from. It can be that a worm is generating source addresses that are not the addresses of the host.

2. Another example can be a simple SYN scan that comes from a single attacker, as seen in the next illustration. Look for SYN and watch the port numbers that they are scanning. You might see:
    - No response
    - Reset packet
    - SYN-ACK response
3. There can be various consequences to this type of attack:
    - In case of no response or reset response, the attacked server is functioning well. In case the server answers with a SYN-ACK response, it might be a risk to the server.
    - The risk is that if too many connections will be opened (SYN/SYN-ACK/ACK) or half opened (SYN/SYN-ACK), the server might get slow due to these connections.
    - You can see a typical TCP SYN attack in the following illustration. A SYN attack becomes DoS/DDoS when it blocks a communication line

or loads a server to the point that it stops functioning.



No.	Time	Source	Destination	Protocol	Info
55372	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 1082 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55373	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 15003 [SYN] Seq=0 Win=1024 Len=0 MSS=146
55374	0.000034	10.0.0.103	10.0.0.10	TCP	33928 > 6567 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55375	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 458 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55376	0.000026	10.0.0.103	10.0.0.10	TCP	33928 > 8383 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55377	0.000035	10.0.0.103	10.0.0.10	TCP	33928 > 2100 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55378	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 1721 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55379	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 8994 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55380	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 6699 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55381	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 10616 [SYN] Seq=0 Win=1024 Len=0 MSS=146
55382	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 2381 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55383	0.000024	10.0.0.103	10.0.0.10	TCP	33928 > 55555 [SYN] Seq=0 Win=1024 Len=0 MSS=146
55384	0.000025	10.0.0.103	10.0.0.10	TCP	33928 > 8193 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
55385	0.000026	10.0.0.103	10.0.0.10	TCP	33928 > 10001 [SYN] Seq=0 Win=1024 Len=0 MSS=146

How it works...

Denial of Service is an attack that denies the use of a network service. The way to do this is by causing the device under attack to allocate hardware resources (CPU, memory, and so on) to the attacker so that nothing is left for the users.

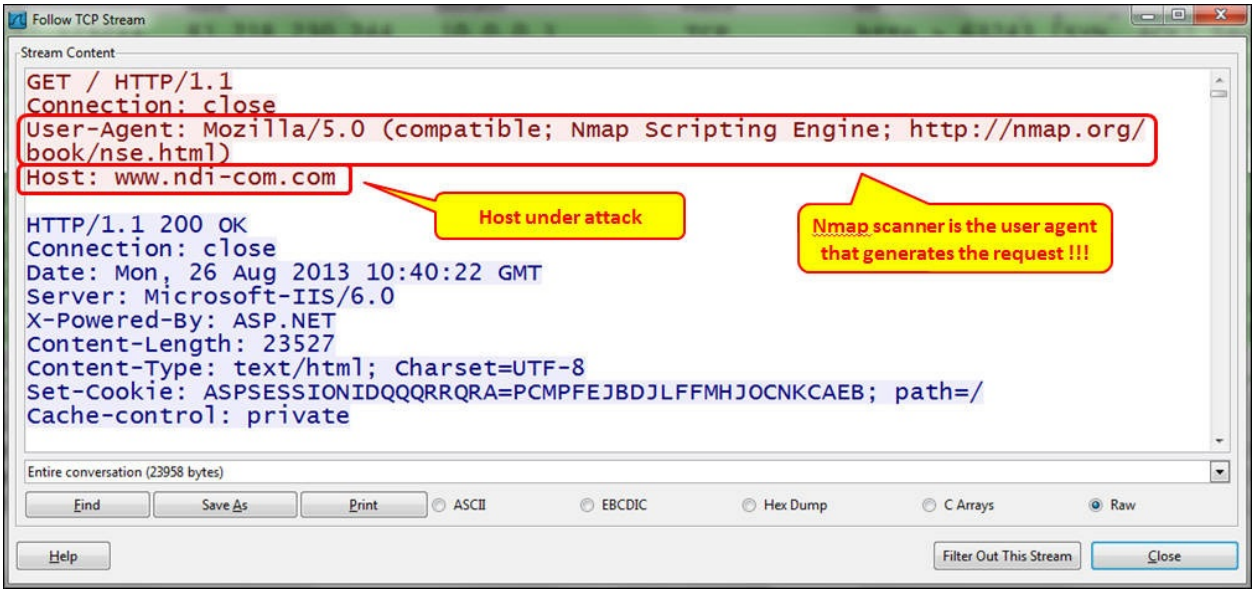
Denial of Service is when there is an attack on a network resource. Distributed DoS is when the attack is coming from multiple sources.

There's more...

DoS/DDoS attacks are sometimes hard to discover since they can simulate a real situation. For example:

1. Ping scans that can also come for management systems.
2. HTTP GET requests that are the normal requests that are accepted by web servers.
3. SNMP GET requests.

These and many others should be monitored for their quantity and sources in order to discover a problem. In the following screenshot, we see what we get when we follow a specific TCP stream.



Locating smart TCP attacks

Another type of attack is when you send unknown TCP packets, hoping that the device under attack will not know what to do with them and hopefully pass them through. These types of attacks are well known, and blocked by most of the modern firewalls that are implemented in networks today; but still, I will tell you about them in brief.

Getting ready

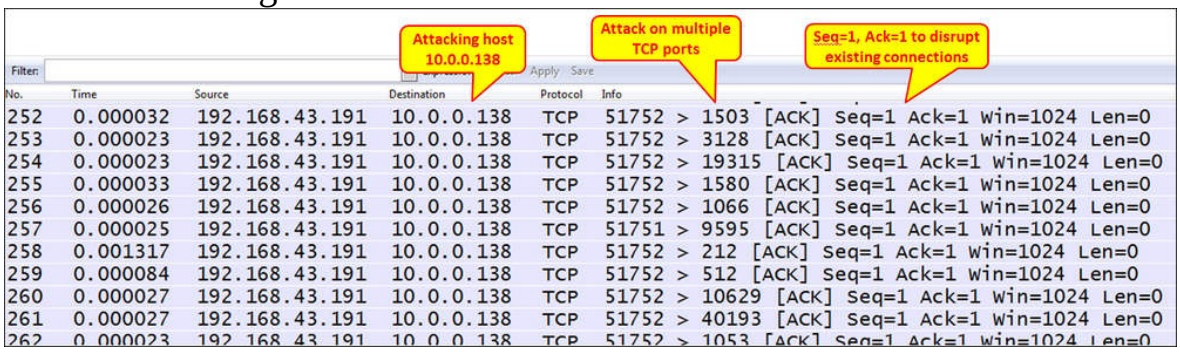
What I usually do when I get to a new network is connect my laptop to the network and see what is running over it. First, I just connect it to several switches and see the broadcasts. Then I configure port mirror to critical servers and communications lines and look at what is running over it.

To look for unusual traffic, port mirror communications links and central servers, and check for unusual traffic patterns.

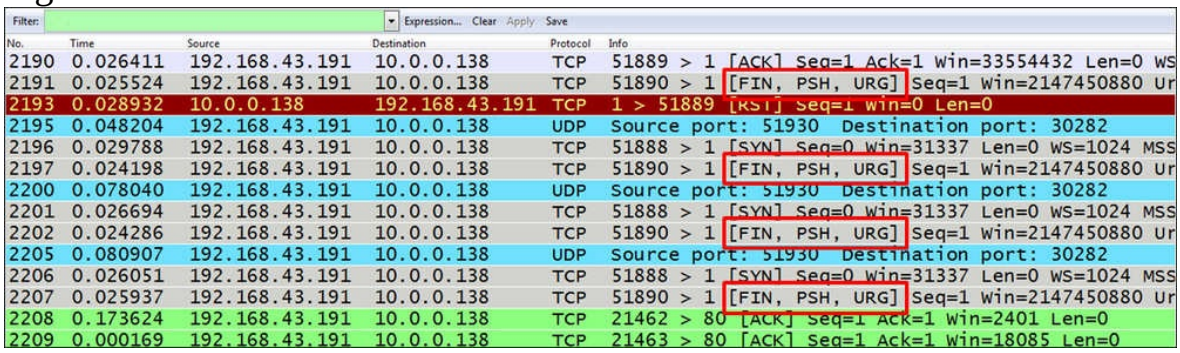
How to do it...

The traffic patterns you should look for are:

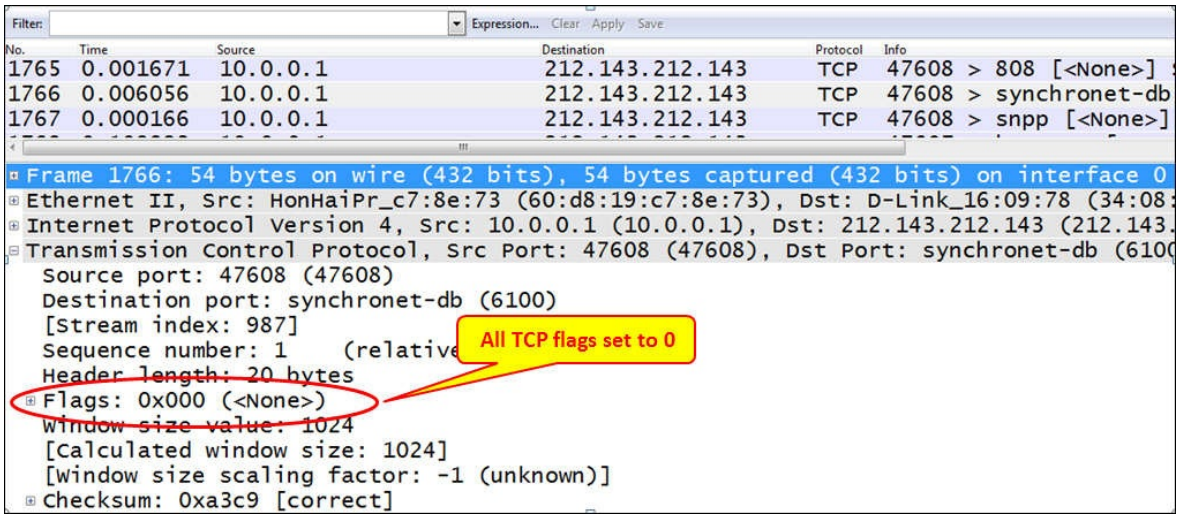
- ACK scanning: Multiple ACKs are sent usually to multiple ports in order to break the existing TCP connections.



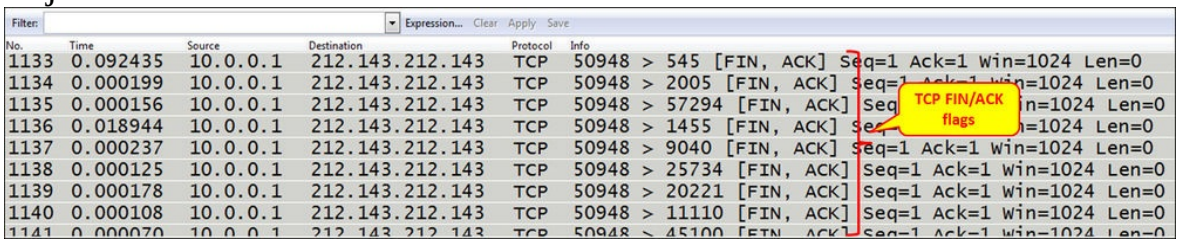
- Unusual flags combinations: This refers to anything with a URG flag, FIN and RST, SYN-FIN, and so on. Unusual flags combinations are not the usual SYN, FIN or RST, with or without ACK. In the following screenshot, you see an example of this scenario. The operations FIN/PSH/URG are together called Xmas scan.



TCP scans with all flags set to "0". This scan is called Null scan.



- **Massive FIN-ACK scanning**: Large amount of packets with FIN and ACK flags set to "1" are sent to multiple ports in order to cause them to be closed or just to flood the network.



How it works...

There are many types of TCP scans based on the assumption that when we send target RST or FIN flags (with or without an ACK) that scan various port numbers, we will cause the target to close connections, and when we send unusual combinations of flags to it, it will make the target busy. This will cause it to slow down and drop the existing connections.

Most of these scans are well known and well protected against firewalls and intrusion detection/preventions systems.

There's more...

You can also configure pre-defined filters to catch these types of attacks, but the best thing to do while suspecting such an event is to go through the captured data and look for unusual data patterns.

See also

For scan types, go to the Nmap.org web page:

http://nmap.org/book/man-port-scanning-techniques.html

Discovering brute-force and application attacks

The next step in network attack is to understand the various types of brute-force attacks. A brute-force attack is a trial-and-error method used to obtain information from the victim, for example, trying to find organizational servers, user directories, and crack passwords.

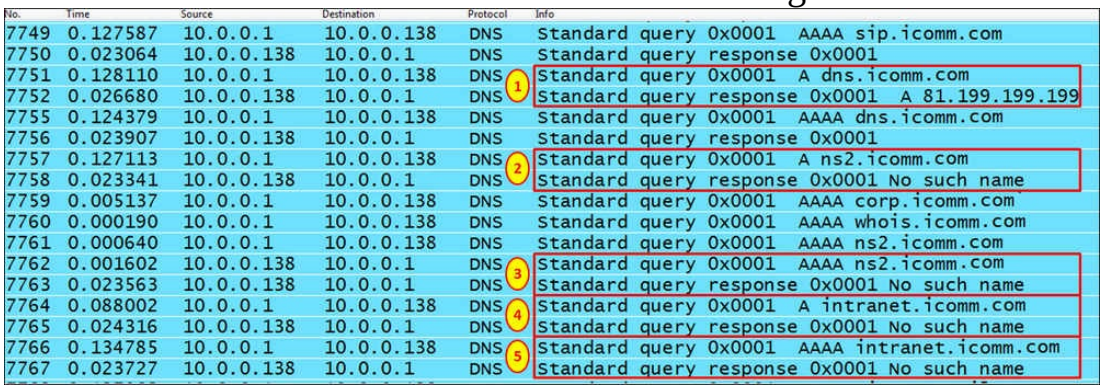
Getting ready

Brute-force attacks usually will not produce non-standard loads on the network, and the way they are discovered is usually by IDS systems or when there is a suspicion that someone is trying to hack into the network. In this recipe, we will learn how to identify typical brute-force attacks.

How to do it...

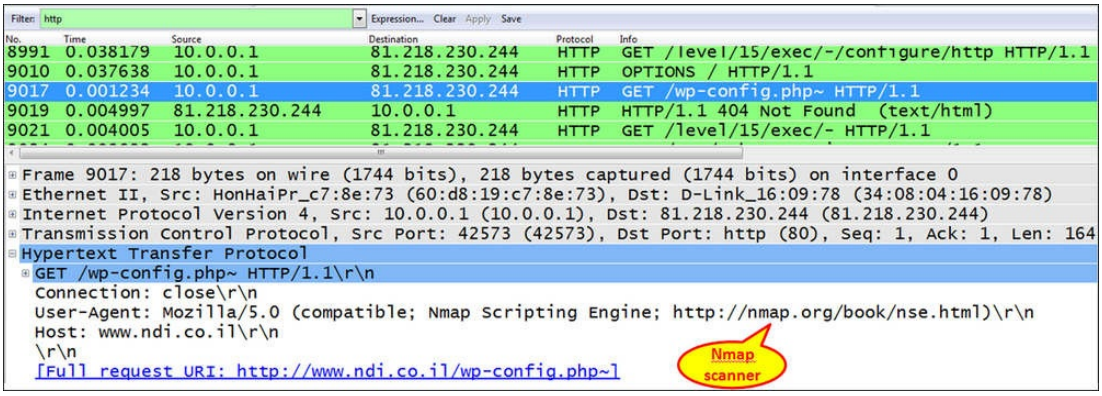
When you suspect a brute-force on the network, follow these steps to locate it.

1. Connect Wireshark with port mirror to the port in the server that you suspect is under attack.
2. For DNS brute-force attacks, look for DNS queries that are asking for common names under your domain. For example, in the following illustration, you can see a scan for ISP servers. We can see DNS queries to common names such as dns (1) and dns2—a record for IPv4 (2) and a record for IPv6 (3), and intranet—a record for IPv4 (4) and a record for IPv6 (5).
    1. In the case of **dns.icomm.co** (1), we got a reply; in all other cases, we did not.
    2. Many queries with no response can indicate a DNS brute attack, but also indicate someone who is looking for a server that does not exist. Look at the source address to see where it is coming from.

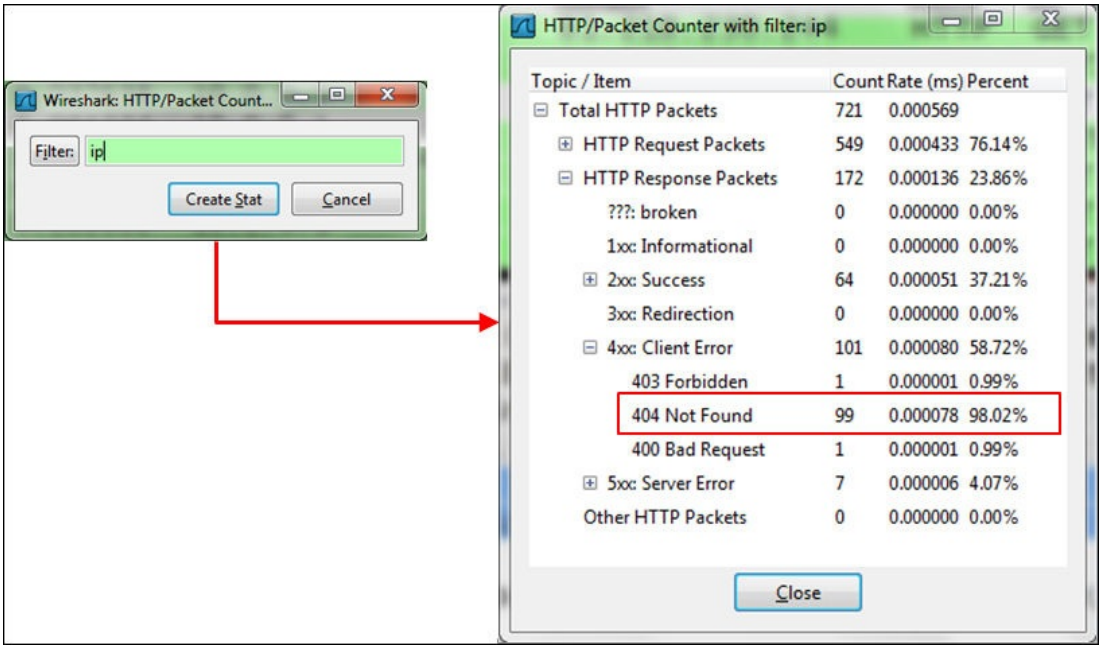


No.	Time	Source	Destination	Protocol	Info
7749	0.127587	10.0.0.1	10.0.0.138	DNS	Standard query 0x0001 AAAA sip.icomm.com
7750	0.023064	10.0.0.138	10.0.0.1	DNS	Standard query response 0x0001
7751	0.128110	10.0.0.1	10.0.0.138	DNS (1)	Standard query 0x0001 A dns.icomm.com
7752	0.026680	10.0.0.138	10.0.0.1	DNS (1)	Standard query response 0x0001 A 81.199.199.199
7755	0.124379	10.0.0.1	10.0.0.138	DNS	Standard query 0x0001 AAAA dns.icomm.com
7756	0.023907	10.0.0.138	10.0.0.1	DNS	Standard query response 0x0001
7757	0.127113	10.0.0.1	10.0.0.138	DNS (2)	Standard query 0x0001 A ns2.icomm.com
7758	0.023341	10.0.0.138	10.0.0.1	DNS (2)	Standard query response 0x0001 No such name
7759	0.005137	10.0.0.1	10.0.0.138	DNS	Standard query 0x0001 AAAA corp.icomm.com
7760	0.000190	10.0.0.1	10.0.0.138	DNS	Standard query 0x0001 AAAA whois.icomm.com
7761	0.000640	10.0.0.1	10.0.0.138	DNS	Standard query 0x0001 AAAA ns2.icomm.com
7762	0.001602	10.0.0.138	10.0.0.1	DNS (3)	Standard query 0x0001 AAAA ns2.icomm.com
7763	0.023563	10.0.0.138	10.0.0.1	DNS (3)	Standard query response 0x0001 No such name
7764	0.088002	10.0.0.1	10.0.0.138	DNS (4)	Standard query 0x0001 A intranet.icomm.com
7765	0.024316	10.0.0.138	10.0.0.1	DNS (4)	Standard query response 0x0001 No such name
7766	0.134785	10.0.0.1	10.0.0.138	DNS (5)	Standard query 0x0001 AAAA intranet.icomm.com
7767	0.023727	10.0.0.138	10.0.0.1	DNS (5)	Standard query response 0x0001 No such name

3. Another brute-force attack to watch out for is HTTP trying to find resources on the server.
    1. To look for HTTP scanning, look for the scanner's signature in the packet details, as seen in the following screenshot.



2. Also, look for too many HTTP error messages. Some examples are illustrated in the following screenshot. Choose **Statistics | HTTP | Packet Counter | PC**. If you get too many error messages, check for their source.

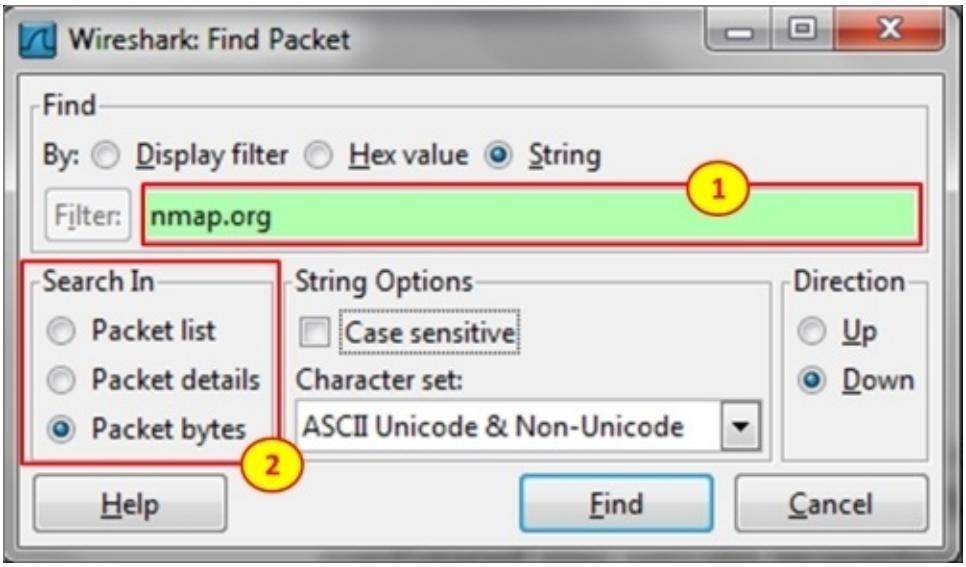


How it works...

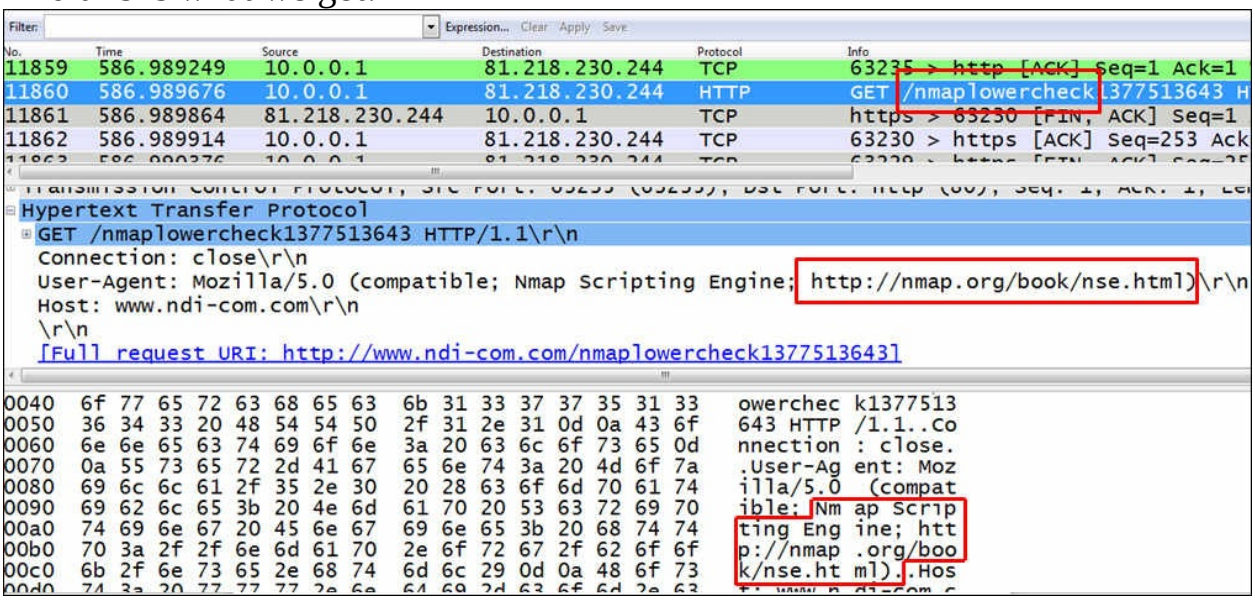
Brute-force attacks are trial and error attacks that send requests to the destination, hoping that some of them will be answered. Since most of these requests will be denied (if you've configured your servers properly), a large amount of **Not Found** messages, forbidden messages, and other error codes can be some of the syndromes for such an attack.

There's more...

For discovering HTTP error codes, configure the display filter `http.response.code >= 400`. The same applies to SIP and any protocol that uses HTTP-like codes. To find known scanners, you can simply use the **Edit** | **Find** packet feature and look for common scanner names. In the following screenshot, you can see an example for Nmap, which is one of the common ones. We chose the string **nmap.org** (1) in **Packet bytes** (2).

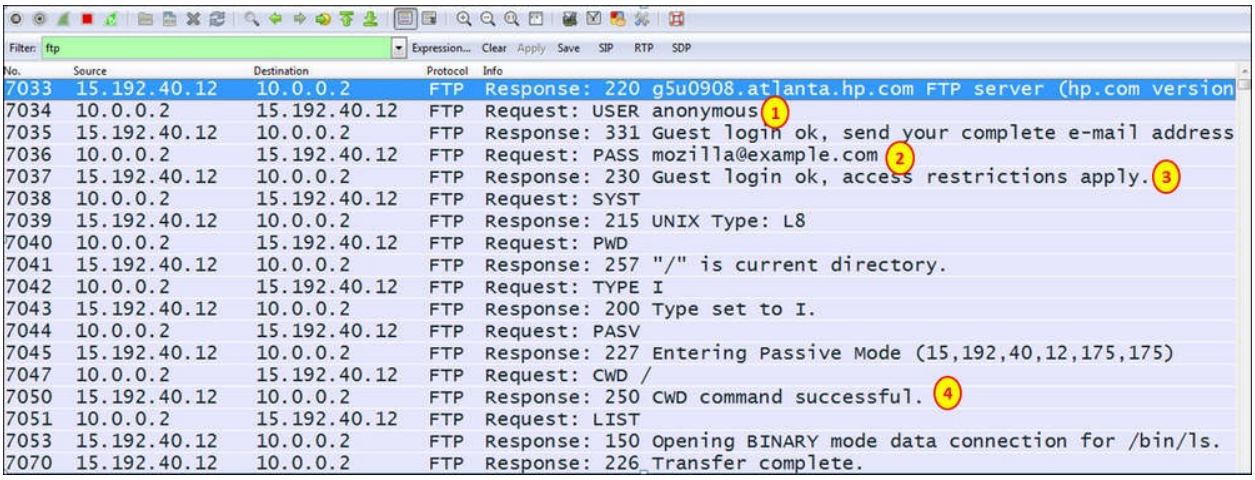


And this is what we got:



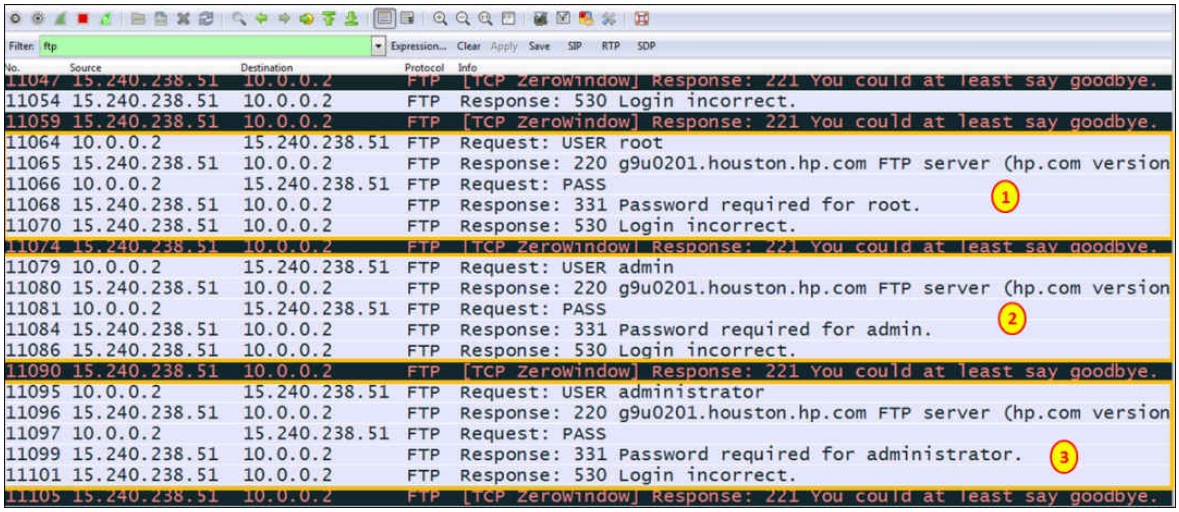
Another important issue is brute force attack, that is, when the attacker tries to guess the password in order to break into a server.

In the following screenshot, you'll see what happens when an attacker tries to break into a well-protected FTP server.



No.	Source	Destination	Protocol	Info
7033	15.192.40.12	10.0.0.2	FTP	Response: 220 g5u0908.atlanta.hp.com FTP server (hp.com version
7034	10.0.0.2	15.192.40.12	FTP	Request: USER anonymous (1)
7035	15.192.40.12	10.0.0.2	FTP	Response: 331 Guest login ok, send your complete e-mail address
7036	10.0.0.2	15.192.40.12	FTP	Request: PASS mozilla@example.com (2)
7037	15.192.40.12	10.0.0.2	FTP	Response: 230 Guest login ok, access restrictions apply. (3)
7038	10.0.0.2	15.192.40.12	FTP	Request: SYST
7039	15.192.40.12	10.0.0.2	FTP	Response: 215 UNIX Type: L8
7040	10.0.0.2	15.192.40.12	FTP	Request: PWD
7041	15.192.40.12	10.0.0.2	FTP	Response: 257 "/" is current directory.
7042	10.0.0.2	15.192.40.12	FTP	Request: TYPE I
7043	15.192.40.12	10.0.0.2	FTP	Response: 200 Type set to I.
7044	10.0.0.2	15.192.40.12	FTP	Request: PASV
7045	15.192.40.12	10.0.0.2	FTP	Response: 227 Entering Passive Mode (15,192,40,12,175,175)
7047	10.0.0.2	15.192.40.12	FTP	Request: CWD /
7050	15.192.40.12	10.0.0.2	FTP	Response: 250 CWD command successful. (4)
7051	10.0.0.2	15.192.40.12	FTP	Request: LIST
7053	15.192.40.12	10.0.0.2	FTP	Response: 150 Opening BINARY mode data connection for /bin/ls.
7070	15.192.40.12	10.0.0.2	FTP	Response: 226 Transfer complete.

1. Since it is FTP, the first trial is with username `anonymous` (1), a password chosen by the attacker (2), login is, of course, approved (3), and the attacker gets in (4).
2. In the following screenshot, you see what happens when the attacker tries other usernames that are not authorized.



No.	Source	Destination	Protocol	Info
11047	15.240.238.51	10.0.0.2	FTP	[TCP ZeroWindow] Response: 221 You could at least say goodbye.
11054	15.240.238.51	10.0.0.2	FTP	Response: 530 Login incorrect.
11059	15.240.238.51	10.0.0.2	FTP	[TCP ZeroWindow] Response: 221 You could at least say goodbye.
11064	10.0.0.2	15.240.238.51	FTP	Request: USER root
11065	15.240.238.51	10.0.0.2	FTP	Response: 220 g9u0201.houston.hp.com FTP server (hp.com version
11066	10.0.0.2	15.240.238.51	FTP	Request: PASS
11068	15.240.238.51	10.0.0.2	FTP	Response: 331 Password required for root. (1)
11070	15.240.238.51	10.0.0.2	FTP	Response: 530 Login incorrect.
11074	15.240.238.51	10.0.0.2	FTP	[TCP ZeroWindow] Response: 221 You could at least say goodbye.
11079	10.0.0.2	15.240.238.51	FTP	Request: USER admin
11080	15.240.238.51	10.0.0.2	FTP	Response: 220 g9u0201.houston.hp.com FTP server (hp.com version
11081	10.0.0.2	15.240.238.51	FTP	Request: PASS
11084	15.240.238.51	10.0.0.2	FTP	Response: 331 Password required for admin. (2)
11086	15.240.238.51	10.0.0.2	FTP	Response: 530 Login incorrect.
11090	15.240.238.51	10.0.0.2	FTP	[TCP ZeroWindow] Response: 221 You could at least say goodbye.
11095	10.0.0.2	15.240.238.51	FTP	Request: USER administrator
11096	15.240.238.51	10.0.0.2	FTP	Response: 220 g9u0201.houston.hp.com FTP server (hp.com version
11097	10.0.0.2	15.240.238.51	FTP	Request: PASS
11099	15.240.238.51	10.0.0.2	FTP	Response: 331 Password required for administrator. (3)
11101	15.240.238.51	10.0.0.2	FTP	Response: 530 Login incorrect.
11105	15.240.238.51	10.0.0.2	FTP	[TCP ZeroWindow] Response: 221 You could at least say goodbye.

3. Here, you can see that the attacker is trying to login with the usernames **root** (1), **admin** (2) and **administrator** (3).
4. The attacker is blocked, and the server sends a **TCP Zero-Window**

message and even answers by saying **you could at least say goodbye**.

Appendix A. Links, Tools, and Reading

In this appendix I would like to refer to some useful links from which you can get further information about Wireshark: learning sources, additional software, and so on.

Useful Wireshark links

The main Wireshark link is of course http://www.wireshark.org. Here you can find:

- The downloads page at http://www.wireshark.org/download.html.
- The learning page at http://www.wireshark.org/docs/.
- And what is called the **Enhancement** area at http://www.riverbed.com/products-solutions/products/performance-management/wireshark-enhancement-products/. This is a link to Riverbed, who acquired CACE Technologies, the primary sponsor of Wireshark, and became the main sponsor; they now develop and sell commercial add-ons.
- As open source software, Wireshark development resources are located under http://www.wireshark.org/docs/wsdg_html_chunked/ or http://www.wireshark.org/develop.html

Also some other useful links are:

- The Wireshark graphical user interface was created using GTK+, or the GIMP toolkit, an open source kit that can be found on the GTK+ project web page at http://www.gtk.org/
- For WinPCap (the Windows capture driver), go to www.winpcap.org, and for remote monitoring go to http://www.winpcap.org/docs/docs_40_2/html/group__remote.html#Config

tcpdump

tcpdump is free Unix-based software that runs under the Unix/Linux command line. Some of the useful resources for it are:

- The tcpdump website: http://www.tcpdump.org/
- The Windows version of tcpdump (Windump): http://www.winpcap.org/windump/default.htm
- A friendly tutorial: http://danielmiessler.com/study/tcpdump/
- The official man page: http://www.tcpdump.org/tcpdump_man.html

Wireshark can open tcpdump files, so when you capture packets with tcpdump, you can later open it with Wireshark or any other graphical tool.

Some additional tools

Although Wireshark is by far the most common network analysis tool on the market, there are also many other network troubleshooting tools that I use a lot. Before getting into the details, I would like to go back some years to one of the funniest network problems I've ever had. The case itself was very simple, but it comes with an important lesson. It had to do with a network in a warehouse of a big hospital. The warehouse workers were equipped with wireless terminals, taking medication as needed and conveying it to the various departments of the hospital. The problem was that all the terminals worked very slowly. They called an integration company to help them with the problem, and these guys came in with every piece of troubleshooting equipment ever made. They came with Wireshark, Sniffer, wireless analyzers, spectrum analyzers, and many other boxes. I went there, and when I saw what they were doing, I told them that they forgot to bring one important thing, their heads. If they had used them, they would have discovered that the problem was a bad RJ45 cable from the warehouse to the hospital's main network 50 meters from there.

The conclusion is very simple of course. Tools are just tools. Without the knowledge of networking and where to use them, they will not help you. In this section I would like to bring in some additional tools, and where to use them.

Note

What I bring here, along with other examples in the book, are devices and software tools that I've worked with over the years. Some of them are freeware and some are commercial products. It is important to note that their descriptions come from my own experience. I don't have a commercial or any other interest in any of them.

SNMP tools

The first sets of tools that I usually use to solve a problem are SNMP tools. There are tools with strong mapping capabilities, there are some with good statistical capabilities, and there are some with good logging and events capabilities.

First, in order to just monitor SNMP counters, you can use simple free MIB browsers and graphical tools such as:

Vendor	Software name	Where to download	Notes	License
Manage engine	MibBrowser	http://www.manageengine.com/products/mibbrowser-free-tool/	Very friendly with minimal configuration.	Free
Open source	MRTG	http://oss.oetiker.ch/mrtg/	Requires time and knowledge to install and configure. Good for long-term statistics. Commonly used by ISPs as a console for their customers.	Free wit up to 10 sensors (*1); Comme from 11 sensors
SolarWinds	Network device monitor	http://www.solarwinds.com/products/freetools/network-device-monitor/	Solarwinds is one of the leaders in network management tools, and along with the commercial stuff, you can find many free tools.	Free

SolarWinds Engineering toolset	Engineer's Toolset	http://www.solarwinds.com/engineers-toolset.aspx	Various tools for network monitoring, discovery, SNMP, configuration, basic scanners and more.	Free wit limited capabili Comme with full capabili

SNMP platforms

SNMP platform are pieces of software that provide a central console that shows a map of the network, collects information and presents statistical reports, and collects SNMP events and presents them by severity and other parameters.

Some of the common tools in this category are:

Vendor	Software name	Where to download	Notes
Castlerock Computing	SNMPc	http://www.castlerock.com/	This is one of the friendliest SNMP tools that I have worked with for more than a decade. The SNMP management platform is very easy to use and is great for network debugging.
SolarWinds	Assorted	http://www.solarwinds.com/network-management-software.aspx	SolarWinds has various tools that provide monitoring, mapping, configuratio management and other network management capabilities. These are some of the best options available bu

			are expensive.
Manageengine	Assorted	http://www.manageengine.com/network-performance-management.html	Various tools that provide monitoring, mapping, configuration management and other network management capabilities. One of the best but expensive.
HP	IMC, NNM, and so on	http://h17007.www1.hp.com/us/en/networking/solutions/network-management/index.aspx#.UkgqGT8YhyI	This is a great platform. HP made it much friendlier than previous Network Node Manager (NNM) software. It is definitely worth checking out
OpenNMS	OpenNMS	http://www.opennms.org/	It is open source but requires know-how o how to configure it.
Nagious	Nagious	http://www.nagios.org/	It is open source but requires a knowledge to configure it.

There are many others tools, such as:

- The open source Cacti (http://www.cacti.net/)
- Zabbix (http://www.zabbix.com/)
- MRTG (http://oss.oetiker.ch/mrtg/)
- Some others (some under the GNU public license).

There are the "heavyweight" suites, such as:

- The HP OpenView suite of management applications (http://www8.hp.com/us/en/software-solutions/software.html?compURI=1174702#tab=TAB1)
- CA Unicenter (http://www.ca.com/us/network-performance-management.aspx)

There are also other medium-sized platforms, various tools from Plixer (http://www.plixer.com/), and many others.

For network monitoring and troubleshooting you will need the very basic tools, while as a platform you will need a more sophisticated one. You can find a nice comparison of management platform on http://en.wikipedia.org/wiki/Comparison_of_network_monitoring_systems.

The NetFlow, JFlow, and SFlow analyzers

NetFlow from **Cisco** (www.cisco.com/go/netflow) and JFlow from **Juniper** (http://www.juniper.net/techpubs/software/erx/junose82/swconfig-ip-services/html/ip-jflow-stats-config2.html) provide a method for collecting TCP/IP traffic flow statistics on your routing devices.

SFlow (http://en.wikipedia.org/wiki/Sflow and http://www.sflow.org/index.php) is an industry standard technology for monitoring high-speed switched networks.

The differences between them are:

- **NetFlow** applies to Cisco routers and L3 switches. In layer-3 switches make sure that they support NetFlow (depends on software version and hardware). In some cases, you will need additional software/hardware for this. It was standardized by RFC3954 (http://www.ietf.org/rfc/rfc3954.txt).
- **JFlow** applies to Juniper routers and L3 switches.
- **SFlow** is a standard for monitoring LAN switches and was standardized by RFC3176 (http://tools.ietf.org/html/rfc3176).
- **IPFIX** (RFCs 5101 and 5102) is a standard developed from **NetFlow** v9, and standardized by the **IETF**.

All Flow/IPFIX technologies are based on the communications device that collects the flow data from the interfaces and sends them to the management station. They require a simple configuration on the router or switch and software to collect the data and present it.

This software can be used for monitoring which users are causing a load on the network (displayed according to IP addresses or DNS names), on which applications (HTTP, SMTP, and so on, displayed according to their port numbers), web pages (displayed according to their IP addresses, translated to DNS names), and other such criteria. While Wireshark is usually used for this purpose in short-term monitoring (the **Conversations** feature), these tools can be used for long-term monitoring as well.

Some common software options include:

- http://www.plixer.com/Scrutinizer-Netflow-Sflow/scrutinizer-flow-analyzer.html from **Plixer**

- http://www.sevone.com/technologies/NetFlow-analysis from **SevOne**

There are freeware tools, and there are commercial tools with free limited capabilities versions (usually limited by the number of interfaces they can monitor); in commercial SNMP platforms, you usually have a free license for two to five interfaces.

HTTP debuggers

HTTP debuggers are tools that provide statistical and detailed data about HTTP. Here are some tools for this:

Vendor	Software name	Where to download	Notes
Eric Lawrence and Telerik	Fiddler	http://fiddler2.com/	The most common freeware HTTP debugging tool, this works as a separate software that captures packets and analyzes them (such as Wireshark).
Simtec Limited	HTTPWatch	http://www.httpwatch.com/	This is available in basic limited and commercial editions. Available as an add-on to Firefox or Internet Explorer. Files can be opened with HTTP Watch Studio. Available also for iPhone iOS.

What you will get with these tools is HTTP statistical and performance information, for example, how much time it took to open a web page, the reasons for delays, and error summaries.

Syslog

Syslog (https://tools.ietf.org/html/rfc5424) is a protocol for message logging. There are many parameters on communication devices that can be configured, so in cases where a problem occurs, a message will be sent to the Syslog server. These are usually hardware- and- software- based problems that are not always covered by SNMP.

A great Syslog server (that receives the messages and presents them) can be found at http://www.kiwisyslog.com/free-edition.aspx. There are many other tools, and they are available for free in many management platforms.

Other stuff

Some other tools you might need to get for working with networks are:

- A neat tool, **Xplico**, for extracting application data contained in capture files: http://www.xplico.org/about (freeware)
- **Nmap** security scanner: http://nmap.org/
- **Netcat** (nc) for Linux: http://nc110.sourceforge.net/

Network analysers

While Wireshark is by far the most common network analysis tool, there are also some other tools that can be used in times of trouble. Some of them are:

- **Riverbed Cascade suite of tools**: This is developed by the Wireshark guys and provides a graphical analysis of Wireshark files. It can be found at http://www.riverbed.com/products-solutions/products/performance-management/
- **WildPackets OmniPeak**: I've used this in some cases for heavily loaded networks, which my laptop with Wireshark couldn't handle. It has great statistics tools and works well under heavy loads. It can be found at http://www.wildpackets.com/products/omnipeek_network_analyzer.

There are probably more, but these are the two I've worked with and both do a great job.

There is a simple analysis tool in some Cisco devices that comes as a part of the IOS. Cisco calls it **Mini Protocol Analyzer**, and you can find it at https://www.cisco.com/en/US/docs/routers/7600/ios/12.2SR/configuration/guide/

Interesting websites

Here are some interesting websites that I use a lot:

- First and most useful is http://www.cisco.com, from where you can learn the technologies along with how to configure them in Cisco
- Many Wireshark files, exercises, and challenges can be found at http://www.honeynet.org/
- A summary table of Wireshark filters can be found at http://packetlife.net/media/library/13/Wireshark_Display_Filters.pdf
- Captured and filed examples can be found at http://wiki.wireshark.org/SampleCaptures
- Another interesting Wireshark filter page is http://www.packetlevel.ch/html/tcpdumpf.html
- Some Perl stuff can be found at http://perldoc.perl.org/perlre.html, and Perl regular expressions can be found at http://perldoc.perl.org/perlre.html#Regular-Expressions and http://www.regular-expressions.info/perl.html

Books

Here is a list of some of the books I've used over the years:

- To understand TCP/IP basics: *TCP/IP Illustrated, Volume 1: The Protocols (Second Edition)* (by *Addison-Wesley Professional Computing Series*)
- A comprehensive, illustrated internet protocol reference, free on the Internet: *The TCP/IP Guide* (http://www.tcpipguide.com/)
- Many books from Cisco press are worth reading, both the technology and certifications books: http://www.ciscopress.com/
- Cisco design guides: Just Google the subject you're looking for

Part 3. Module 3

Mastering Wireshark

Analyze data network like a professional by mastering Wireshark - From 0 to 1337

Chapter 1. Welcome to the World of Packet Analysis with Wireshark

This chapter provides you an introduction to the basics of the TCP/IP model and familiarizes you with the GUI of Wireshark along with a sample packet capture. You will be introduced to the following topics:

- What is Wireshark?
- How does it work?
- A brief overview of the TCP/IP model
- An introduction to packet analysis
- Why use Wireshark?
- Understanding the GUI of Wireshark
- The first packet capture

Introduction to Wireshark

Wireshark is one of the most advanced packet capturing software, which makes the life of system/network administrators easy and proves its usefulness among the groups of security evangelists. Wireshark is also called a protocol analyzer, which helps IT professionals in debugging network-level problems. This tool can be of great use to optimize network performance.

Wireshark runs around dissecting network-level packets and showing packet details to concerned users as per their requirement. If you are one of those who deals with packet-level networking everyday, then Wireshark is for you and can be used for multiple troubleshooting purposes.

A brief overview of the TCP/IP model

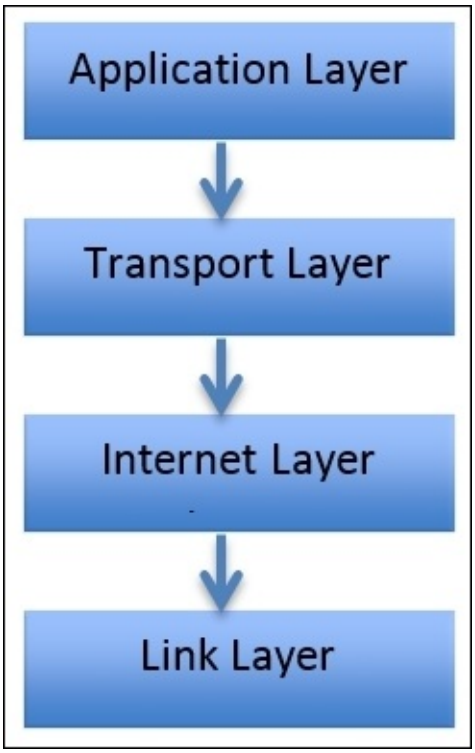
Next, it's time to discuss the most important topic in the world of networking. In order to understand how all these things stick together, we need to understand the basics of the TCP/IP model. Even the world of computers needs a set of rules and regulations to communicate, and this is taken care by the networking protocols, which govern the transmission of packets/segments/frames over a dedicated channel between hosts.

The TCP/IP model was originally known as the DoD model, and the project was regulated by United States Department of Defense. The TCP/IP model takes care of every aspect of every packet's life cycle, namely, how a packet is generated, how a single packet gets attached with a required set of information (PDU), how a packet is transmitted, how it comes to life, how it is routed through to intermediary nodes to the destination, how it is integrated back with other packets to get the whole information out, and so on.

If you have any confusion regarding the basics of networking protocols, I would recommend that you do a quick revision before proceeding ahead, as this book requires familiarity with the TCP/UDP protocols. By the time you come back, you will be able to visualize and answer all of these questions on your own.

The layers in the TCP/IP model

The TCP/IP model comprises four layers, as shown in the following diagram. Each layer uses a different set of protocols allocated to it. Every protocol has specific designated roles, and all of them are designed in such a way that they comply with industry standards.



The first layer is the **Application Layer** that directly interacts with users and other network-level protocols; it is primarily concerned with the representation of the data in an understandable format to the user. The Application layer also keeps track of user web sessions, which users are connected, and uses a set of protocols, which helps the application layer interface to the other layers in the TCP/IP model. Some popular protocols that we will cover in this book are as follows:

- **The Hyper Text Transfer Protocol** (**HTTP**)
- **The File Transfer Protocol** (**FTP**)
- **The Simple Network Management Protocol** (**SNMP**)

- **The Simple Mail Transfer Protocol** (**SMTP**)

The second layer is the **Transport Layer**. The sole purpose of this layer is to create sockets over which the two hosts can communicate (you might already know about the importance of network sockets) which is essential to create an individual connection between two devices.

There can be more than one connection between two hosts at the same instance. IP addresses and port numbers together make this possible. An IP address is required when we talk about WAN-based communication (in LAN-based communication, the actual data transfer happens over MAC addresses), and these days, a single system can communicate with more than one device over multiple channels which is possible with the help of port numbers. Apart from the restricted range of port numbers, every system is free to designate a random port for their communication.

This layer also serves as a backbone to the communication between two hosts. The most common protocols that work in this layer are TCP and UDP, which are explained as follows:

- **TCP**: This is a connection-oriented protocol, often called a reliable protocol. Here, firstly, a dedicated channel is created between two hosts and then data is transferred. Then, the sender sends equally partitioned chunks, over the dedicated channel, and then, the receiver sends the acknowledgement for every chunk received. Most commonly, the sender waits for a particular time after which it sends the same chunk again for assurance. For example, if you are downloading something, TCP is the one that takes care and makes sure that every bit is transferred successfully.
- **UDP**: This is a connection-less protocol and is often termed an unreliable form of communication. It is simple though because there is no dedicated channel created, and the sender is just concerned with sending chunks of data to the destination, whether it is received or not. This form of communication actually does not hamper the communication quality; the sole purpose of transferring the bits from a sender to receiver is fulfilled. For example, if you are playing a LAN-based game, the loss of a few bytes is not going to disrupt your gaming experience, and as a result, the user experience is not harmed.

The third layer is the **Internet Layer**, which is concerned with the back and forth movement of data. The primary protocol that works is the **IP** (**Internet Protocol**) protocol, and it is the most important protocol of this layer. The IP provides the routing functionality due to which a certain packet can get to it's destination. Other protocols included in this layer are ICMP and IGMP.

The last layer is the **Link Layer** (often termed as the Network Interface Layer) that is close to the network hardware. There are no protocols specified in this layer by TCP/IP; however, several protocols are implemented, such as **Address Resolution Protocol** (**ARP**) and **Point to Point** (**PPP**). This layer is concerned with how a bit of information travels inside the real wires. It establishes and terminates the connection and also converts signals from analog to digital and vice versa. Devices such as bridges and switches operate in this layer.

The combination of an IP address and a MAC address for both the client and server is the core of the communication process, where the IP address is assigned to the device by the gateway or assigned statically, and the MAC address comes from the **Network Interface Card** (**NIC**), which should be present in every device that communicates with other hosts. As data progresses from the Application layer to the Link Layer, several bits of information are attached to the data bits in the form of headers or footers, which allow different layers of the TCP/IP model to coordinate with each other. The process of adding these extra bits is called data encapsulation, and in this process, a **Protocol data unit** (**PDU**) is created at the end of the networking model.

It consists of the information being sent along with the different protocol information that gets attached as part of the header or footer. By the time PDU reaches the bottom-most layer, it is embedded with all the required information required for the real transfer. Once it reaches the destination, the embedded header and footer PDU elements are ripped off one by one as it passes through each and every layer of the TCP/IP model as it progresses upward in the model.

The following figure depicts the process of encapsulation:

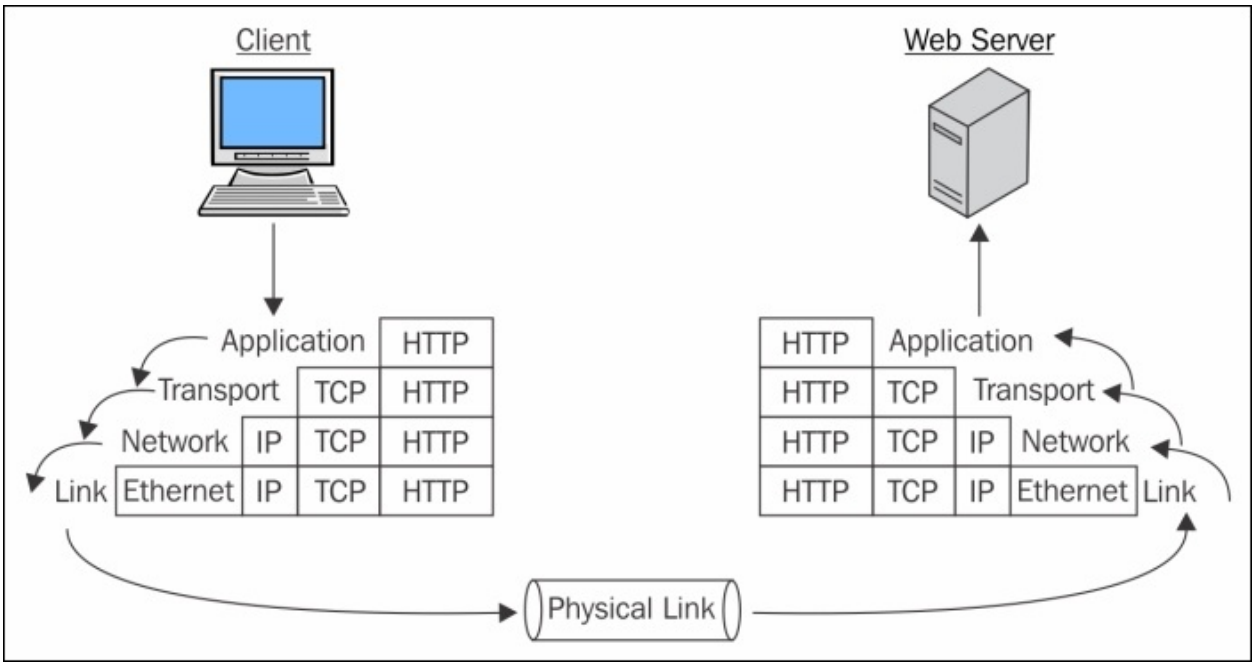


Figure 1.1: Data encapsulation

An introduction to packet analysis with Wireshark

Packet analysis (also known as packet sniffing or protocol analyzing) is used to intercept and capture live data as it travels over the network (Ethernet or Wi-Fi) in order to understand what is happening in the network. Packet analysis is done by protocol analyzers such as Wireshark available on the Internet. Some of these are free and some are paid for commercial use. In this book, we will use Wireshark to perform network analysis, which is an open source software and the best free-network analyzer available on the Internet.

Numerous problems can happen in today's world of networking; for this, we need to be geared up all the time with the latest set of tools that can avail us of the ease of troubleshooting in any situation. Each of these problems will start from the packet level and can gradually grow up to a high network downtime. Even the best of protocols and services running on a system can go bad and behave maliciously. To get to the root of the problem, we need to look into the packet level to understand it better. If you need to maintain your network, then you definitely need to look into the packet level. Packet analysis can be used for the following aspects:

- To analyze network problems by looking into the packets and their specific details so that you can get a better hold over your network.
- To detect network intrusion attempts and whether there are any malicious users who are trying to get into your network, or they have already got access to something in your network.
- To detect network misuse by internal or external users by establishing firewall rules in your security appliance and then monitoring each of these rules through Wireshark.
- To isolate exploited systems so that the affected system doesn't become a pivot point for your network for malicious users.
- To monitor data in motion once it travels live in your network to have better control over the allowed and restricted categories of data. For instance, say you want to create a rule for your firewall that will block the access to Bit Torrent sites. Blocking access to them can be done from your manageable router, but knowing from where the request was originated can be easily

audited through Wireshark.

- To gather and report network statistics by filtering the most specific packets as per your requirements and then creating specific capture filters for your perusal that can help you in the long run.
- Learning who is on the network and what they are doing, is there something they are not allowed to do, and is there anyone who is trying to bypass the network restrictions. All of these simple day-to-day tasks can be achieved easily through Wireshark.
- To debug client/server communications so that all the request and replies communicated between the peers on our network can be audited to maintain the integrity of your network.
- To look for applications that are sitting in the corner of your own network and eating the bandwidth. They might be making your network insecure or making it visible to the public network. Through this unnoticed application, different forms of network traffic can enter without any restrictions.
- To debug network protocol implementations and any kind of anomalies present due to various misconfigurations in the current running devices.

To identify possible or malicious attacks that your network can be a victim of, to analyze them, control/supervise them, and make yourself ready for any possible malicious activity.

When performing a packet analysis, you should take care of things such as which protocols can be interpreted, which is the best software you can use according to your expertise, which protocol analyzer will best suit your network requirement. Experience does count in this field; once you start working with Wireshark, gradually you will come up with new ideas to troubleshoot and analyze your packets in a much more advanced way.

Packet sniffers can interpret common network protocols (such as IP and ICMP), transport layers (such as TCP and UDP), and application protocols (such as DNS and HTTP).

Due to the overwhelming amount of information presented by Wireshark's GUI, it might seem complex to some users and might be considered as one of its demerits. There are a few CUI/GUI tools that can solve this purpose. They are pretty simple to use and also present a simpler interface, for example, TShark, tcpdump, Fiddler, and so on.

How to do packet analysis

When traffic is captured, either all raw data is captured or only the header data is captured without capturing the total content of the packet. Captured information is decoded from raw data to a human-readable form, which allows users to understand the exchanged data between the networks in a much more precise manner.

What is Wireshark?

Wireshark is a packet-sniffing software that is used by IT professionals all around the world for analysis purpose. You can download it for free from https://www.wireshark.org/download.html.

Wireshark can be installed on a variety of platforms, including Linux, MAC, and Windows (most of the versions). This is open source software, which means that the code of the software and its required libraries can be downloaded from the same website we mentioned earlier.

One of the important key aspects of packet sniffing is where to place the packet sniffer in the physical network to achieve the maximum utilization out of it; packet sniffing is often referred to as tapping into the wire.

Tapping into the wire is not just about starting Wireshark on your system; there are a couple of things a person should know about before starting the sniffer. For instance, placing the sniffer at a proper place in the organization's infrastructure, having working knowledge of different networking devices because each of the networking devices (hubs, switches, routers, and firewalls) behave differently. It is also important to know how each of them work and how network devices handle network traffic. Placing the sniffer in the right place can impact your packet analyzing experience in a detailed manner, which in the end can lead to drastic results if done correctly.

After you have placed your sniffer, you should confirm that your NIC supports promiscuous working. By enabling this, your interface card will start learning about even those packets that are not destined or routed through your machine. A network's broadcasted traffic can be captured and analyzed by every client, which is part of the same network. Network devices broadcast multiple types of traffic that can be listened to by an interface, which supports the promiscuous mode.

The ARP protocol's traffic is broadcasted. The address resolution protocol is responsible for resolving MAC to IP addresses and vice versa. Devices such as switches send an ARP packet to all devices asking for the correct device to respond with it's MAC address. Gradually, the switch will maintain a list of

MAC addresses and their corresponding IP addresses, which is even termed as the CAM table (content addressable memory). Now, whenever any host wants to communicate with its other corresponding peers over the LAN, information required for the transfer is communicated to the sender from the switch. Information such as IP and MAC addresses for different devices can be easily captured and recorded through ARP traffic.

How it works

Wireshark comes with the libcap/Winpcap driver, which lets you switch your NIC to the promiscuous mode; the only time you don't want to sniff in the promiscuous mode is when the packets are directly, intentionally destined to your device. On a Windows-based system, you should have elevated administrator privileges to sniff and analyze the packets. There are three common step processes that every protocol analyzer follows: collect, convert, and analyze. These are described as follows:

- **Collect**: This is the first step where you choose a certain interface to listen on, and through this, you can acquire a certain amount of raw data from the network, which can be achieved by switching your interface into a promiscuous mode so that, after capturing what ever traffic is being broadcasted in your network, it can be displayed in your Wireshark GUI.
- **Convert**: This is to increase the readability of the collected binary form. Network packets can be converted by the protocol analyzer, such as Wireshark, to simple and easier formats so that people like us can have a better understanding of packets and solve our day-to-day problems easily.
- **Analyze**: In this final step, after the collection and conversion of the network packets, a step-by-step process of analyzing the data starts where we look into the specific details about the protocols and their specific configuration details. Then, we move on to host and destination addresses and the kind of information they are sharing. Rest of the analysis is left to the user's consent and how they filter and review the collected data.

If you want to get a foothold on understanding the process of packet capturing and analysis, you really need to be well versed with networking protocols and how they work because the whole communication that happens over a network is governed by various protocols, such as ARP, **Dynamic Host Control Protocol** (**DHCP**), **Domain Name Service** (**DNS**), **Transmission Control Protocol** (**TCP**), **Internet Protocol** (**IP**), HTTP, and many others.

Protocols are the rules and regulations that govern the process of communication between two network devices and control the environment under which they operate. Each of these protocols has different complexity levels depending on how and where they are being implemented. Majorly, all protocols work in the

same fashion, where they send a request and wait for the confirmation, and as they receive an acknowledgement, they let the devices communicate.

After the data has been successfully transferred between them, the connections should be terminated gracefully in order to mark a communication as successful without loss of even a single bit. While the data is transferred, protocols need to maintain the integrity of the communication as well, that is, if `abc` information is sent from the sender's side, it should be received in the same order and manner. If the bits are being tampered during the transition, this means that the protocol used isn't reliable. Analyzing all of these tasks is the basic work responsibility of any network protocol analyzer.

Capturing methodologies

Network packets can be captured through various techniques. Depending on the requirement, a protocol analyzer is placed at a certain place in network with a particular type of configuration.

Hub-based networks

Hub-based networks are the easiest ones to sniff out because you've the freedom to place the sniffer at any place you want, as hubs broadcast each and every packet to the entire network they are a part of. So, we don't have to worry about the placement. However, hubs have one weakness that can drastically decrease network performance due to the collision of packets. Because hubs do not have any priority-based system for device that send packets, whoever wants to send them can just initiate the connection with the **HUB** (central device) and start transmitting the packets. Often, more than one devices start sending packets at the same instance. Now, as a result, the collision of the packets will happen, and the sending side will be informed to resend the previous packet. As a consequence, things such as traffic congestion and improper bandwidth utilization can be experienced.

The switched environment

Due to some restrictions present in switched-based infrastructures, packet analysis becomes a bit complex. To bypass these restrictions and make the life of administrators easy, we will talk about a couple of solutions such as port mirroring and hubbing out.

In **port mirroring**, once you have the command-line configuration console or web-based interface to mage you're the access point (router/switch), then we can easily configure port mirroring.

Let's make it simpler for you with a logical illustration. For instance, let's assume that we have a 24-ports switch and 8 PCs which (PC-1 to PC-8) are connected. We are still left with more than 15 ports. Place your sniffer in any of those free ports and then configure port mirroring, which will copy all the traffic from whatever device we want to the port of our choice, where our protocol analyzer sits, which can see the whole bunch of data traveling through the mirrored port.

Once this is completely configured, we will be able to easily analyze each and every piece of information going back and forth from the mirrored port. This technique is one of the easiest among others to configure; the only thing you should know beforehand is how to configure switches with command-line interfaces. These days, admins are provided with a GUI for configuration purposes if it is the case for you to just go for it. The following figure depicts a simple demonstration of port mirroring:

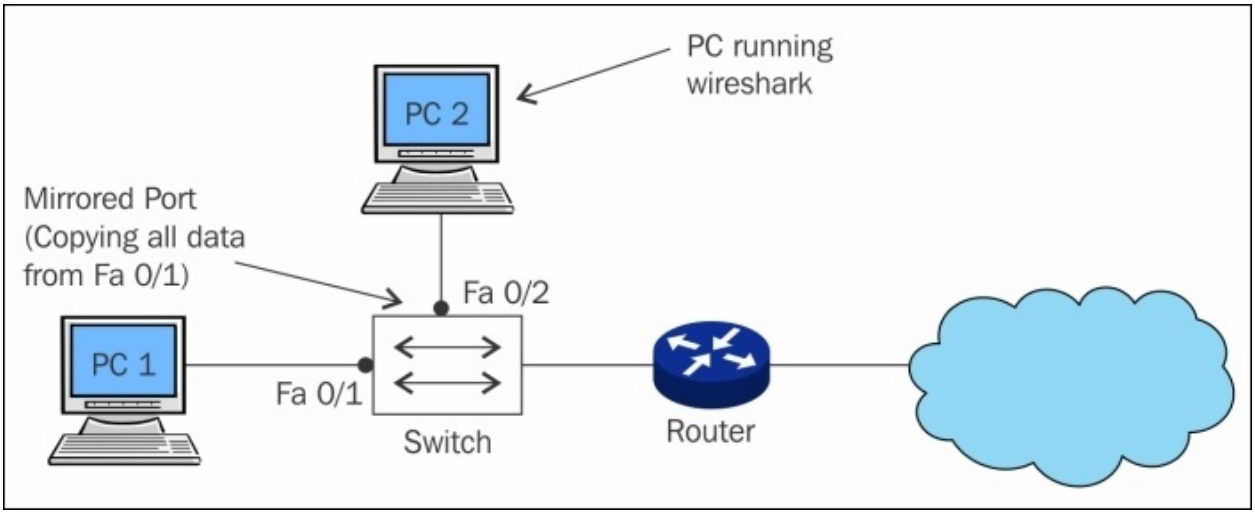


Figure 1.2: Port mirroring

Hubbing out is feasible when your switch doesn't support port mirroring. To use the technique, you have to actually plug the target PC out of the switched network, then plug your hub to the switch, and then connect you analyzer and target device to the switch so that becomes the part of the same network.

Now, the protocol analyzer and the target are part of the same broadcast domain. Your analyzer will easily capture every packet destined to target or originated from the target. But make sure that the target is aware about the data loss that can happen while you try to create hubbing out for analysis. The following figure will make it easier for us to understand the concept precisely:

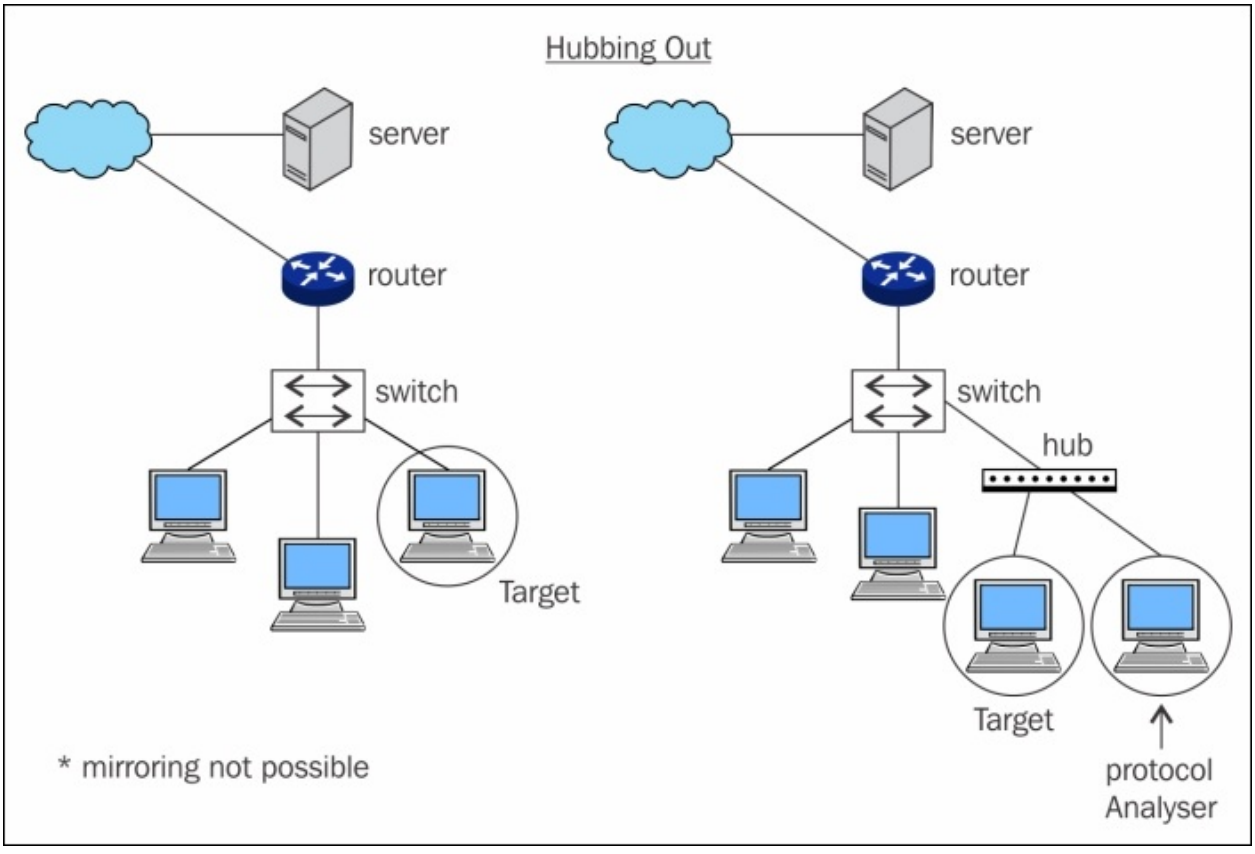


Figure 1.3: Hubbing out

ARP poisoning

This is an unethical way to capture network traffic where we try to imitate another device between two parties. Let's say, for example, we have our default gateway at `192.168.1.1` and our client is located at `192.168.1.2`. Both of these devices must have maintained a local ARP cache that facilitates them to send packets without any extra overhead over the LAN. Now, the question is what kind information does the ARP cache hold, and in which form. Let me tell you, the command to view the ARP cache, which displays MAC addresses associated for a particular IP address is `arp -a` . Issuing the `arp -a` command (the same works for most of the platforms) populates a table that holds a device's IP address and its MAC address. Have a look at the following diagram which shows a normal scenario of ARP poisoning:

Before ARP Cache

192.68.1.1 – (Server)
192.68.1.2 – AA:BB:EE
192.68.1.3 – AA:BB:DD

192.68.1.2 – (Client)
192.68.1.1 – AA:BB:CC
192.68.1.3 – AA:BB:DD

192.68.1.3 – (Attacker)
192.68.1.1 – AA:BB:CC
192.68.1.2 – AA:BB:EE

Now that we've understood what is stored inside an ARP cache, let's try to poison it.

After ARP Cache

192.68.1.1 – (Server)
192.68.1.2 – AA:BB:DD
192.68.1.3 – AA:BB:DD

192.68.1.2 – (Client)
192.68.1.1 – AA:BB:DD
192.68.1.3 – AA:BB:DD

192.68.1.3 – (Attacker)
192.68.1.1 – AA:BB:CC

192.68.1.2 – AA:BB:EE

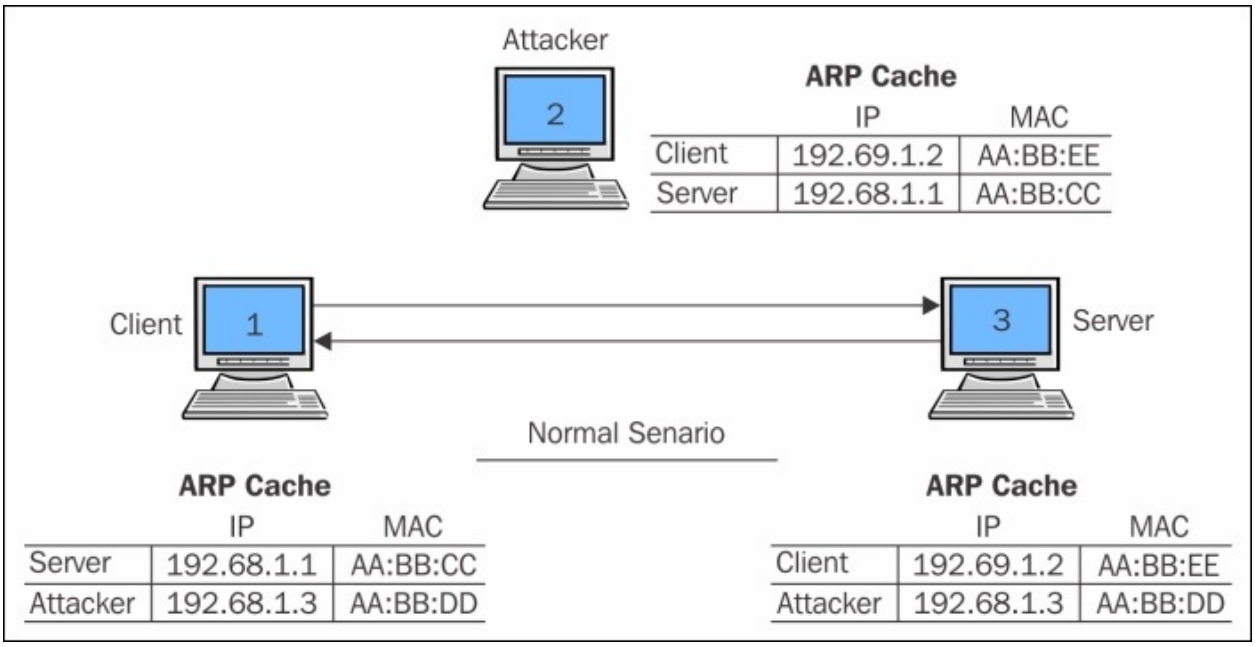


Figure 1.4: ARP poisoning (the normal scenario)

Now that you've understood what is the importance of the ARP protocol and how it works, we can try to poison the `arp` cache of both the default gateway and the client with the attacker's MAC address. In simple terms, we will replace the client's MAC address in the default gateway's ARP cache with the attacker's MAC address. We will do the same in the client's MAC address, replacing the default gateway's MAC address with the attacker's MAC address. As a result, every packet destined to the client from the default gateway and vice versa will be sent to the attacker's machine.

If port forwarding is already configured on the attacker's side, the received packet will be forwarded to the real intended destination, without giving any hints to the client and the default gateway that the packet is being sniffed.

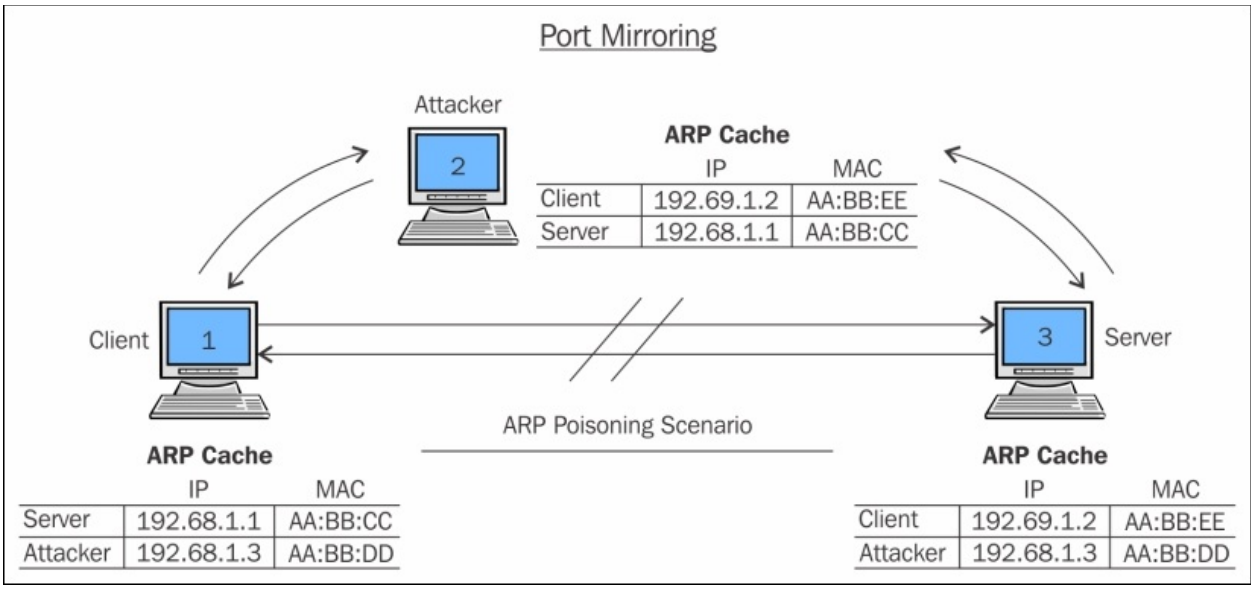


Figure 1.5: ARP poisoning (the poisoned scenario)

Other than these two techniques, there is a variety of hardware available on the market, which are popularly known as taps and can be placed between any two devices to sniff and analyze the traffic. Though this technique is effective to capture network traffic in some scenarios, it should be practised or deployed in a controlled environment because it can prove to be malicious to the internal corporate network.

Passing through routers

When dealing with routed environments, the main aspect of packet analyses is to place your sniffer at the right place from where we can gather the required information. Dealing with routed structures demands more skills, as sometimes you need to rethink about the placement of your sniffer. Consider a routed environment with three routers:

Router 1, router 2, and router 3 are working together; each of them owns 2-3 PCs. Router 1 is the acting like a root node while controlling its child networked nodes (router 2 and router 3). Router 3 clients are not able to connect to router 1 clients. To resolve this issue, the admin of the organization has placed the sniffer inside the router 3 area.

After a while, the admin has collected quite a good amount of packets; the admin is still not able to detect the anomaly within the network. So, he/she decides to move the sniffer to another area in the network. After placing the sniffer in the router 1 area, the admin can see quite a useful stream of packets that he/she was looking for earlier. This is quite a simple illustration of moving the sniffer around, which can be helpful in certain situations. The moral is that placing the sniffer in your networked infrastructure is quite an important task.

After reading this, I hope you would now like to see how Wireshark actually looks like, so let's take a look at the GUI of the software and how we have to initialize the process of capturing network packets.

If you do not have Wireshark installed, you can get a free copy from https://www.wireshark.org/download.html. To go through the illustrations in this book, you also need to be familiar with the interface.

Why use Wireshark?

I hope I am not the only one who is obsessed with the simplicity of the packet capturing scenario, which Wireshark facilitates for us. I will just quickly point out the reasons why most people prefer Wireshark to other packet sniffers:

- **User friendly**: It does count for every GUI we have ever seen or worked with, how easily the options are presented, and how convenient it is to use (I guess, even the ones who don't know about packet analysis can start capturing packets in Wireshark without any prior specialized knowledge).
- **Robustness**: The amount of information Wireshark can handle is outstanding; what I actually mean by this is software of this kind may hang or crash (because of thousands of packets that are captured and displayed every second) when trying to display the packets traveling all over the network. However, Wireshark doesn't—a big hand to Wireshark creators for how well they have structured it.
- **Platform independent**: Yeah, this one is definitely on the list. This free software can be installed on any platform that is used for computing purposes by administrators these days, whether Linux-based, Windows-based, or Macintosh-based platforms.
- **Filters**: There are two kinds of filtering options present in Wireshark:
    - You choose what to capture (capture filters)
    - You choose what to display after you've captured (display filters)
- **Cost**: Wireshark comes free, and is developed and maintained by a dedicated community. Wireshark offers some paid professional tools also. For more details refer to Wireshark's official website.
- **Support**: Wireshark is being developed very actively by a group of contributors scattered around the globe . We can sign up to the Wireshark's mailing list or we can get help from the online documentations, which can be accessed through the GUI itself; and various online forums are available to get the most effective; go to Google *paid Wireshark support* to know more about it.

The Wireshark GUI

Before we discuss its awesome features, let me take this opportunity to explain the history of Wireshark and how it came into existence.

Wireshark was built during the late '90s. Combs, a young college graduate from Kansas city developed Ethereal (the basic version of Wireshark), and by the time Combs developed this awesome piece of invention, he had landed himself a job where he signed a formal contract. After a few years of service, Combs decided to quit his job and to pursue his dreams by developing Ethereal further. Unfortunately, as per the legal terms, the Combs invention was part of the company's proprietary software. Despite this, Combs left the job and started working on the new version of Ethereal, which he titled Wireshark. Since 2006, Wireshark has been in active development and is being used worldwide. It supports a majority of protocols (more than 800), which are implemented in the wild today.

The installation process

Follow these steps to install Wireshark on your system:

1. In this book, I am going to you use a Mac PC; for other platforms, the installation is the same. Some OSes, such as Kali Linux, come with a preinstalled version of Wireshark.
2. So, if you are using Macintosh, then first and foremost, you need to download X11 Quartz (XQuartz-2.7.7), which will simulate an environment to run Wireshark (for Windows just download the respective executable compatible with your processor).
3. Now, you can install Wireshark (Wireshark 1.12.6 Intel 64), which we downloaded earlier in this book.
4. Once both of these are successfully installed, we need to restart our computer.
5. After the PC has been restarted, start Wireshark. As soon as the packet analyzer opens, you will see that the X11 server starts on its own. You don't need to worry about it; just leave it in the background.
6. Once it is opened completely, it will look as shown in the following screenshot:

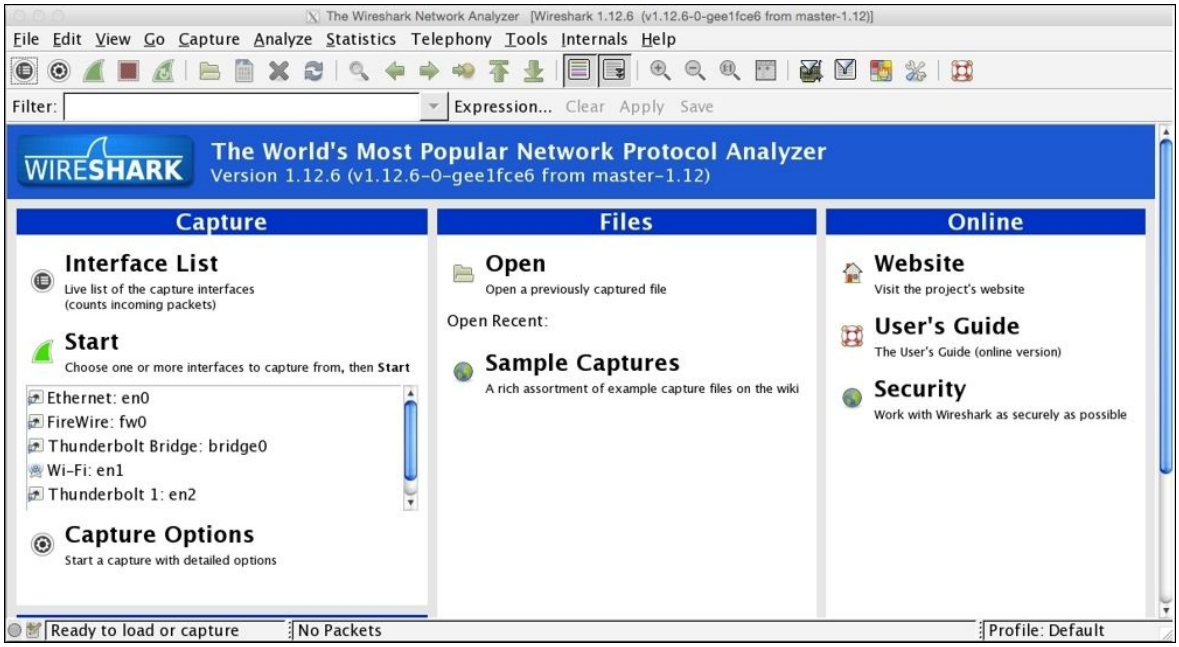


Figure 1.6: The Wireshark screen

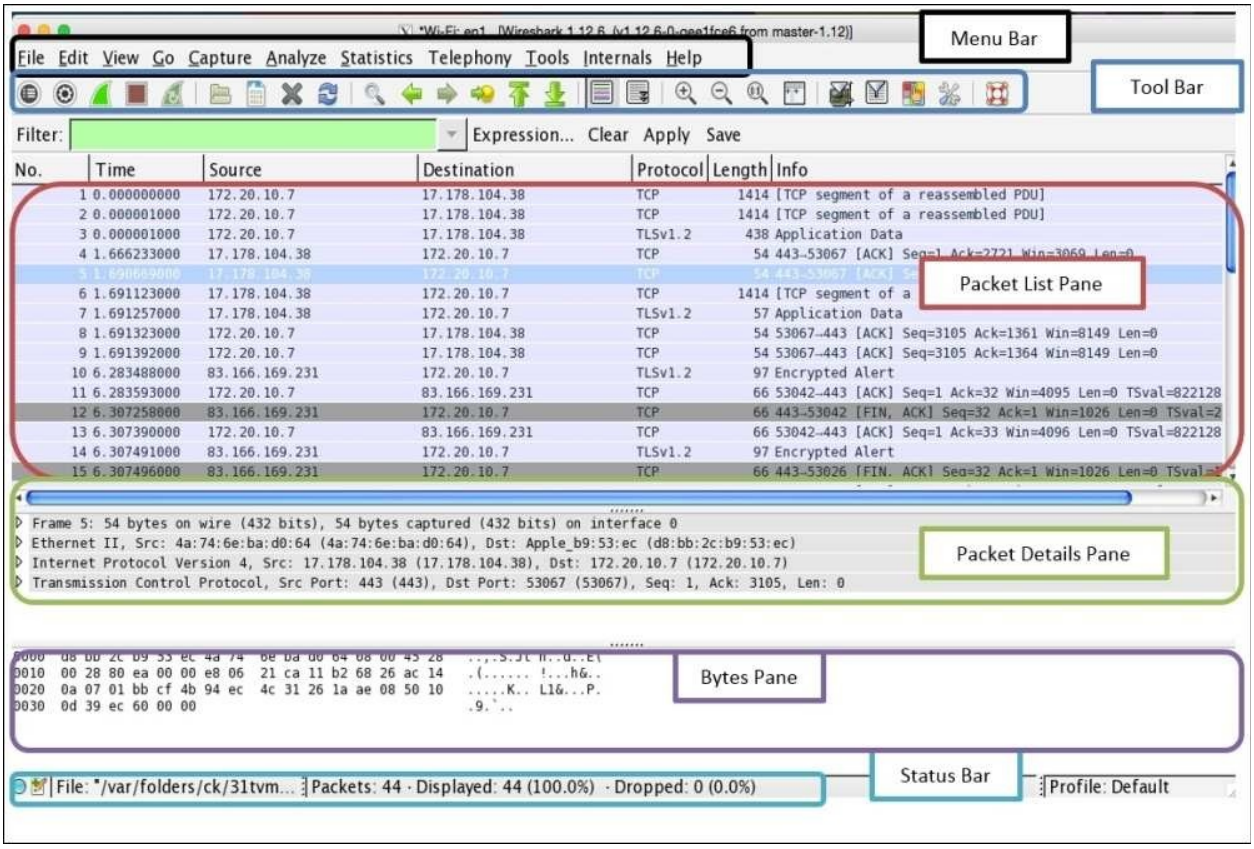
Before we go ahead and start the first capture, we need to get a bit familiar with the options and menus available.

There are six main parts in the Wireshark GUI, which are explained as follows:

- **Menu Bar**: This represents tools in a generalized form that are organized in the **Applications** menu.
- **Main Tool Bar**: This consists of the frequently used tools that can offer efficient utilization of the software.
- **Packet List Pane**: This window area displays all the various packets getting captured by Wireshark.
- **Packet Details Pane**: This window gives us details pertaining to the selected packet in the packet list pane are shown. For example, we can view source and destination IP addresses and different protocols used for communication arranged in the bottom-top approach (Link Layer to Application Layer). Information regarding the packets is listed in different categories of protocols that can be expanded to get more details for the selected packet.

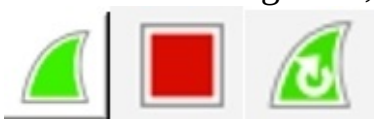
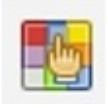
- **Bytes Pane**: This shows the data in the packets in the form of hex bytes and their corresponding ASCII values; it shows the values in the form in which they travel in the wires.
- **Status Bar**: This displays details such as total packets captured.

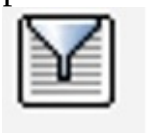
The following screenshot will help you to identify different sections in the application, please make sure you get yourself acquainted with all of them before proceeding to further chapters.



Within the toolbar area, we have a few useful tools. I would like to give you a brief overview of some of them:

-  : This gives you the option to choose an interface for listening

-  : Through this, you can customize the capturing process
-  : These are to start/stop/restart the capturing process
-  : This is to open a saved capture file
-  :This is to save the current capture in a file
-  : This is to reload the current capture file
-  : This is to close the current capture file
-  : This is to go back to the recent most visited packet
-  : This icon is to go forward to the most recently visited packet
-  : This is used to go to a specific packet number
-  : Toggle Color coding for the packets On/Off
-  This is used to toggle the autoscroll on/off
-  : This is to zoom in, zoom out, and reset zoom to the default
-  : This is used to change the color coding as per requirements

-  : This is used to narrow down the window in order to capture packets
-  : This is used to configure display filters to only see what is required

Even after selecting a working interface, sometimes, you won't be able to see any packets in your packet list pane. There can be multiple reasons for this, some of which are listed as follows:

- You do not have any network traffic
- The packets traveling in the network are not destined to your device
- You do not have the promiscuous mode activated or do not have an option for the promiscuous mode

After launching the Wireshark application, you will see something like the following screenshot on our screens. Although it doesn't look so interesting at first glance, what makes it interesting are the packets that are flowing around. Yeah, I am talking about capturing packets.

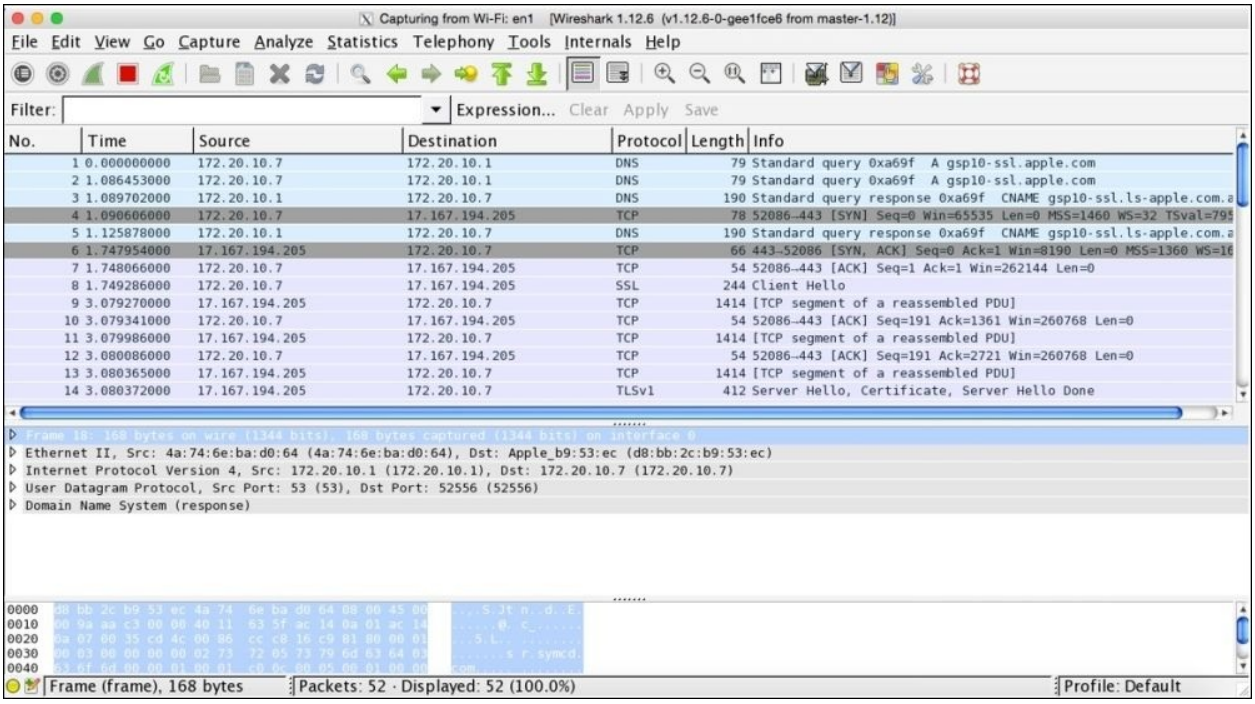


Figure 1.7: The Wireshark capture screen

Starting our first capture

As you've been introduced to the basics of Wireshark and since you have learned how to install Wireshark, I feel you are ready to initiate your first capture. I will be guiding you through the following series of steps to start/stop/save you first Wireshark capture:

1. Open the Wireshark application.
2. Choose an interface to listen to.

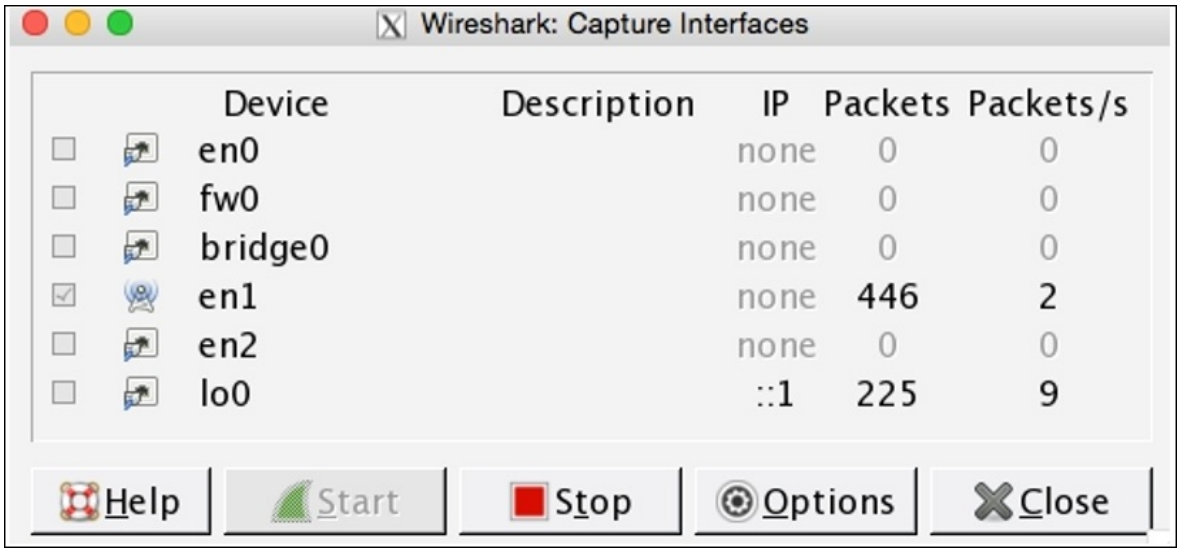


Figure 1.8: The interface window

3. Before you click on **Start**, we have the **Options** button, which gives us the advantage of customizing the capture process; but as of now, we will be using the default configuration.

   ### Tip

   Make sure that the **Promiscuous** mode is activated so that we can capture the traffic that is not destined to our machine.

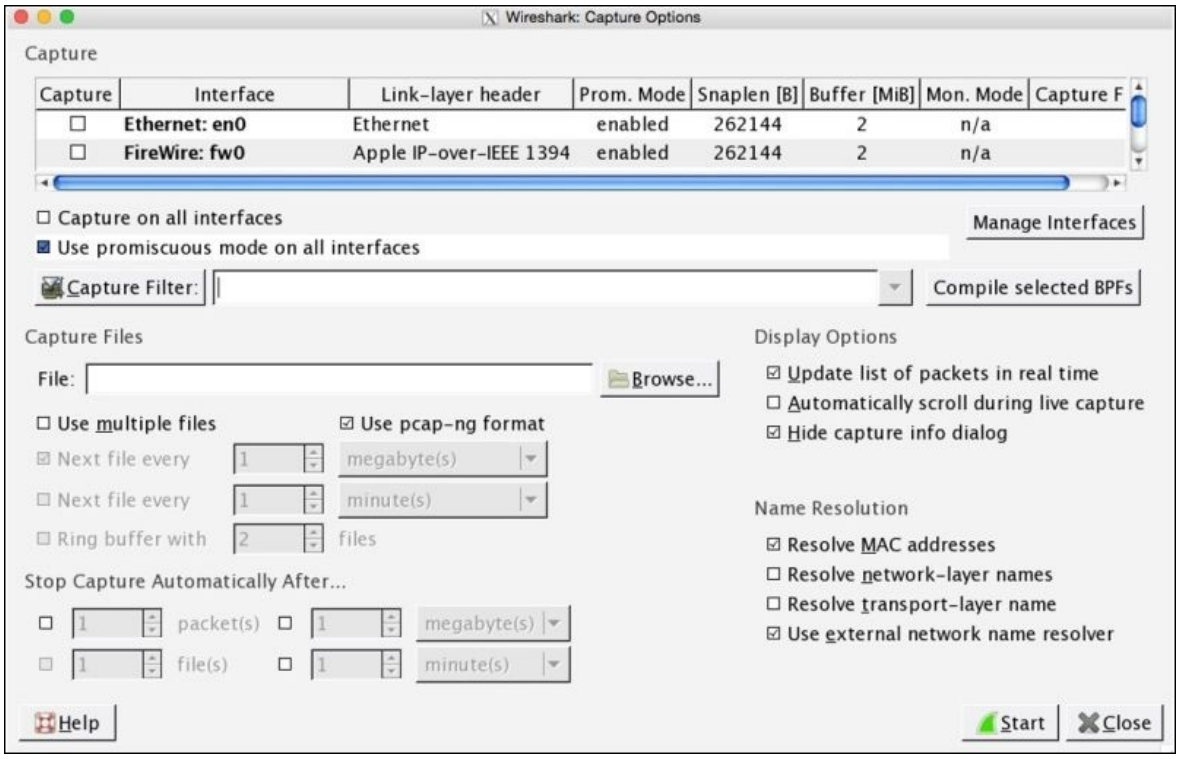


Figure 1.9: The capture customization screen

4. Click on the **Start** button to initiate the capturing process.
5. Open your browser.
6. Visit any website you want to.

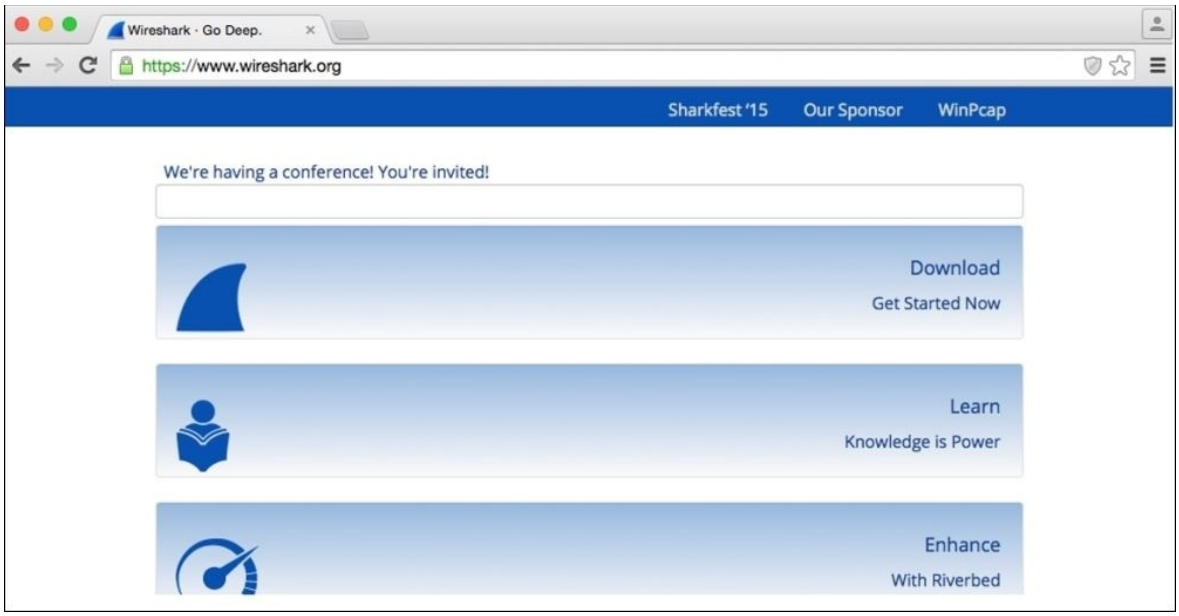


Figure 1.10: The Wireshark website

7. Switch back to the Wireshark screen; if everything goes well, you should be able to see a numerous packets getting captured in your Wireshark GUI inside the packet list pane.

   To stop the capture, you can just click on the **stop capture** button in the toolbar area or you can click on **Stop** under the **Capture** menu bar.

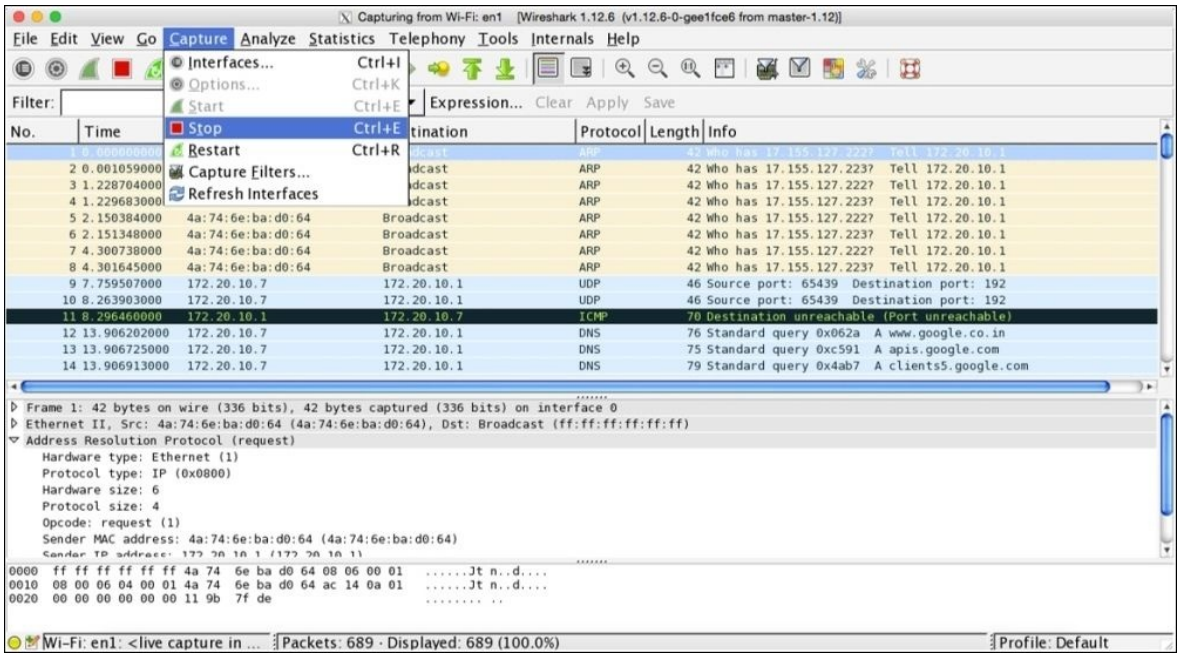
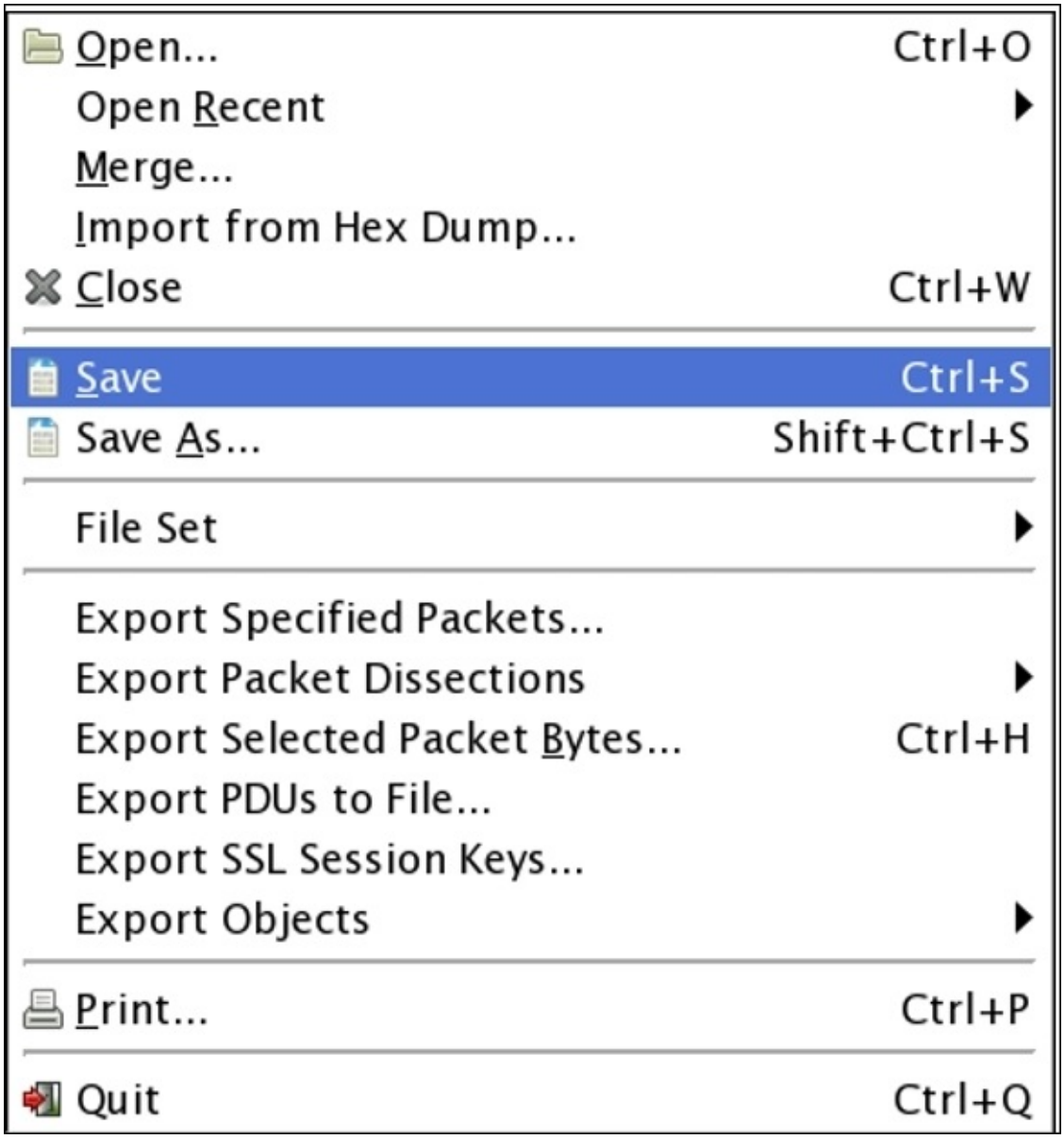


Figure 1.11: Stopping capture

8. I know there is an overwhelming amount of information you will see by now, but don't worry about it. I am here to make it simple for you.
9. The real process of packet analysis starts when you have captured packets—I mean packet filtering. We will be discussing packet filtering in detail in the upcoming chapters.
10. Now, the last step is to save the capture file for later use:



- Save your file with the default `.pcapng` extension in you folder.

If you have read all the steps all the way up to this point, I would encourage you to create your first capture file.

Summary

This chapter lays the foundation of basic networking concepts along with an introduction of the Wireshark GUI. Wireshark is a protocol analyzer that is used worldwide by IT professionals to capture and analyze network-level packets.

The TCP/IP model has four layers: the Application Layer, Transport Layer, Network Layer, and Link Layer. Data gets encapsulated as it passes on from one layer to another; the resulting packet at the bottom is called a complete PDU, which actually travels over the channel.

To install Wireshark, you just need to visit http://www.wireshark.org and then download the appropriate version of this open source software. The Wireshark community is governed by real-world geeks; this can be a good source of learning and for troubleshooting purposes.

The Wireshark GUI is user friendly, robust, and platform independent; even new IT professionals can easily adapt the tool.

One important aspect of protocol analyzing is to place the sniffer at the right place; every organization's infrastructure is different from another, where we might need to apply different techniques in order to get the right packets to use.

Hubbing out, port mirroring, ARP poisoning, and tapping are some of those useful techniques that can be used to monitor and analyze traffic in different situations.

There are six main parts in the Wireshark tool window: **Menu Bar**, **Main Tool Bar**, **Packet List Pane**, **Packet Details Pane**, **Bytes Pane**, and **Status Bar**.

Using the back/forward key during a packet analysis scenario can be really useful. One should know about all the tools that are displayed in the main toolbar area.

In the next chapter, you will learn how to work with different kinds of filters available in Wireshark.

Practice questions

Q.1 How many layers are there in the TCP/IP? Name them.

Q.2 Which layer in the TCP/IP model handles Layer 2 addresses?

Q.3 The Link Layer is also called?

Q.4 The HTTP protocol uses TCP or UDP?

Q.5 IP, ICMP, and _____ are the protocols in the Internet Layer Q.6 How many parts of the Wireshark window do you know?

Q.7 ARP is a Layer 3 protocol—true/false?

Q.8 Does the TCP protocol follow a three-way handshake?

Q.9 The Port Mirroring technique is possible through switches only—True/False?

Q.10 The Hubbing out technique uses a router to isolate a PC from it peers—true/false?

Q.11 TCP is an unreliable protocol—true/false?

Q.12 Install Wireshark and start a sample capture using your wireless interface. Save your capture file on the desktop with the name `first.pcap`, and close Wireshark.

Q.13 Open your `first.pcap` capture file in Wireshark and check how many packets you captured in total.

Q.14 Which pane displays information in the HEX and ASCII form for each packet we've captured?

Q.15 Switch off the promiscuous mode from the capture options window and observe whether you are still able to receive packets from other devices or not.

Chapter 2. Filtering Our Way in Wireshark

This chapter will talk about different filtering options available in Wireshark, namely, capture and display filters. We will also look at how to create and use different profiles. The following are the topics we will cover in this chapter:

- An introduction to capture filters
- Why and how to use capture filters
- Lab up—capture filters
- An introduction to display filters
- Why and how to use display filters
- Lab up—display filters
- Colorizing traffic
- Creating a new Wireshark profile(s)
- Lab up—profiles

I hope you are ready to start analyzing packets using different filtering options present in Wireshark and to reuse the filters that we previously created in a user-defined profile. I will be guiding you with a technique to filter packets based on certain expressions, which we will create using different primitives that are available.

Before we go ahead and start creating awesome filters, I want to mention one more interesting tool that is used to find packets: the find utility.

An introduction to filters

In the world of Wireshark, there are two kinds of filters that can be used over live traffic, and on saved capture files. Filters enhance the flexibility of packet analysis, where a certain user is given the privilege of seeing what he/she wants to see to capture what they want to capture.

The two types of filters are capture filter and display filter. Now, let's have look at each one of them in detail.

Capture filters

This gives you the facility to capture what you want to capture—others will be discarded. Capturing packets is a processor-intensive task, and Wireshark will acquire a quite good amount of primary memory as well. So, sometimes, we will have to save the resources for other processes, which can be utilized to analyze packets, and in some cases, we would like to capture only that data which meets our expression—rest of it will be dropped.

Wireshark offers some interesting options to configure an interface, which will be capturing traffic that meets only a certain expression, and this is achievable through the **Capture Options** window, as shown in the following screenshot:

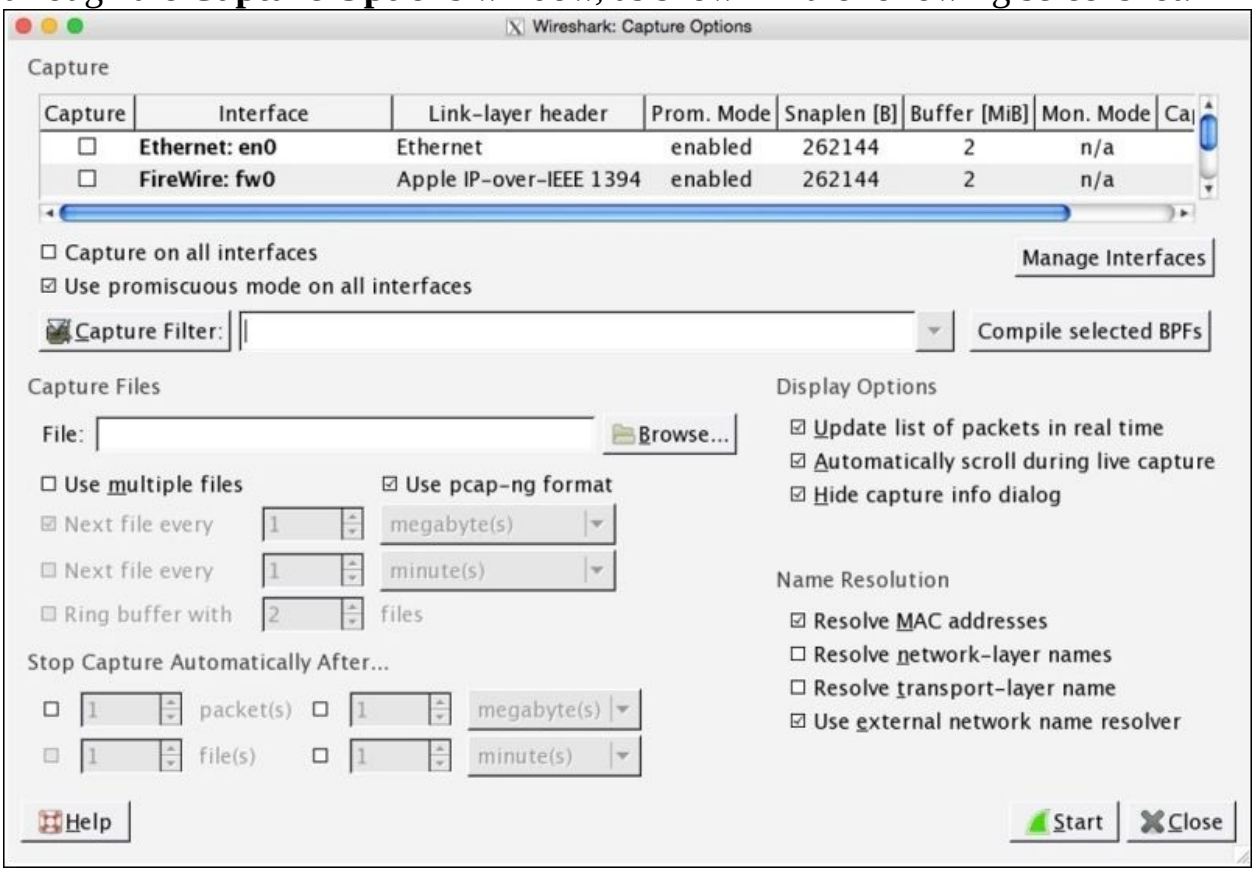


Figure 2.1: The Capture Options dialog

Here, points list various capture options dialog related details

- **Capture**: In this window, you can choose the interface you want to capture packets from, and you can even select multiple interfaces at once to listen on all of them. The details for every interface are listed under separate columns such as **Capture**, **Interface**, the name of the interface, whether the promiscuous mode is enabled or not, and so on. Under the **Capture** dialog, you will see a checkbox to toggle the promiscuous mode, and you can even choose the **promiscuous on all interfaces** option to activate what you require in just one click.
- **Manage Interfaces**: This button facilitates addition or removal of a new interface for listening purposes you intend to. You can add even remote machine interfaces, where you would be required to have root level privileges.
    - **Capture Filter**: By clicking on this **Capture Filter** button, you will be able to see a dialog similar to what is shown here. The already configured capture filters are listed by default, and here, we can create and save our custom capture filters as well.

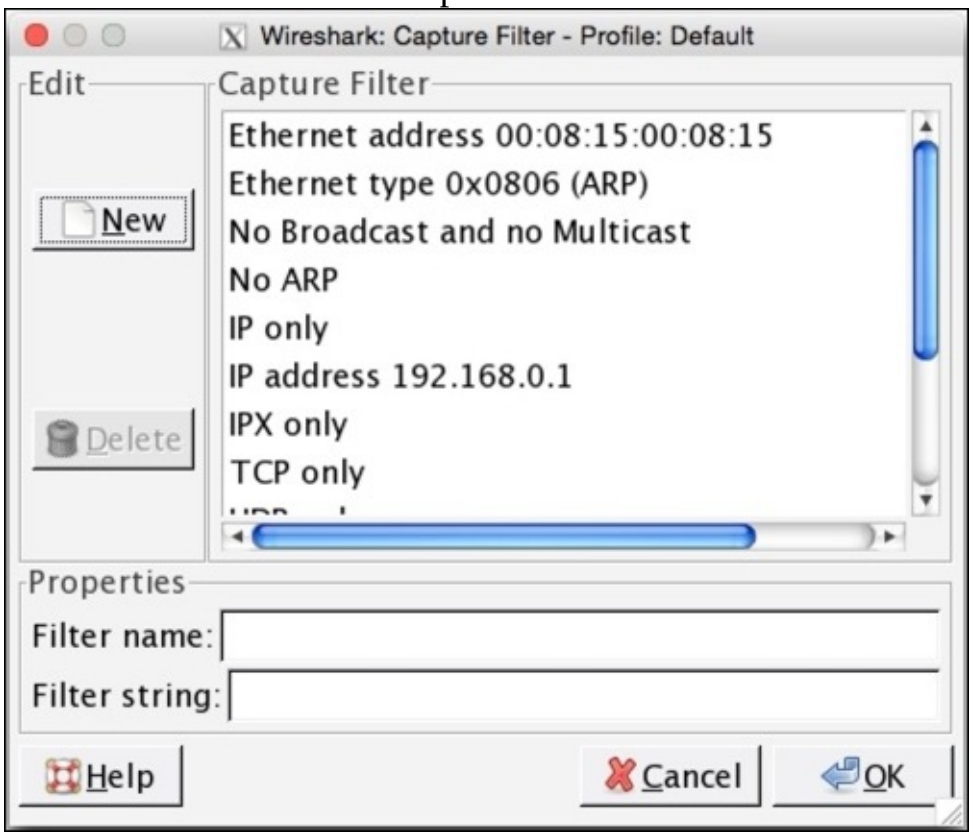
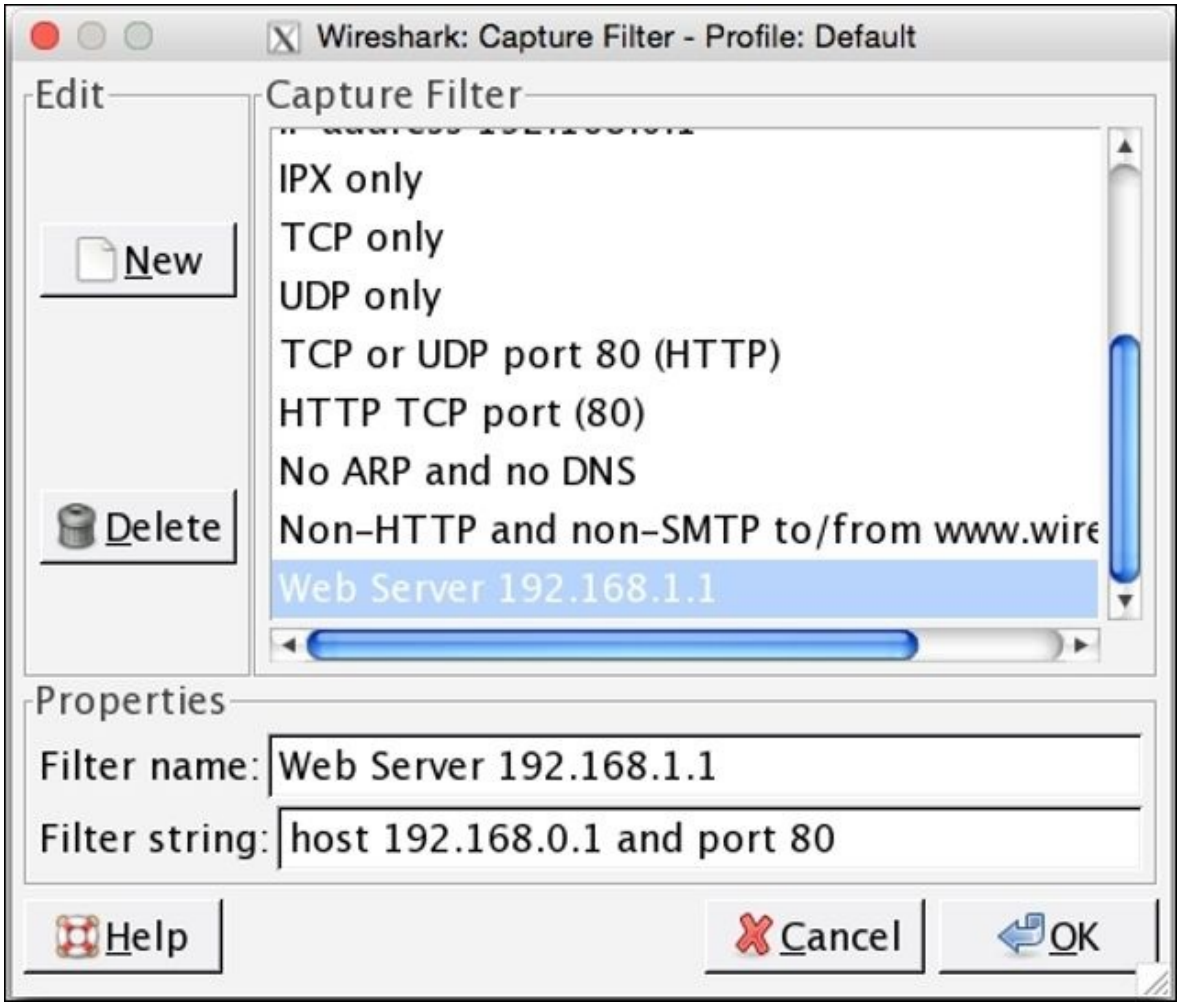


Figure 2.2 :Default Capture filters

To start off, users can use these default filtering profiles and get an idea about how to create custom filtering strings. Once you are well versed with the basics, you can go ahead and use the same window to create your own custom filters, but make sure that you have followed the **Berkley Packet Filtering** (**BPF**) syntax. The BPF syntax is an industry standard and is used by multiple protocol analyzers, which make your filter's configuration file portable.

Let's create one together to get a better hold over it; consider a scenario where we have to capture packets originating from a web server that is located at `192.168.1.1` (change the IP address to the web server's address that you are monitoring), and follow the next steps:

1. Open the **Capture Options** dialog.
2. Click on **Capture Filter**.
3. Click on **New**.
4. Write `Web server 192.168.1.1` inside the **Filter name** textbox.
5. Write `host 192.168.1.1 and port 80` inside the Filter String text-box



6. Once you've done this, click on **OK**; if you've entered everything correctly, the textbox followed by the **Capture Filter** button will be displayed with a green background, as shown in the following screenshot:

Figure 2.4 :Creating a sample capture filter

- **Capture Files**: This option gives you the flexibility to save your captured packets into the file(s) that already exists on your system. The captured packets will be added to the file of your choice if you don't choose any. A

temporary file will be created, and data will be written to it, which can be saved to a user-specified location. To achieve this, write the name of the file that uses absolute path referencing or click on **Browse** followed by the **File** textbox to choose a location.

If you select the multiple files option, then you can save your packets in multiple files, where we can customize more options, which are stated as follows:

- **Next File Every**: After capturing a certain amount of data, Wireshark will create a new file and your data will be added to it. For instance, I want to create a new file after Wireshark captures 2 MBs of data.
- **Next File Every**: After a certain amount of time, Wireshark will create a new file and your packets will be added to it. For instance, I want to create a new file after every 5 minutes of the capturing process.
- **Ring buffer**: Using this option, you can restrict the creation of a new file. Wireshark uses the **First in First Out** (**FIFO**) option to write data to multiple filesets. For example, you have selected the **Ring buffer** option and increased the number of files to `5`, and you have configured that after every 5 MBs, a new file should be created.

Now, according to this configuration, once you start capturing packets, after every 5 MBs of data, a new file will be created and the packets will be written to it. Once the limit that you specified in the **Ring Buffer** area is exceeded, Wireshark will not create a new file; instead, it will roll back to the first file and append data to it. The following screenshot shows a similar kind of configuration:

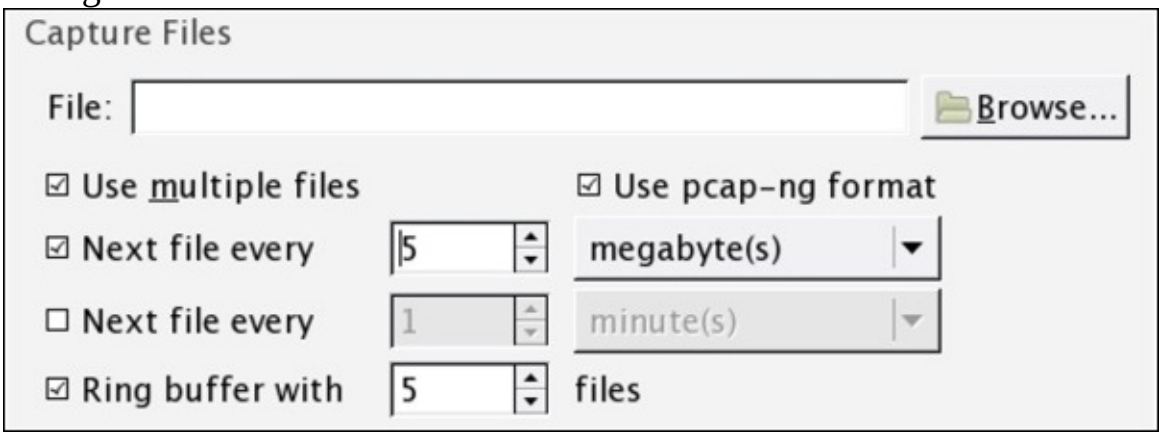


Figure 2.5 : The Capture Files option

- **Stop Capture Settings**: This option lets you stop the capturing process after a certain condition is triggered; we have four different kinds of triggers. Activating these can stop Wireshark from capturing new packets, and they are stated as follows:
  - **Packet(s)**: Stop capturing after a certain count of packets is reached
  - **File(s)**: Stop capturing after the creation of a certain number of files
  - **Megabyte(s)**: Stop capturing after capturing a certain amount of data
  - **Minute(s)**: Stop capturing after running for a certain period of time

There might be one question that you may want to ask: what if we select more than one option at a time? For instance, as shown in the following figure.

You can activate more than one option at a time; Wireshark will stop capturing whichever condition is met first.

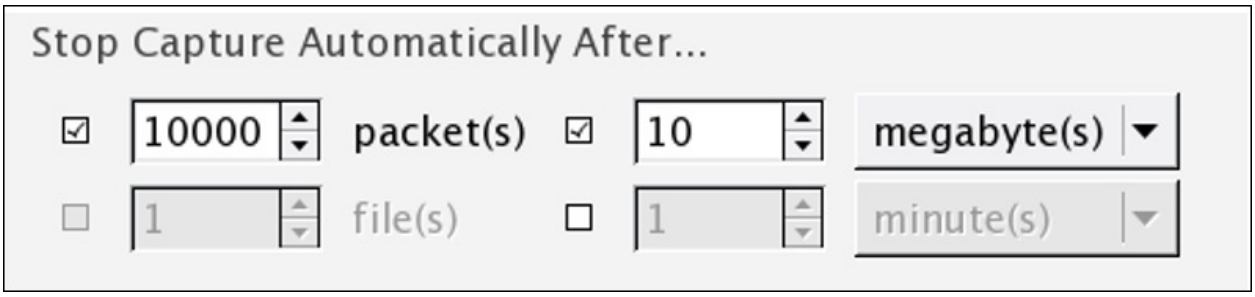


Figure 2.6 : The Stop Capture options

- **Display Options**: There are a few options available in this section that can be configured to restrict how the packets and their corresponding information will be displayed in the **Packet List Pane** option and the **Protocol hierarchy** window. Refer to the following figure to see this.

If you select **Update list of packets in real-time**, you will observe that **Packet List Pane** is updated as soon as Wireshark captures a new packet, and the pane will be scrolled upwards automatically. Choose these options if needed; otherwise, the resources acquired by these two tasks can be used for other processes.

If you check the **Hide capture info dialog** box, the **Protocol Hierarchy** window, that shows the statistics (in percentage) , will be hidden. If you don't have any specific purpose, I would recommend that you uncheck all these options.

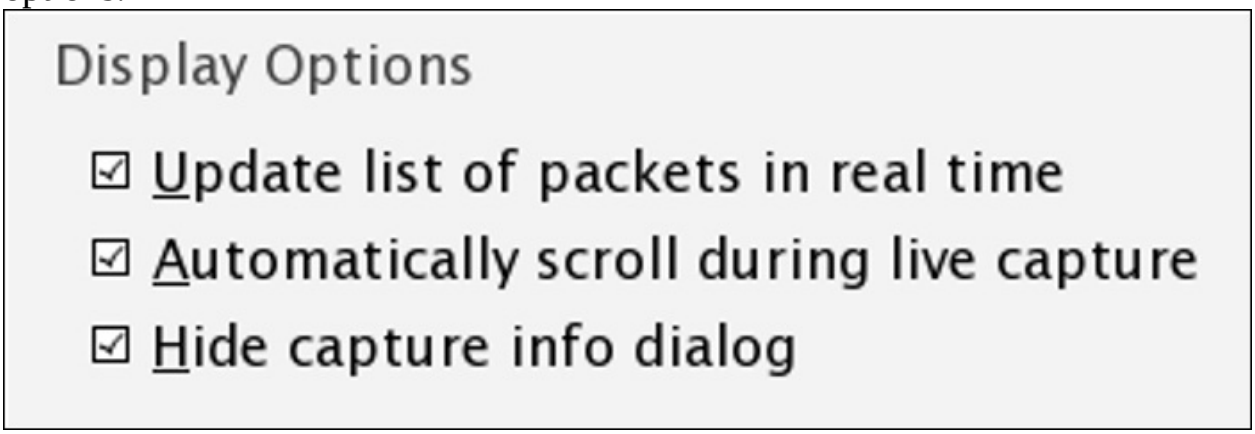
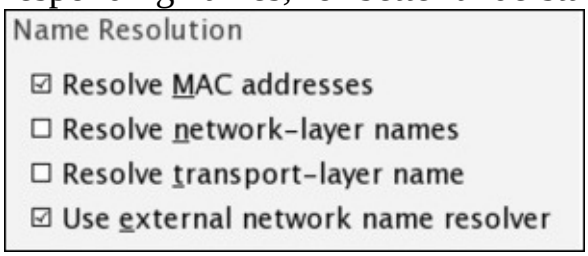


Figure 2.7: Display Options

- **Name Resolution**: If selected, this feature can resolve the Layer 2, Layer 3, and Layer 4 addresses to their corresponding names; for better understanding,



refer to the following screenshot:

Figure 2.8: Name Resolution

Why use capture filters

Capturing only traffic that meets your requirement is really useful when you have a large volume of packets flowing around. Creating your own custom capture filters can come in really handy while you analyze a production environment. Capture filters are applied before you initiate the actual capture process. In general, every packet captured by Wireshark is passed to the capturing engine so that it gets translated to a human-understandable format, but if you have applied a capture filter, Wireshark will drop the packets that don't meet your expression. All these dropped packets won't be passed to the capturing engine, . In comparison, display filters are much more specific and powerful; while using capture filters, you should be careful, because there is no way of recovering dropped packets that do not meet the expression that you created.

The **Berkley Packet Filter** (**BPF**) syntax is used to create capture filters, and several protocol analyzers use it as well, thus maintaining industry standards. It is significantly easy to learn and practice, just use the basic format to structure an expression.

How to use capture filters

Using the BPF syntax earlier, we created a simple capture filter through the capture filter dialog; let's discuss it in detail because it is really crucial to know about BPF, as it is used by a variety of analyzers.

If you're using the BPF syntax, you have to follow a certain format structure, which is a combination of two arguments: identifiers and qualifiers, which are explained as follows:

- **Identifiers**: This is the value that you are looking for in your packets. For example, if you are filtering the packets for a certain IP address, then your capture filter will look something like `host 192.168.1.1`, where the value `192.168.1.1` is an identifier.
- **Qualifiers**: These are categorized into three different sections:
    - **Type**: There are three types of type qualifiers: `host`, `port`, and `net`. In short, a type qualifier refers to the name or the number that your identifier refers to. For example, in your `host 192.168.1.1` filter, `host` is the type qualifier.
    - **Direction**: Sometimes, when you need to capture packets from a particular destination or source, we can specify direction qualifiers as well. For example, in the `src host 192.168.1.1` capture filter, `src` specifies that we've to capture packets originating from a specific host only. Likewise, if you specify `dst host 192.168.1.1`, would capture packets only destined to `host 192.168.1.1`.
    - **Proto**: This refers to protocol qualifiers that give us the feature where we can mention the specific protocol that we want to add in our expression for capture purposes. For example, if you want to capture `http` traffic coming from your host `192.168.1.1`, then your expression will look something like `src host 192.168.1.1 and tcp port 80`.

In the previous example, we combined two expressions together using the concatenation operator (&/and). Similarly, we've the alteration operator (|/or) and the negation operator (!/not), which can be used to combine and create complex filters.

For example, as per our previously created filter `src host 192.168.1.1 and`

`tcp port 80`, all the packets originating from `192.168.1.1` and going to port `80` will be captured.

If you add the `or` operator between `src host 192.168.1.1 or tcp port 80`, then when an expression in your filter matches, then the packet will be captured. This means that every packet originating from `192.168.1.1` or any packet associated with port `80` will be captured regardless of the second condition.

In the case of the `not` operator, a capture filter such as `not port 80` states that any packet associated with port `80` should not be captured.

Once you start working in a production environment, you will see how common it is to combine filters using the AND, OR, and NOT operators.

An example capture filter

Though you have a variety of filters available in Wireshark itself, which can give you an overview of the BPF syntax, to access the present filters by default, go to **Capture** | **Capture Filers** or click on the **Capture Options** button in the main toolbar and then click on **Capture Filter**. From the same window, we have an option to create new filters that we already discussed.

Refer to the following table for sample capture filters:

Filters	**Description**
host 192.168.1.1	All traffic associated with host 192.168.1.1
port 8080	All traffic associated with port 8080
src host 192.168.1.1	All traffic originating from host 192.168.1.1
dst host 192.168.1.1	All traffic destined to host 192.168.1.1
src port 53	All traffic originating from port 53
dst port 21	All traffic destined to port 21
src 192.168.1.1 and tcp port 21	All traffic originating from 192.168.1.1 and associated with port 21
dst 192.168.1.1 or dst 192.168.1.2	All traffic destined to 192.168.1.1 or destined to host 192.168.1.2
not port 80	All traffic not associated with port 80
not src host 192.168.1.1	All traffic not originating from host 192.168.1.1
not port 21 and not port 22	All traffic not associated with port 21 or port 22

tcp	All tcp traffic
Ipv6 tcp or udp host www.google.com ether host 07:34:aa:b6:78:89	All ipv6 traffic All TCP or UDP traffic All traffic to and from Google's IP address All traffic associated with the specified MAC address

Note

It is essential to know about the BPF syntax. As and when you get into Wireshark in more detail, you will feel its importance. I would suggest that you practice it once when you are comfortable with the syntax.

Capture filters that use protocol header values

Capture filters can be created on the basis of offset values present in protocol header fields. The syntax to create such filters looks like `proto[offset:size(optional)]=value`. Here, `proto` is any protocol that you want to filter, `offset` is the position of the corresponding value in the header, `size` is the length of the data you are looking for, and `value` is the data you want to find.

Say, for instance, we want to capture only ICMP reply packets; now, if you observe the following figure, you will note that the ICMP header type is located at the first place and the offset counting starts from `0`. So, the offset value will be `0` in this case, and the size of the field is 1 bytes. We have all the required information to create a capture filter, so now, the resulting expression will look like `icmp[0:1]=0`.

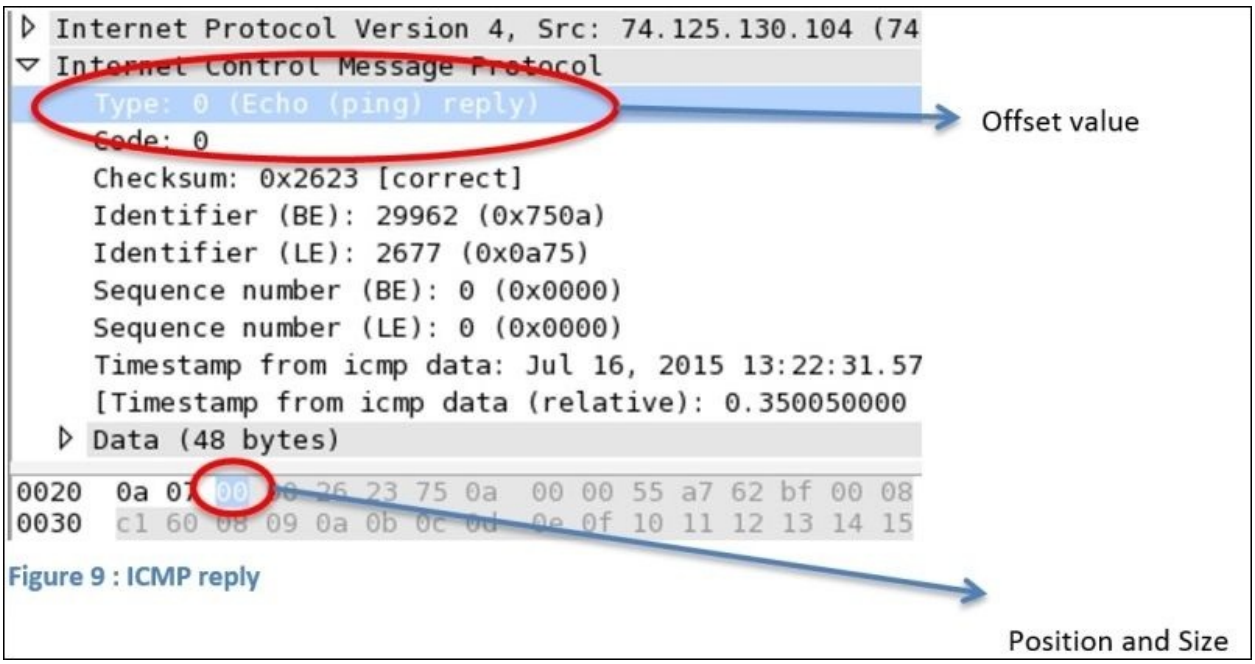


Figure 9 : ICMP reply

Figure 2.9: ICMP reply

Let's try to apply the same to Wireshark; we will then ping www.google.com to check whether it works.

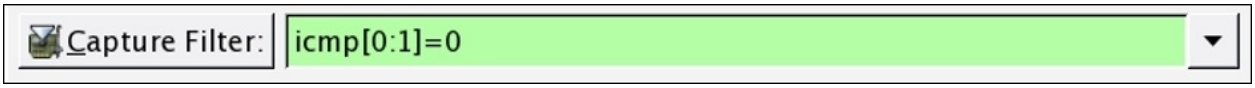


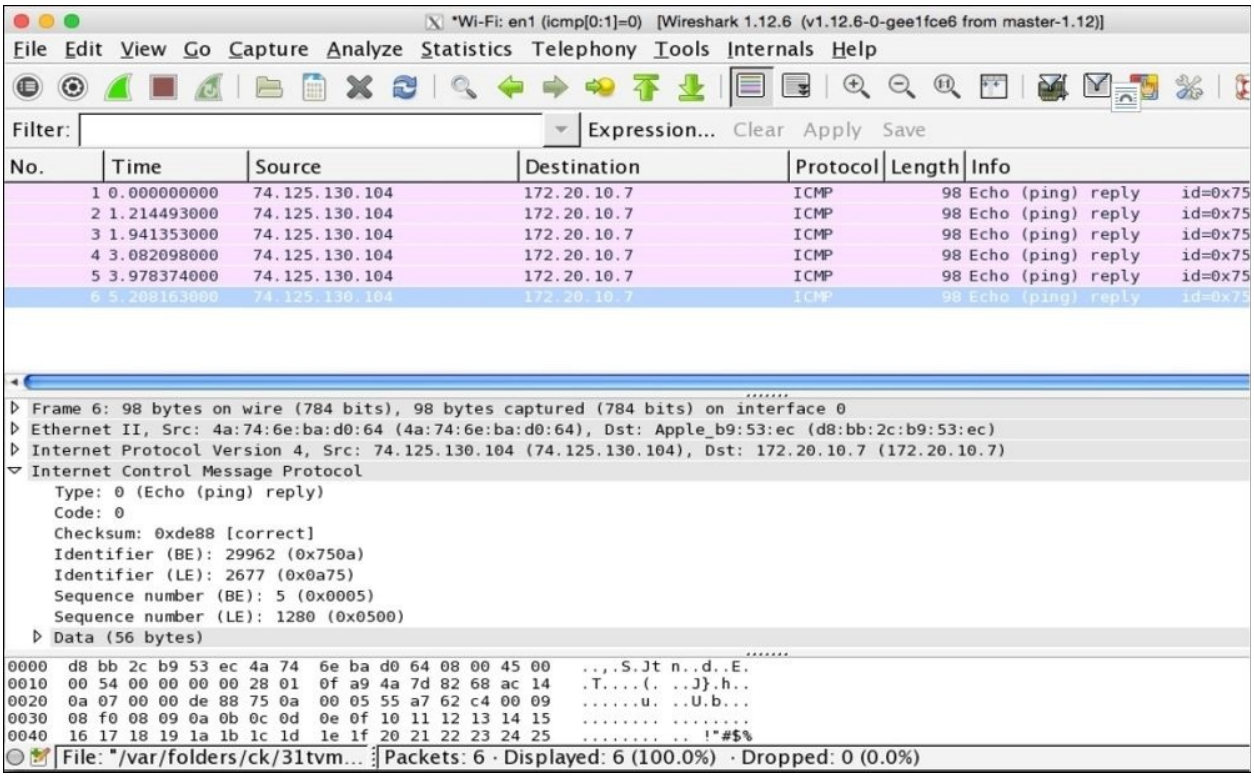
Figure 2.10 : ICMP capture filter

Let's ping www.google.com and check whether it works.

```
charits-MacBook-Pro:~ NotFound$ ping www.google.com
PING www.google.com (74.125.130.104): 56 data bytes
64 bytes from 74.125.130.104: icmp_seq=0 ttl=40 time=350.085 ms
64 bytes from 74.125.130.104: icmp_seq=1 ttl=40 time=559.549 ms
64 bytes from 74.125.130.104: icmp_seq=2 ttl=40 time=282.911 ms
64 bytes from 74.125.130.104: icmp_seq=3 ttl=40 time=420.467 ms
64 bytes from 74.125.130.104: icmp_seq=4 ttl=40 time=311.638 ms
64 bytes from 74.125.130.104: icmp_seq=5 ttl=40 time=539.921 ms
```

Figure 2.11: Browse google.com

As a result, Wireshark will capture only the ICMP reply packets. Using the same technique, you can filter out traffic on the basis of the protocol header value:



The following table lists some sample bytes-based capture filters for TCP and ICMP; try practicing them too:

Filter	**Description**
`icmp[0] = 0`	ICMP request packets
`icmp[0:1] = 8`	ICMP reply packets
`icmp[0:1] = 3` `tcp[13] = 2`	ICMP destination host unreachable packets TCP SYN flag packets only
`tcp[13] = 18`	TCP SYN/ACK flag packets only
`tcp[13] = 32`	TCP URG flag set packets only

Display filters

Display filters are much more flexible and powerful when compared to capture filters. Display filters do not discard any packets; instead, the packets are hidden to make viewing convenient or convenience. Discarding packets is not a very effective practice because, once the packets are dropped, they cannot be recovered. When you apply the display filter, only those packets that meet the specification of your filter will be displayed. In the the second column of the status bar of the Wireshark window, you will see a number of packets displayed after you apply a filter.

A display filter can be used for a capture file in the **Filter** dialog box located above the **Packet List Pane**. Display filters are more popular than capture filters. The syntax used for display filters can be easily adapted and applied. For new users, a display filter is like a super power that gives you the functionality of hiding inappropriate packets in run-time that do not meet your requirements as per the current scenario.

Display filters can be created on the basis of several different constraints such as the IP address, protocols, port numbers, and header values in specific protocols. There are lot of conditional tools and concatenation operators that can be used to create complex expressions. You can combine different sets of expressions to get more specific sets of packets that we are looking for. Each and every packet shown in the **Packet List Pane** can be filtered using the fields that a packet contains.

Display filters do not delete data; instead, packets are hidden, which can be made visible again once the filter in the **Filter** dialog above the list pane is cleared. For instance, to display only ICMP packets, just enter `ICMP` in the filter dialog and click on **Apply**; it's really simple, isn't? If you want to see all packets again, just click on the **Clear** button and everything will be back to normal.

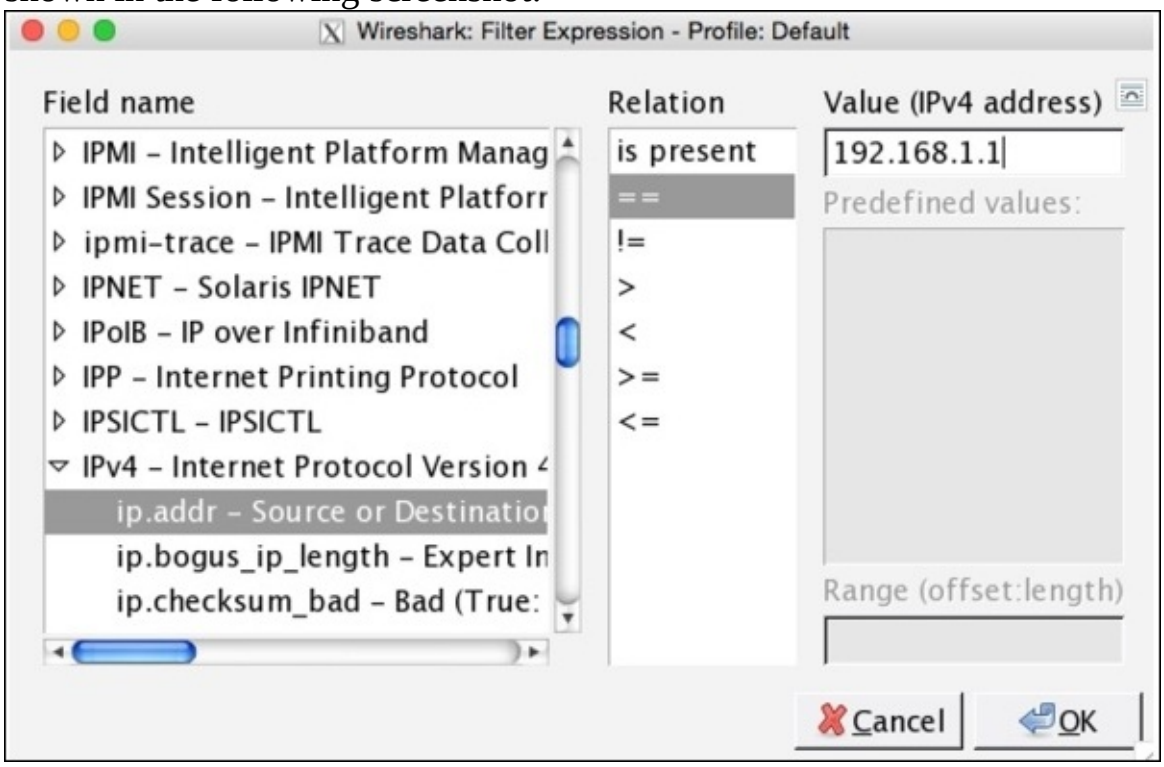
Wireshark has a very awesome feature that can assist you while creating your filter. Just click on the **Expression** button at the end of the **Filter** dialog box, choose the protocol you want to filter, and specify the value if there is one.

Using the **filter expression** dialog is really easy, and if you are a beginner, then

this is a boon for you. Let's learn how to use the expression dialog.

Figure 2.12 : The filter expression

1. As show in the preceding screenshot, click on the **Expression** button.
2. Now, you will be presented with the **Expression** window like the one shown in the following screenshot:



- For example, if you want to see only packets associated with `ip:192.168.1.1`, then just scroll down in the **Field Name** to find **IPv4**. Then, expand the section and choose the **ip.addr** option.
- Then, from the **Relation** box next to it, choose the operator you wish to add in your expression.
- At last, write the IP you are looking for in the **Value (IPv4 address)** box.
- At last, just click on **OK**. If you've followed all the steps up to here correctly,

then you would be able to see the packets originated from the `ip` that you mentioned (change **192.168.1.1** to your IP address).

- Below the **Value** box, there is a **Predefined value** box that is used when a certain protocol restricts us to use only a specific set of values. You can choose a value form here.
- Below the **Predefined Value** box, there is a **Range** box that allows us to enter a range of values such as `1-78, 0-5, 120-255` if the protocol allows the same.

This is one of the easiest ways to create a display filter; there is one more way following which we can also create such filters. Entering filters manually can drastically increase the speed of your work, but it requires a bit more skill than there are in a novice user.

Before we start digging into creating filters manually, I want you to know about a few more things, such as comparison and logical operators. These can be used to create simple and the most complex filters for Wireshark.

The following table lists the comparison operators used to create display filters:

Operator	Description
==/eq	Equal to
!=/ne	Not equal to
</lt	Less than
<=/le	Less than equal to
>/gt	Greater than
>=/ge	Greater than equal to

Next, let's have a look at the logical operators that are used to combine different conditions together. The following table lists all of them:

Operator	Description

AND/&&	The AND logical operator is used when we want both parts of the expression to state `true`. For example, the `ip.src==192.168.1.1` and `tcp` filters would only display packets originated from `ip 192.168.1.1` and associated with the `tcp` protocol. Only the packets that match both the expressions will be shown.
OR/||	The OR logical operator is used when we just focus on one condition to be true at a time; if both are true, even then it's ok. For example, the `port 53` or `port 80` filters would display all packets associated with port `53` (`DNS`) along with all packets associated with port `80` (`http`).
NOT/!	The NOT logical operator is used when we want to exclude some packets from the list pane. For example, the `!dns` filter would hide all the packets associated with the DNS protocol.

Retaining filters for later use

Sometimes, you will have a requirement where having access to previously created filters would make your work easy and fast enough. Wireshark gives you the facility where you can retain your display filters through their saved names and use them at a later point of time whenever required. This option will save you the great amount of time and effort required to type some of the complex display filters. To create one for yourself, follow the given steps:

1. Go to **Analyze** | **Display filters**; this will give you a window like the one shown in the following screenshot:

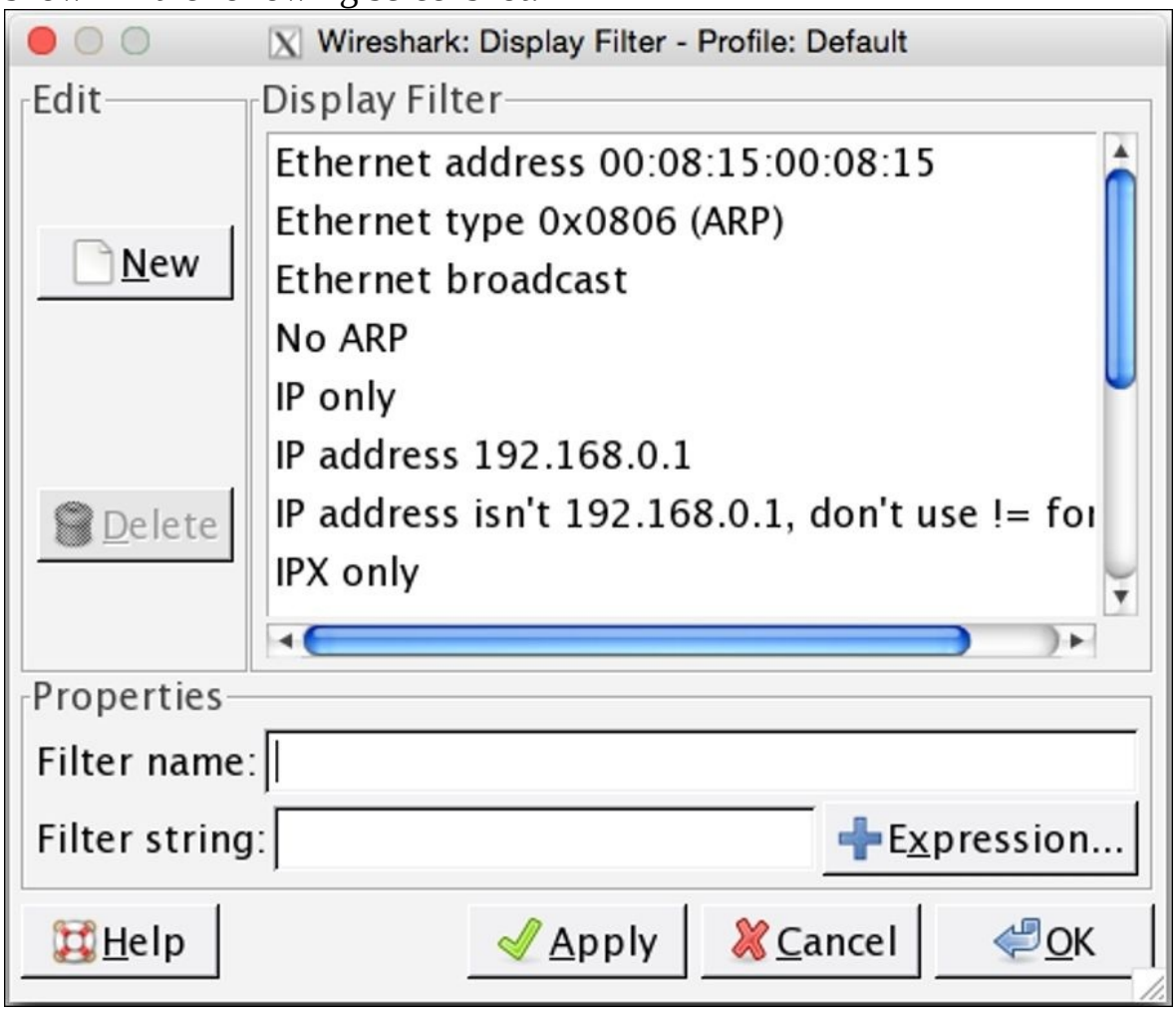


Figure 2.13: Adding Display Filters

- Now, click on **New**, enter the values in the **Filter name** and **Filter string** fields. For instance, we want to create a display filter for no ARP packets. Then, the values will look something like the following screenshot:

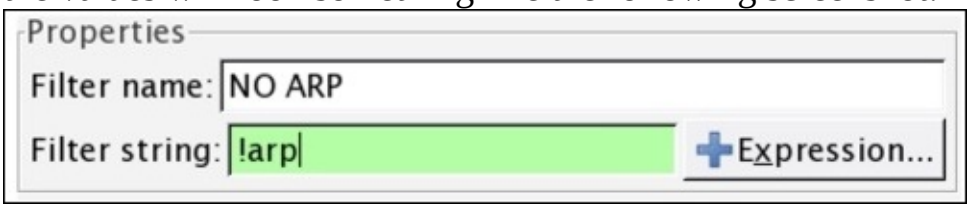


Figure 2.14 : Creating a new filter

- After entering the same, click on **Apply**. Now, in the list of default filters present you would be able to see **NO ARP**, which can be used later.
- Make sure that the **Filter String** box is shown with a green background, which denotes that your expression is correct; if it is in red color, then you need to recheck it, and if it is in yellow, this denotes that the results can be unexpected. Now, you can click on **Apply** and then click on **Ok**.
- If you need assistance to create any filter you want, simply click on the **Expression** button next to the **Filter string** box, where all the protocols and majorly used filter expressions can be found.
- The **Delete** button will assist you in deleting an existing filter from the list.
- The **Cancel** button will discard any unsaved changes and close the window.
- The **Ok** button commits **Save** and will close the window.
- Now, let's try applying the filter we just created. Navigate to **Analyze** | **Display Filter** | (Scroll and select) **Display Filter** | **Apply**.

Try following the same and create your own display filter that you might want to reuse.

Searching for packets using the Find dialog

If you want to find a packet for a particular criterion, you can use the Find dialog. It has a couple of useful search techniques that can be applied easily and effectively on an already captured file or on a live running capture. You can access the Find utility by navigating to **Edit** | **Find packets** or using the shortcut *Ctrl + F*.

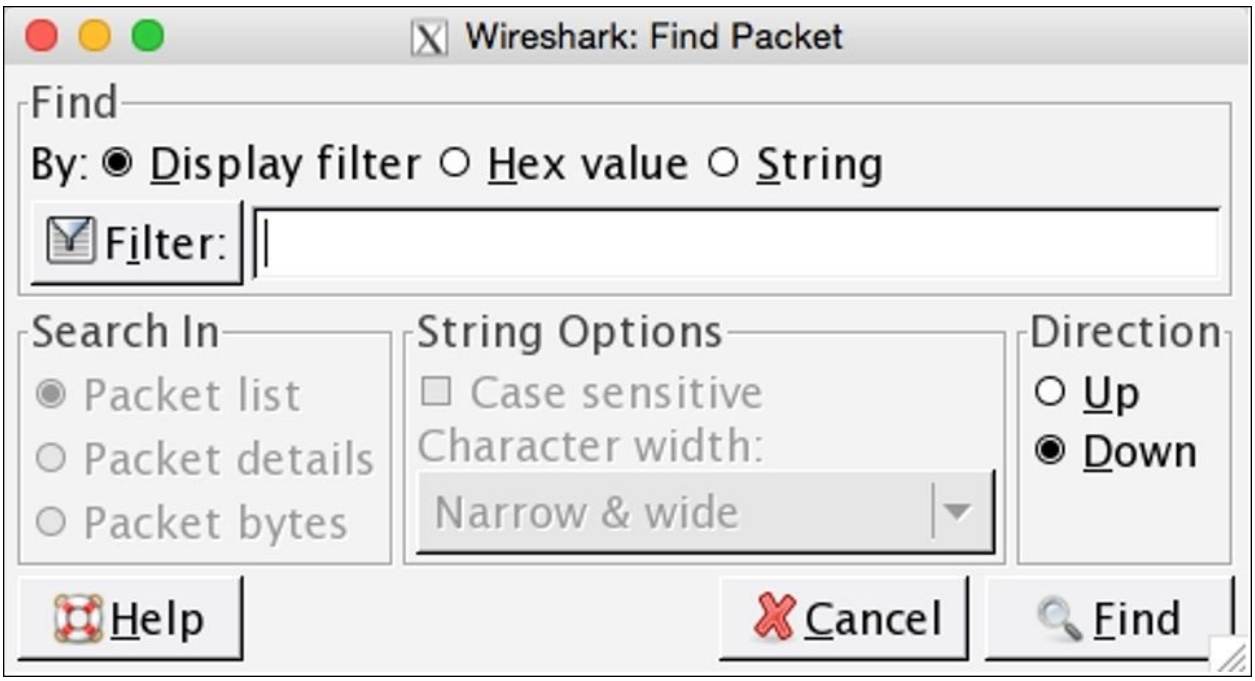


Figure 2.15: The Find Packet dialog

Let's see some more configurable options in it:

- The **display filter**: After capturing the traffic, while analyzing whether you just want to see some specific packets based on a certain IP /Port/ Protocol, those packets that meet a certain criteria will be displayed in the list pane, for example:
  - The `ip.addr == 192.168.1.1` (based on an IP address)
  - The `port 8080` (based on a port number)

    - `http` (based on a protocol)
- The **Hex value**: If you have the hex value for a certain packet that you are looking for, then this option can be selected. Just write the physical address separated by colons, for example:
    - `0A:C4:22:90:45:00`
    - `AA:BB:CC`
- **String**: The next and last option is a text-string-based search where you can enter the name of the DNS server, name of the machine, and any resolved name that you know about (enter any string or word), for example:
    - Cisco
    - An administrator
    - A web server
    - Google
- **Search In**: This feature gives us the ability to search in a specific pane. For instance, if you are looking for a packet in the **bytes** pane, which matches the value **Google** (the ASCII value in the packet bytes pane will be matched), then we can go ahead and first choose the **String** option and then check the **Search In** box and choose **Packet Bytes**.
- **String Options**: To use this, first select the **String** option and then select **Case-Sensitive** and then if you want, choose the character width as well (but I would suggest not changing this unless until you have a specific reason to do so).
    - **Direction**: This last option changes the direction of a search; you can change it to upward or downwards.

Once you have customized the options, enter the text and click on **Find**. This will give you the first exact capture that matches your criterion. To move back and forth between the matched packets, you can use *Ctrl + N* (next) and *Ctrl + B* (previous).

Colorize traffic

For better and convenient viewing experience, Wireshark gives us a feature where we can colorize a certain type of traffic that we want to highlight. Colorization of traffic is done in order to distinguish between different sets of traffic. Coloring a specific set of traffic with a different rule other than the default one will be like finding a needle in a haystack.

The default profile for most protocols is already created because of which we are able to see traffic in the packet list pane in different colors. You can access it by navigating to **View** | **Edit coloring rules** or clicking on the **Edit coloring rules** button from the main toolbar to open a window as shown in the following screenshot:

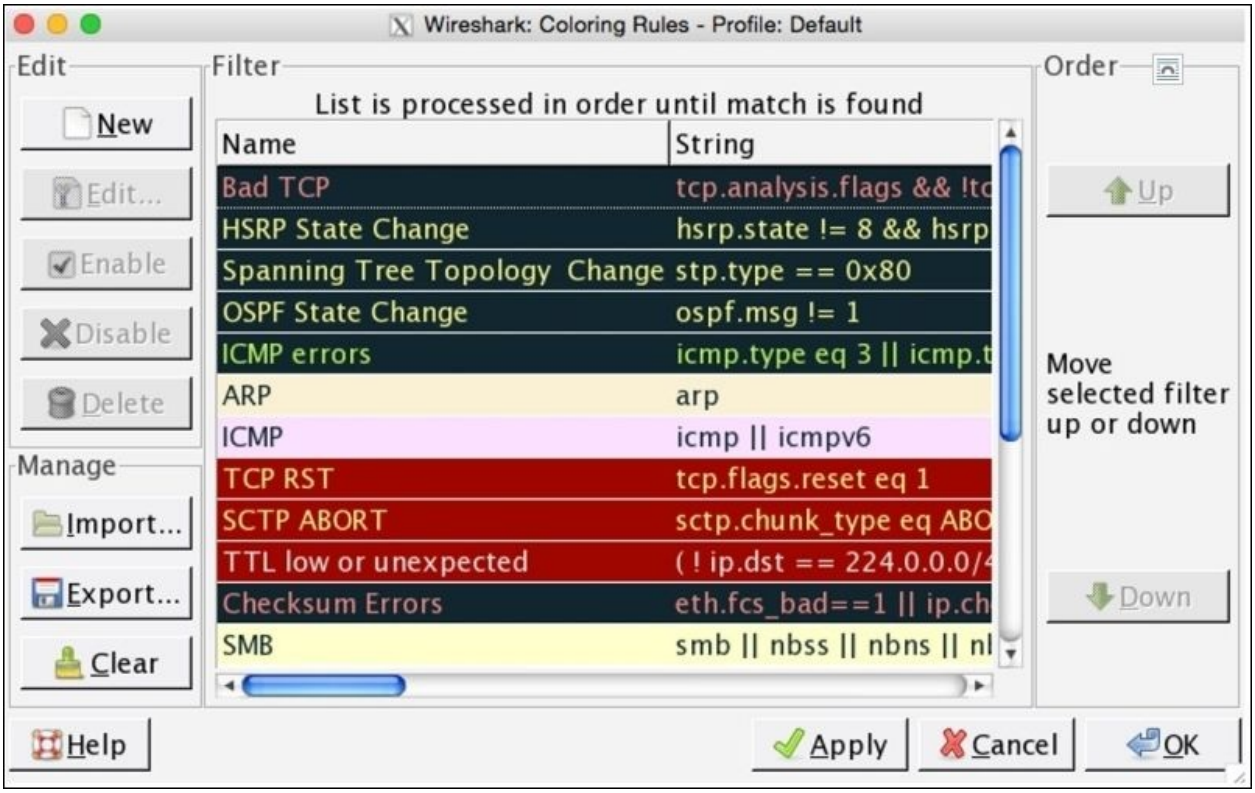


Figure 2.16: Coloring rules

All rules that are currently saved as part of your global configuration file to

colorize traffic with certain foreground and background colors are listed in this dialog. Every packet listed in the packet list pane follows a certain rule, which gives them a unique and distinguished look and feel.

Let's use this feature and color the `http error` packets with a color of our choice. Say, for instance, I've a web server running on my machine that is used by the clients connected for file accessing purpose. Now, one of the clients in my network is trying directory listing and gets `HTTP 404` error messages. These error messages will pop up in my packet list pane but will be colored using the same `http` coloring rule that makes these errors less visible to me. To make this more visible, I want to colorize the `HTTP 404` error messages with a `black` background and with a `cyan` foreground. Follow the steps shown here that will achieve the same:

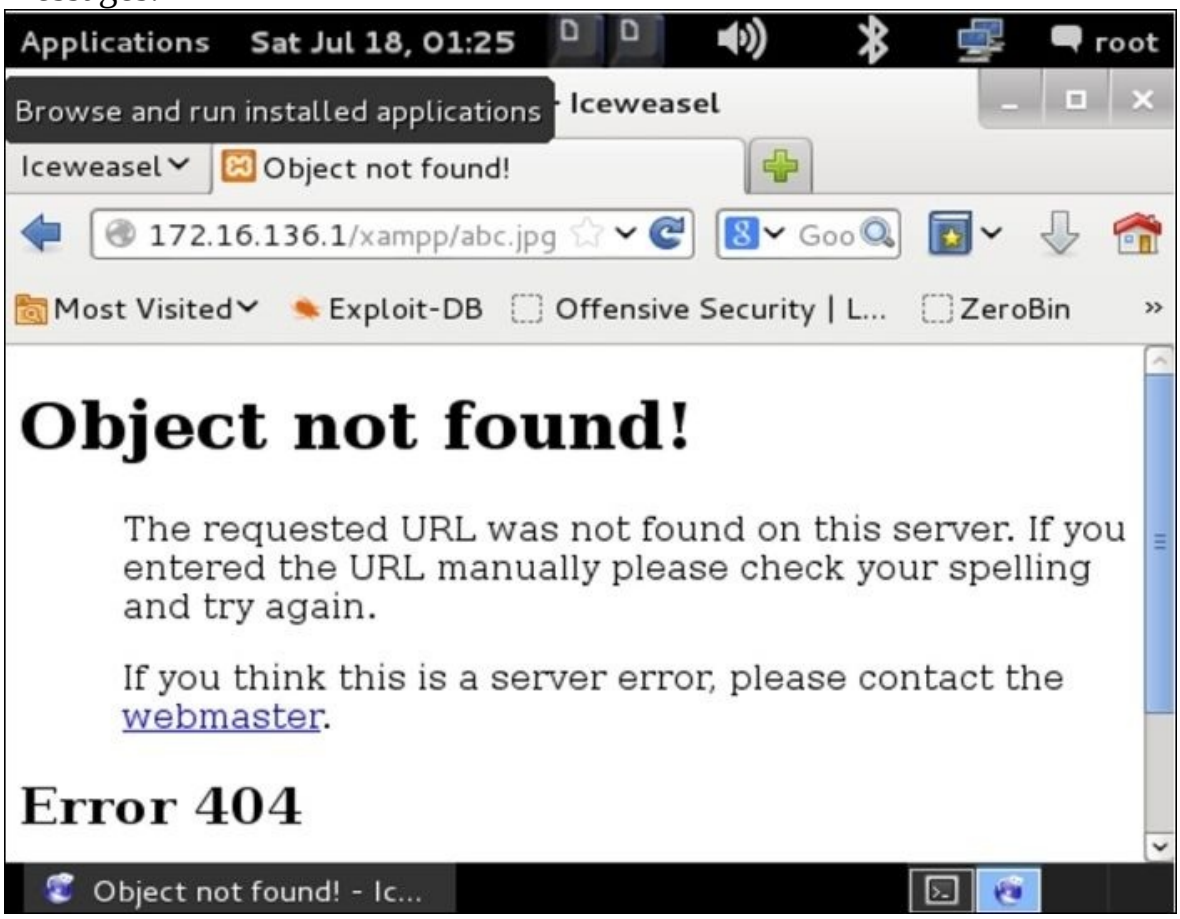
1. I have configured a Linux box running on `172.16.136.129`, and my Mac OS is running on `172.16.136.1` that serves as a web server for Linux, as Shown in the following screenshot:

Figure 2.17: The web server running on 172.16.136.1

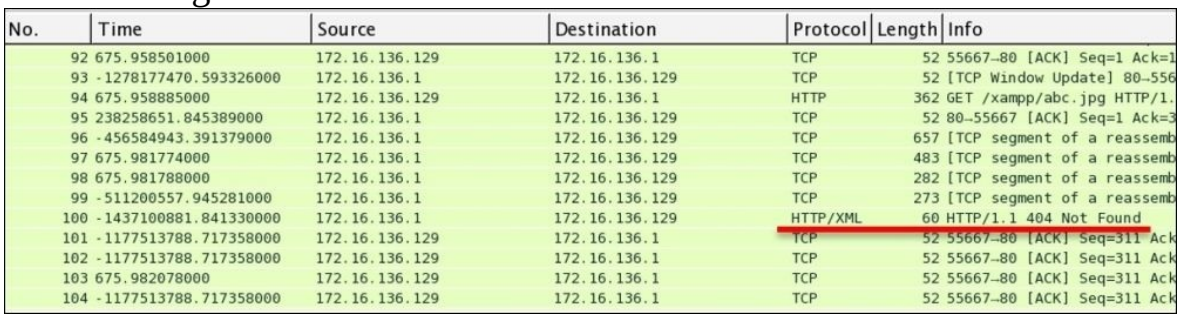
Normal traffic from a Linux-accessing web server looks something like the screenshot here:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.16.136.129	172.16.136.1	TCP	60	55658→80 [SYN] Seq=0 Win=2920
2	-950618696.077286000	172.16.136.1	172.16.136.129	TCP	64	80→55658 [SYN, ACK] Seq=0 Ack
3	-2021440336.836621000	172.16.136.129	172.16.136.1	TCP	52	55658→80 [ACK] Seq=1 Ack=1
4	-1898165200.561362000	172.16.136.1	172.16.136.129	TCP	52	[TCP Window Update] 80→55658
5	41863044.612094000	172.16.136.129	172.16.136.1	HTTP	355	GET /xampp/ HTTP/1.1
6	0.001038000	172.16.136.1	172.16.136.129	TCP	52	80→55658 [ACK] Seq=1 Ack=304
7	0.084997000	172.16.136.1	172.16.136.129	HTTP	940	HTTP/1.1 200 OK (text/html)
8	0.085422000	172.16.136.129	172.16.136.1	TCP	52	55658→80 [ACK] Seq=304 Ack=88
9	381882809.099438000	172.16.136.129	172.16.136.1	HTTP	400	GET /xampp/head.php HTTP/1.1
10	0.106560000	172.16.136.1	172.16.136.129	TCP	52	80→55658 [ACK] Seq=889 Ack=65
11	-1437096632.910449000	172.16.136.129	172.16.136.1	TCP	60	55659→80 [SYN] Seq=0 Win=2920
12	-950618696.095408000	172.16.136.1	172.16.136.129	TCP	64	80→55659 [SYN, ACK] Seq=0 Ack
13	-136085583.409139000	172.16.136.129	172.16.136.1	TCP	52	55659→80 [ACK] Seq=1 Ack=1 Wi
14	-1321431987.061550000	172.16.136.1	172.16.136.129	TCP	52	[TCP Window Update] 80→55659

2. Now that everything is up and running, we will try to do some directory listing manually from Linux, which will give eventually `HTTP 404` error messages.



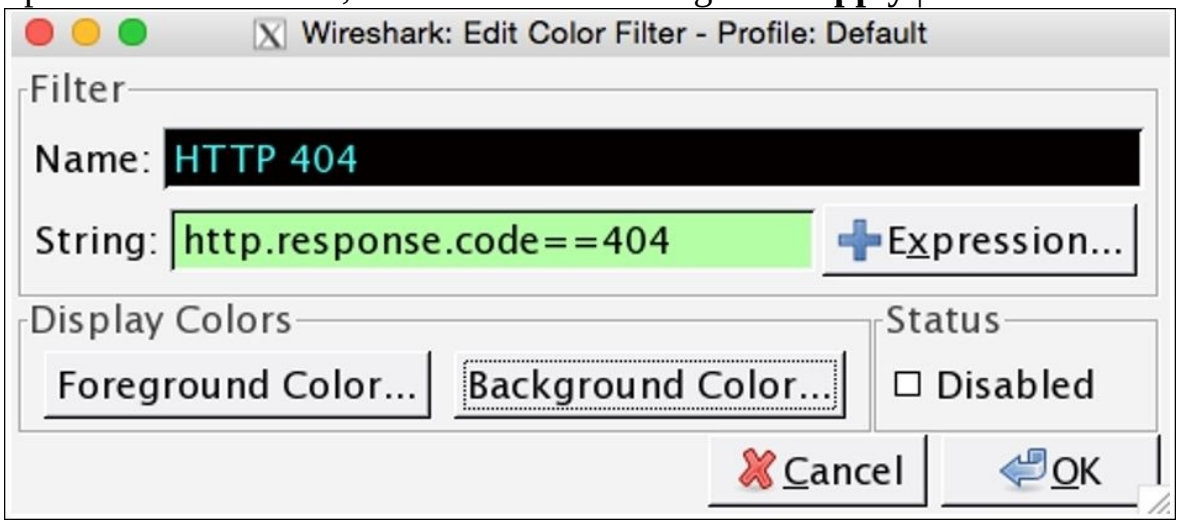
The traffic generated through this request is captured, which can be seen in the following screenshot:



No.	Time	Source	Destination	Protocol	Length	Info
92	675.958501000	172.16.136.129	172.16.136.1	TCP	52	55667→80 [ACK] Seq=1 Ack=1
93	-1278177470.593326000	172.16.136.1	172.16.136.129	TCP	52	[TCP Window Update] 80→556
94	675.958885000	172.16.136.129	172.16.136.1	HTTP	362	GET /xampp/abc.jpg HTTP/1.
95	238258651.845389000	172.16.136.1	172.16.136.129	TCP	52	80→55667 [ACK] Seq=1 Ack=3
96	-456584943.391379000	172.16.136.1	172.16.136.129	TCP	657	[TCP segment of a reassemb
97	675.981774000	172.16.136.1	172.16.136.129	TCP	483	[TCP segment of a reassemb
98	675.981788000	172.16.136.1	172.16.136.129	TCP	282	[TCP segment of a reassemb
99	-511200557.945281000	172.16.136.1	172.16.136.129	TCP	273	[TCP segment of a reassemb
100	-1437100881.841330000	172.16.136.1	172.16.136.129	HTTP/XML	60	HTTP/1.1 404 Not Found
101	-1177513788.717358000	172.16.136.129	172.16.136.1	TCP	52	55667→80 [ACK] Seq=311 Ack
102	-1177513788.717358000	172.16.136.129	172.16.136.1	TCP	52	55667→80 [ACK] Seq=311 Ack
103	675.982078000	172.16.136.129	172.16.136.1	TCP	52	55667→80 [ACK] Seq=311 Ack
104	-1177513788.717358000	172.16.136.129	172.16.136.1	TCP	52	55667→80 [ACK] Seq=311 Ack

We can see, in the preceding captured traffic, that the client requested the **abc.jpg** resource, which was not available; thus, the client received a **404 Not found** error.

3. We figured out easily because there is just one client requesting a single resource. Consider a production environment where thousands of clients are present and they might do the same. In such cases, coloring a specific set of packets with a different rule is a game changer.
4. Navigate to **Edit Coloring Rules** | **New**. Type **HTTP 404** in the **Name** box. Type `http.response.code==404` in the **String** box. Choose the **Foreground Color** option as Cyan, and choose the **Background Color** option as Black. Then, click on **OK** and navigate to **Apply** | **OK**.



5. Once you click on **Apply**, you will see that only the `HTTP 404` error packets will be colored according to your new coloring rule.

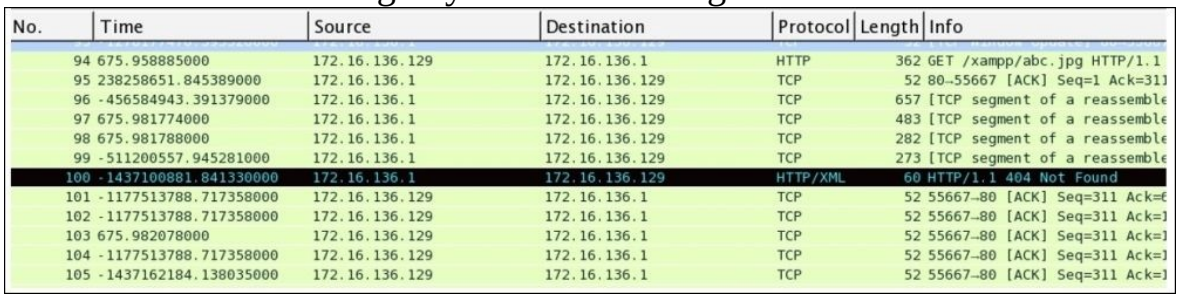


Figure 2.19: After applying the new coloring rule

Try the same using a virtual environment to give yourself more insight into the topic.

Coloring rules listed in the **Edit Coloring Rules** dialog will be checked in a top-to-bottom manner. With every packet, there is coloring rule information attached that can be listed from the **Packet Details Pane** under the **Frame** section. Consider the following screenshot illustrating the same:

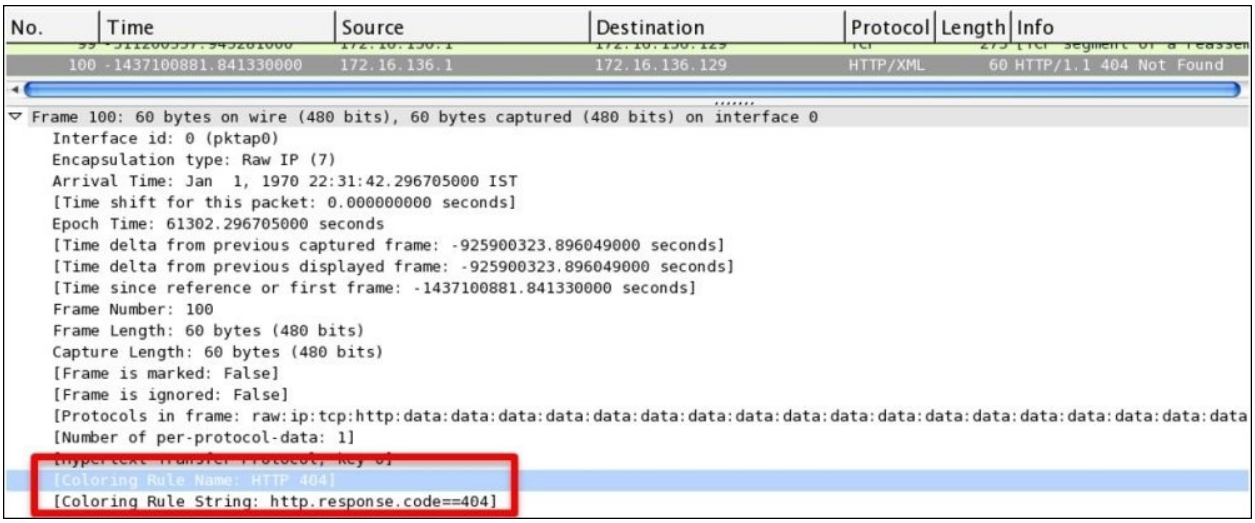


Figure 2.20: Coloring info in a frame header

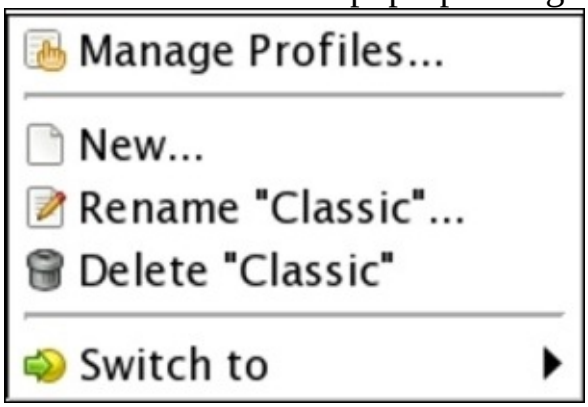
Create new Wireshark profiles

Profiles in Wireshark are like customized environments, which can save a significant amount of time while auditing a network. A profile is a set of different components, such as capture filters, display filters, time preferences, column preferences, protocol preferences, color profiles, and so on, that fit together and give you a case-specific scenario, which you might require instantly.

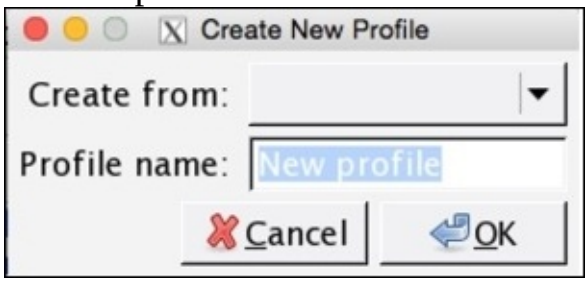
Importing and exporting profiles is very easy in Wireshark, which is pretty useful while auditing a network where you don't have your preinstalled tools. Just copy and paste the Profile configuration files in a certain directory to use them. To create a profile, follow these steps:

1. Right-click on the **Profile** column in **Status Bar**.

2. Click on **New...** in the pop-up dialog.



3. Now, choose any profile you wish to use as a template and type the name of the new profile.



4. And then, click on **OK**.

Now, in the status bar, you will see the the same profile has been activated. The changes that you are going to make in this profile stay here, for example, you can create capture/display filters, change protocol preferences, and change color preferences. This means that any changes in a profile do not alter the contents of other profiles that are saved.

This way, we can create different profiles for case-sensitive scenarios that can save time and make the task easy.

Summary

Using the Find utility can be pretty useful sometimes, and can be accessed from the Edit menu in Wireshark. The Find utility gives us various vectors to search the packet content.

Filtering traffic lets you see only those packets that you are interested in; there are two types of filters: display filters and capture filters.

Display filters hide the packets, and once the expression you made is cleared, all packets can be seen again. However, capture filters discard the packets that do not meet the expression that you created. Discarded packets are not passed to the capturing engine.

Capture filters use the BPF syntax, which is an industry standard and is used by several other protocol analyzers.

Coloring preferences can be really useful while filtering a certain set of traffic based on a specific expression. Distinguishing packets will be become easy, as the matched packets will be shown with a different coloring scheme.

Profiles are like case-sensitive scenarios that can save your time and workload. Changes made to the profiles with respect to its different components, such as display/capture filter and color/protocol/time preferences, stay within the same.

Exporting profiles and various settings from Wireshark is very simple, which make the software more portable.

In the next chapter, you will learn how to work with Wireshark's advanced features such as graphs and statistical options.

Practice questions

Q.1 Explain the difference between display filters and capture filters, and which is more efficient in terms of system resource utilization.

Q.2 Explain the difference between **Find Utility** and **Filters**. Use the Find utility to search using hex values.

Q.3 Create a capture filter to capture only ARP broadcast packets.

Q.4 Create a capture filter to capture all packets except the packet destined to and originated from your physical address.

Q.5 Create a capture filter to capture only TCP SYN packets and TCP ACK packets.

Q.6 Create a capture filter to capture HTTP traffic sent only from you machine.

Q.7 Create a display filter to show packets originating only from your IP.

Q.8 Create a display filter to see packets that are only related to the protocol Secure Socket layer.

Q.9 Create a display filter to see only the ICMP destination host's unreachable packets.

Q.10 Create a display filter to see only TCP packets with a FIN and ACK flags set.

Q.11 Create a display filter to show TCP packets with header length greater than 40.

Q.12 Change the coloring scheme for all the DNS query Type A packets to the color of your choice.

Q.13 Change the coloring scheme of all HTTP error messages to the color of your choice.

Q.14 Create a profile with the name DNS using a default profile, and create a capture filter in this profile that will capture DNS traffic. Then, change the coloring scheme of all DNS response packets to the color of your choice.

Chapter 3. Mastering the Advanced Features of Wireshark

In this chapter, we will look under the hood of the Statistics menu in Wireshark and work with different command-line utilities that come pre-packaged with Wireshark. Here, we will cover the following topics:

- Collecting network stats using Wireshark's Statistics menu
- LabUp—Summary, Protocol Hierarchy, Conversations, and Endpoints
- Mapping overall traffic in graphical form
- LabUp—Graphs
- View network traffic in plain-text form
- LabUp—TCP Streams
- Learn how to view logged anomalies in your trace file
- LabUp—Expert Infos
- Using command-line tools for protocol analysis
- LabUp—CommandLine
- Practice questions

With Wireshark, you can access a variety of statistics about the packets and protocols involved in the communication between two hosts. We can collect basic as well as advanced and specific information about protocols that are involved in the communication process. We will discuss most of the useful tools available in this menu, which can give us a better insight into dealing with day-to-day complex situations.

The Statistics menu

Statistics in Wireshark are not presented to you just through recorded figures; there are graphical features too, which can present the figures in terms of graphs. Using this, the analysis process becomes easier and much efficient. Multiple types of graphs are available, which we can use to collect valuable information.

Command-line tools are like a samurai's sword, which will enhance the capability of a moderate user to become and act like an advanced user. In this chapter, we will see a couple of inbuilt tools that are command based.

Using the Statistics menu

A wide range of tools related to network stats is available in the menu, which facilitate users in gaining information ranging from general info to specific protocol related info in detail.

The general details with respect to the packets captured, filters applied, marked packets, and various other stats can be checked in the Statistics menu. Though this option is just for informational purpose, at times this can be pretty much useful.

To access the summary stats, click on **Statistics | Summary**; now, you will be able to see a window, as shown in the upcoming screenshot.

The Summary dialog is partitioned into a couple of sections, which are as follows:

- **File**: General information, such as the name of the file, location of the file, format used, and encapsulation, is listed under this
- **Time**: This section will tell you the time when the first and the last packets were captured and the time elapsed (total capture duration)
- **Capture**: This lists the name of the OS along with the version used and the interface used to dump packets from the live network traffic
- **Comments**: This shows any comments that the user mentioned for reference
- **Interface(s)**: This lists the details of every interface, using which the traffic is captured
- **Display**: This section gives statistics regarding any display filter that has been used and the percentage of ignored packets after a filter was applied

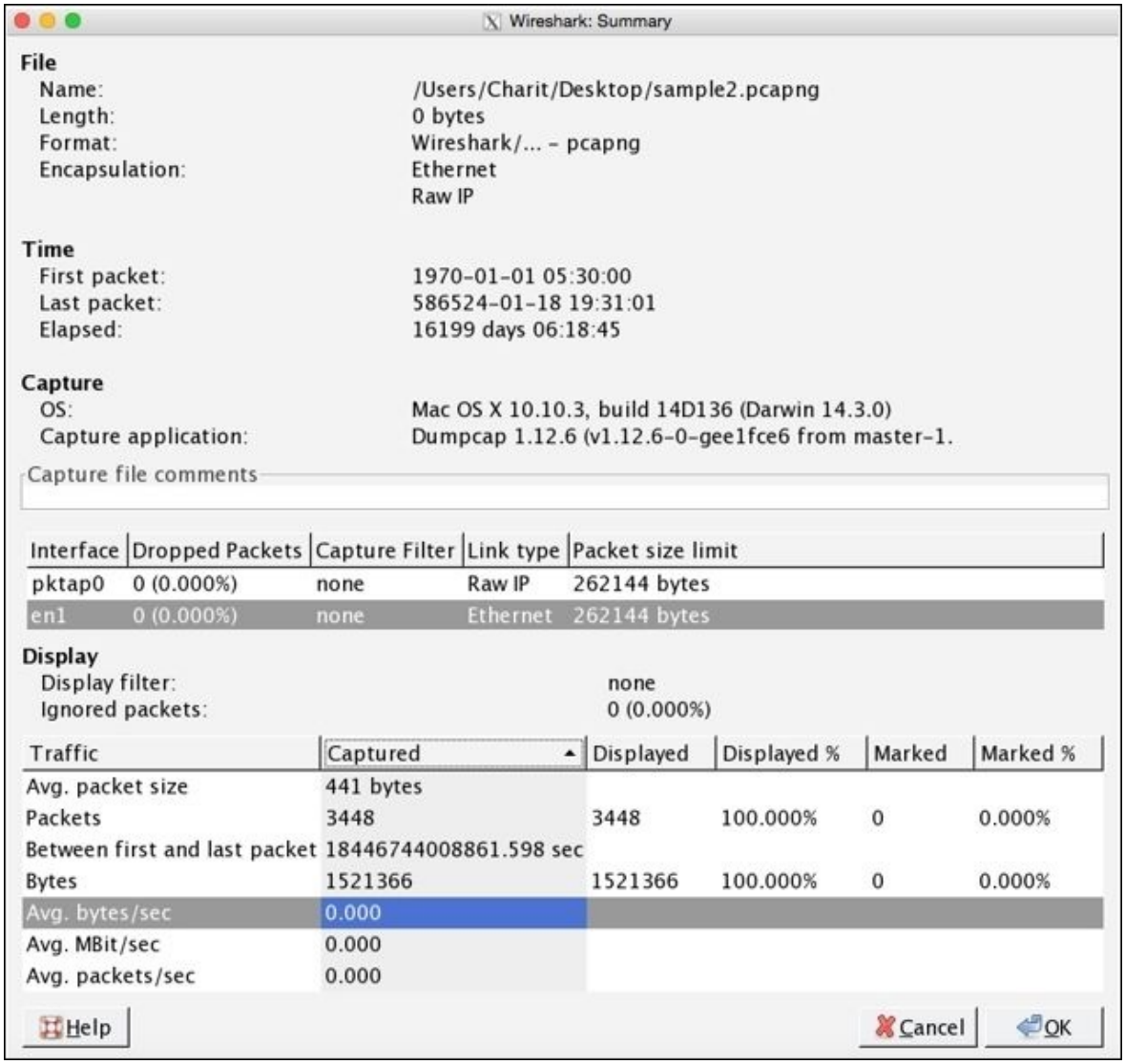


Figure 3.1: Summary dialog

Just below the **Display** section, you must see a few columns listing various details, which include a summary in a tabular format that is grouped on the basis of different categories, such as average packet size, total number of packets captured, time elapsed between the first and last packet captured, and so on.

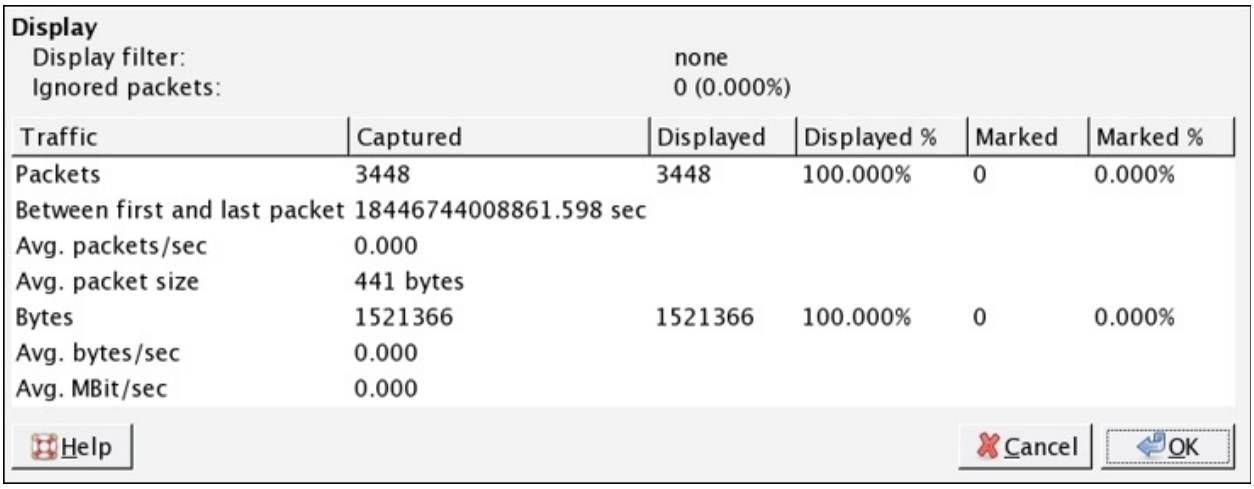


Figure 3.2: Without display filter(screenshot 1)

Let's say, for instance, we have a capture file over which we have applied the display filter **http**. After this, we can access the **Summary** option. Take a look at the following screenshot and try to compare them in order to understand the difference a display filter would make in the representation of the packets related summary.

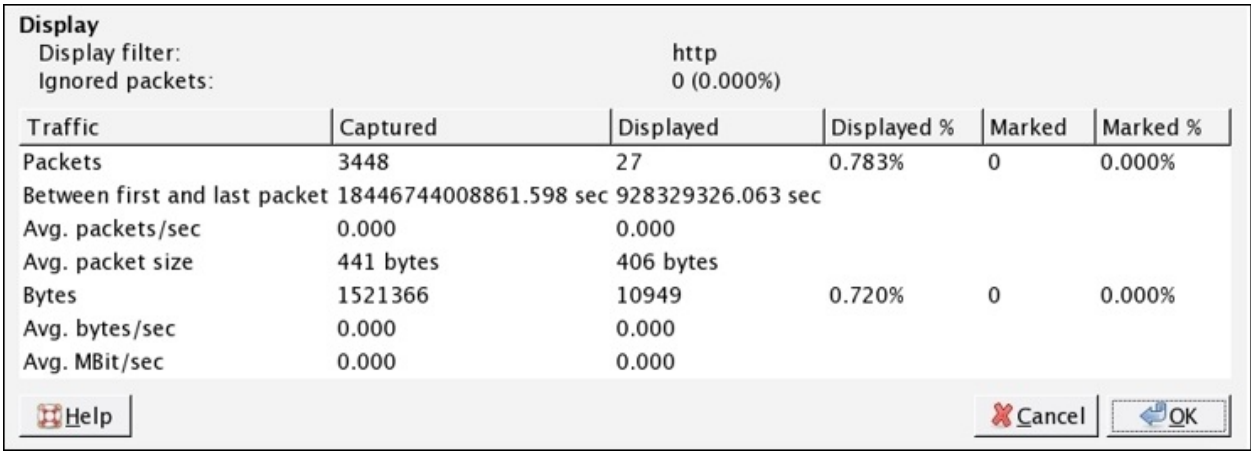


Figure 3.3: With display filter(screenshot 2)

Now, after applying the filter, the variance among the values listed in the stats

can be observed. That is, after applying the display filter **http**, the **Displayed%** column has a different set of values as compared to the previous one without display filter.

Protocol Hierarchy

The Protocol Hierarchy window provides us with an overview regarding distribution of protocols used in the communication process and how to spot unusual activities in your network that do not follow the benchmark as expected. By distribution of protocols, I mean in what percentage a certain protocol has been used in the communication between two hosts, and statistics, for example, how many bytes and packets are being sent and received for every protocol, are collected easily. Any form of unusual activity can be easily figured out by matching our current traffic with the baseline created.

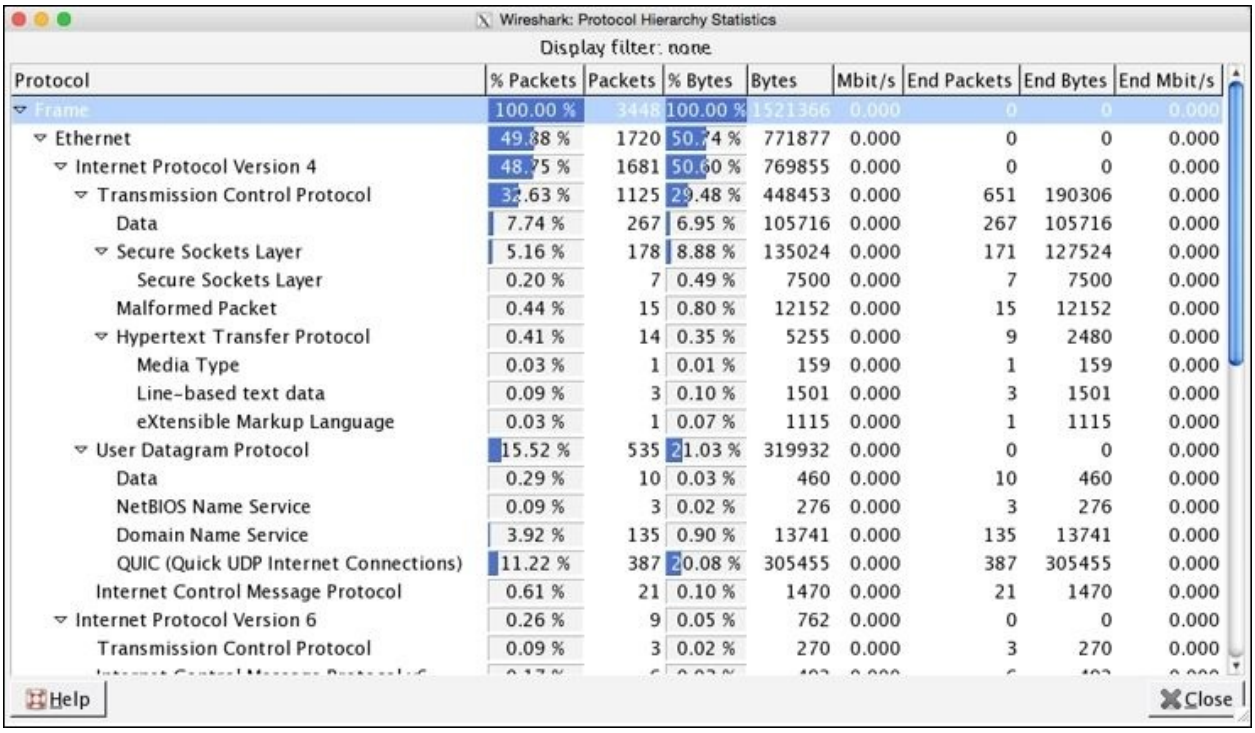
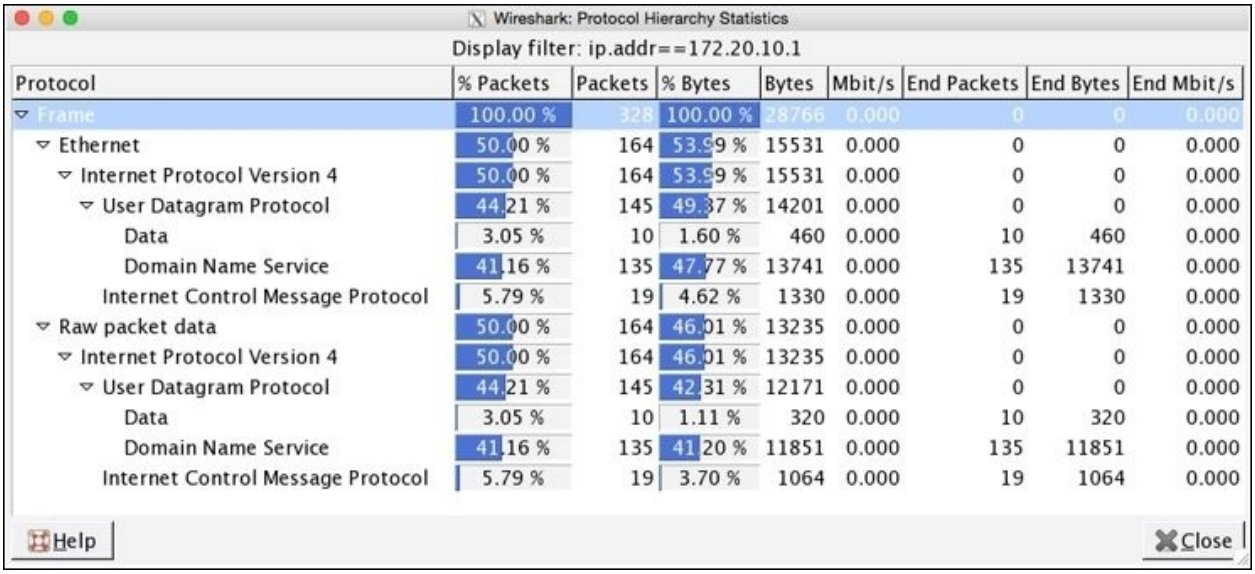


Figure 3.4: Protocol Hierarchy window

If you want to check the protocol distribution for a specific host, then before you open the Protocol Hierarchy window, apply a display filter, for example, **ip.addr==172.20.10.1**. The same filter will be visible at the top of the Hierarchy window just below the title bar. This makes it easy for us to figure out what kind of traffic is actually generated from a certain host, and any malicious traffic from

a certain host can be easily figured out.

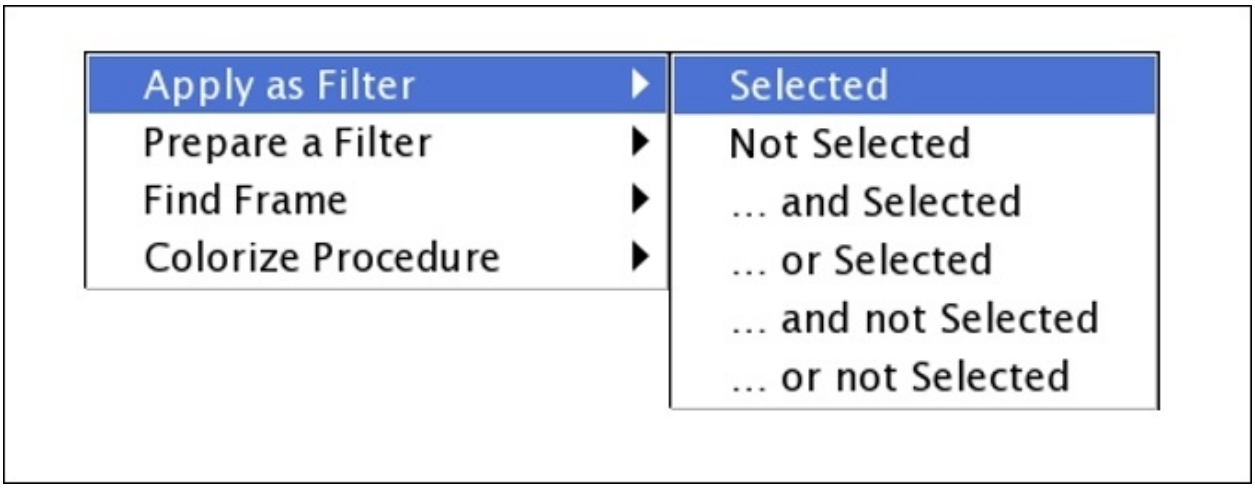
Refer to the following screenshot:



Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00 %	328	100.00 %	28766	0.000	0	0	0.000
Ethernet	50.00 %	164	53.99 %	15531	0.000	0	0	0.000
Internet Protocol Version 4	50.00 %	164	53.99 %	15531	0.000	0	0	0.000
User Datagram Protocol	44.21 %	145	49.37 %	14201	0.000	0	0	0.000
Data	3.05 %	10	1.60 %	460	0.000	10	460	0.000
Domain Name Service	41.16 %	135	47.77 %	13741	0.000	135	13741	0.000
Internet Control Message Protocol	5.79 %	19	4.62 %	1330	0.000	19	1330	0.000
Raw packet data	50.00 %	164	46.01 %	13235	0.000	0	0	0.000
Internet Protocol Version 4	50.00 %	164	46.01 %	13235	0.000	0	0	0.000
User Datagram Protocol	44.21 %	145	42.31 %	12171	0.000	0	0	0.000
Data	3.05 %	10	1.11 %	320	0.000	10	320	0.000
Domain Name Service	41.16 %	135	41.20 %	11851	0.000	135	11851	0.000
Internet Control Message Protocol	5.79 %	19	3.70 %	1064	0.000	19	1064	0.000

Figure 3.5: Protocol Hierarchy window after applying display filter

Using the **Protocol Hierarchy** window, you can create filters too. Just right-click on the protocol you wish to use and then go ahead and specify the expression, as shown in the following screenshot:



There will be situations when a certain host in your network has been breached and you might be observing some unusual traffic associated with a particular host. In such situations, the **Protocol Hierarchy** window will prove worthy.

Conversations

When two devices are connected to each other on the network, they are supposed to communicate; this is considered normal behavior. However, suppose you have thousands of devices connected to your network and you want to figure out the most active device that is generating too much traffic, then in that instance, the **Conversations** window will be quite useful.

To access this nice tool, click on **Statistics** | **Conversations**. After this, you will be presented with a window like the one shown in the following screenshot, which lists various details in terms of several columns listing the packets that were transferred, the bytes that were transferred, the flow of traffic, devices' MAC addresses, and various other details. At the top, you will observe various protocols displayed individually in separate tabs, and along with each active protocol tab, you will notice a number that denotes the number of unique conversations.

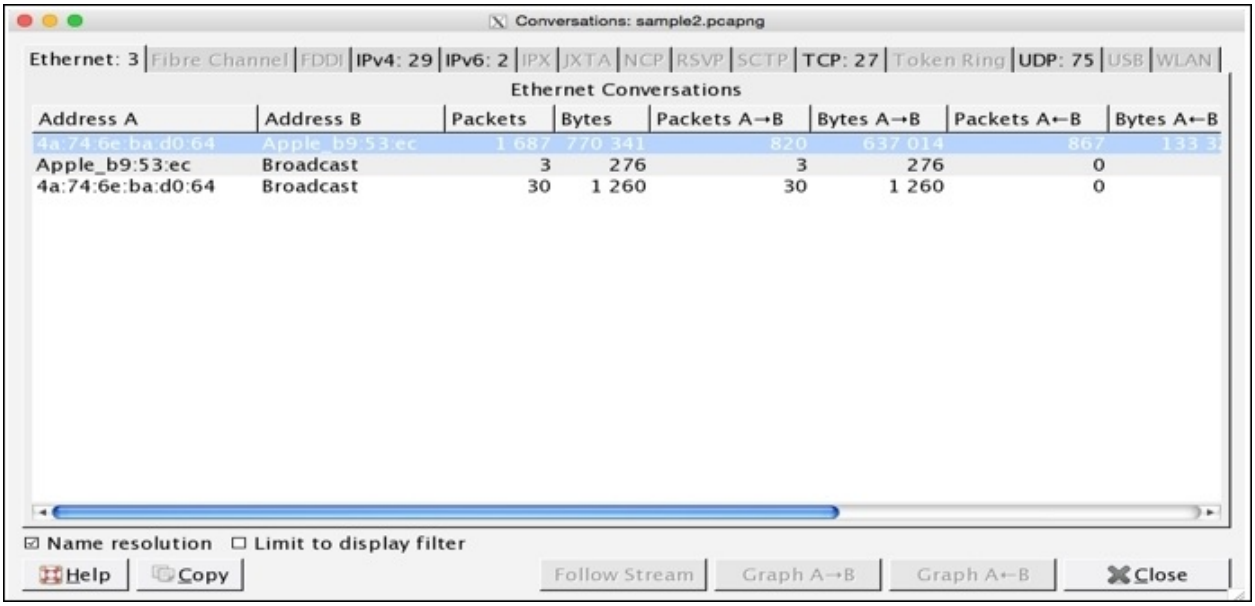


Figure 3.6: Conversations window

For example, if you are looking for the devices that generated a lot of packets and from where major data transfer has happened, then open the **Conversations**

dialog, go to the **IPv4** tab, and sort the packets column in a descending order. Here, the device listed in the first row is your answer. Take a look at the following screenshot that illustrates the same.

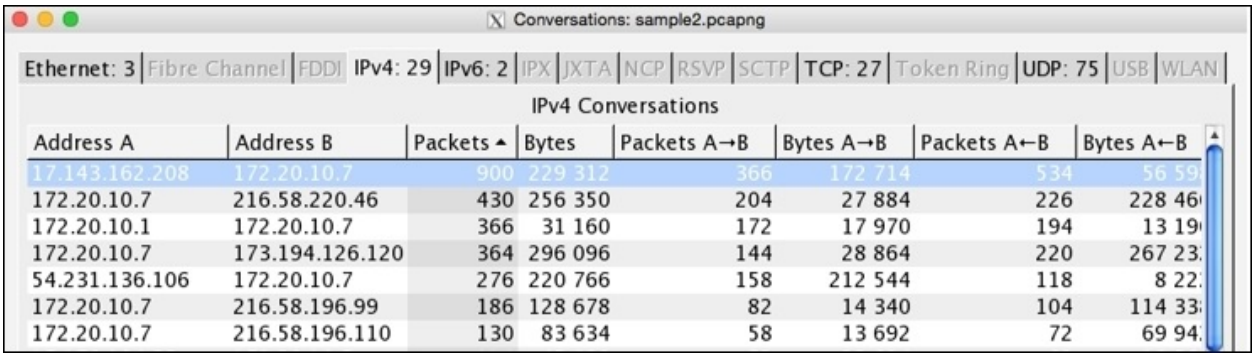
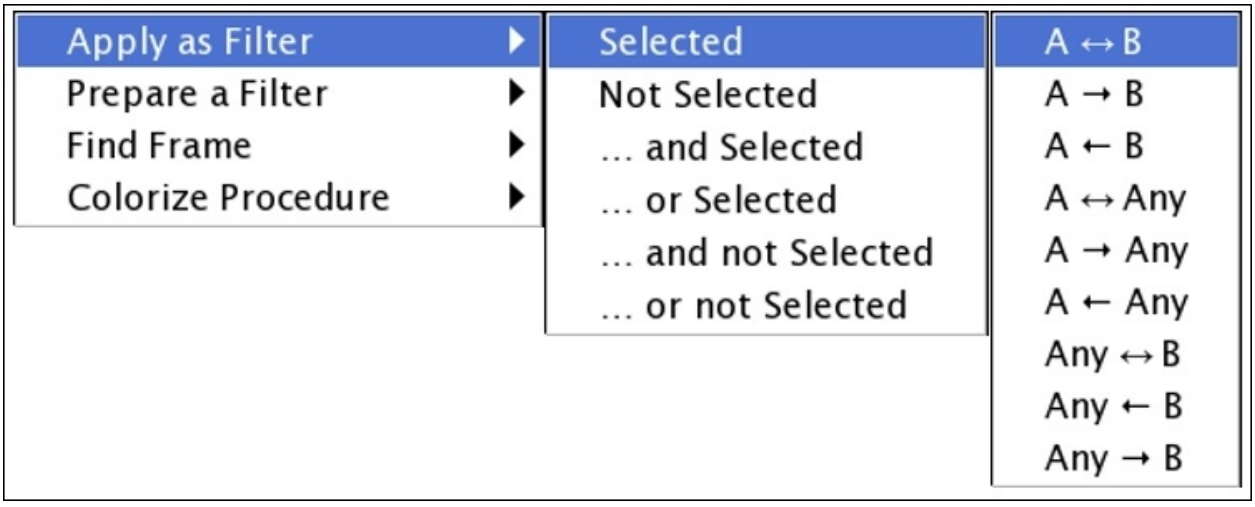


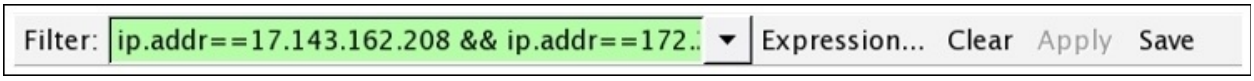
Figure 3.7: Busiest devices

In the first row, we can see how many packets/bytes have been sent and received by each endpoint and the total elapsed duration. If you wish to create a filter for the same, right-click on the first row and then create the respective expression you are thinking about. I chose the first option, **A<->B**, which only shows packets that are associated with Address **A** and Address **B**:



The respective filter will be inserted in the **Display Filter** dialog, as shown in

the following screenshot:



The **Conversations** dialog will let us collect and analyze details in a more granular form, which can be used in various scenarios while troubleshooting and auditing networking infrastructures.

Endpoints

Two devices that share data with each other are often referred to as endpoints with reference to Wireshark. As we have noticed and observed, if a host intends to talk to another host on the network, they would require some form of address to send and receive packets—yes, I am talking about the physical address that every device holds.

Every host is able to communicate with the help of an **Network Interface Card** (**NIC**) that holds a physical address (often termed as a MAC address), and the same address is used for communication over a local network. Devices that communicate in this kind of infrastructure are termed as endpoints. Wireshark gives us the facility of analyzing and collecting information regarding these two devices.

Let's say, for example, that we are observing heavy network traffic flowing across a network, which is kind of unusual according to our daily traffic pattern. Now, we want to figure out due to which device(s) the traffic pattern differs. For us, the **Endpoints** dialog comes to the rescue, which can be accessed from the **Endpoints** menu under **Statistics**, which looks something like the following screenshot. Before you go ahead and open the **Endpoints** dialog, simply click on any TCP packet from the **Packet List** pane. What you will see is a list of tabs visible at the top, each stating a different a protocol. Some of them will be shown as active, and some of them will be shown as inactive because if in your traffic you have a packet relating to a certain protocol, the tab listing that particular protocol will be shown as active; otherwise, it will be shown as inactive.

By default, you will be presented with the **Ethernet** tab (lists the Layer-2 MAC address) in most cases. Along with the protocol, you must observe a number that states the number of endpoints captured for that specific protocol. As in our case, we are seeing **3** and the same number of rows are visible in the **Main** pane.

In the **Main** pane, many more specific details can be seen for every endpoint, such as the total number of packets transferred, total number of bytes transferred, and total bytes and packets received and transmitted for an individual endpoint.

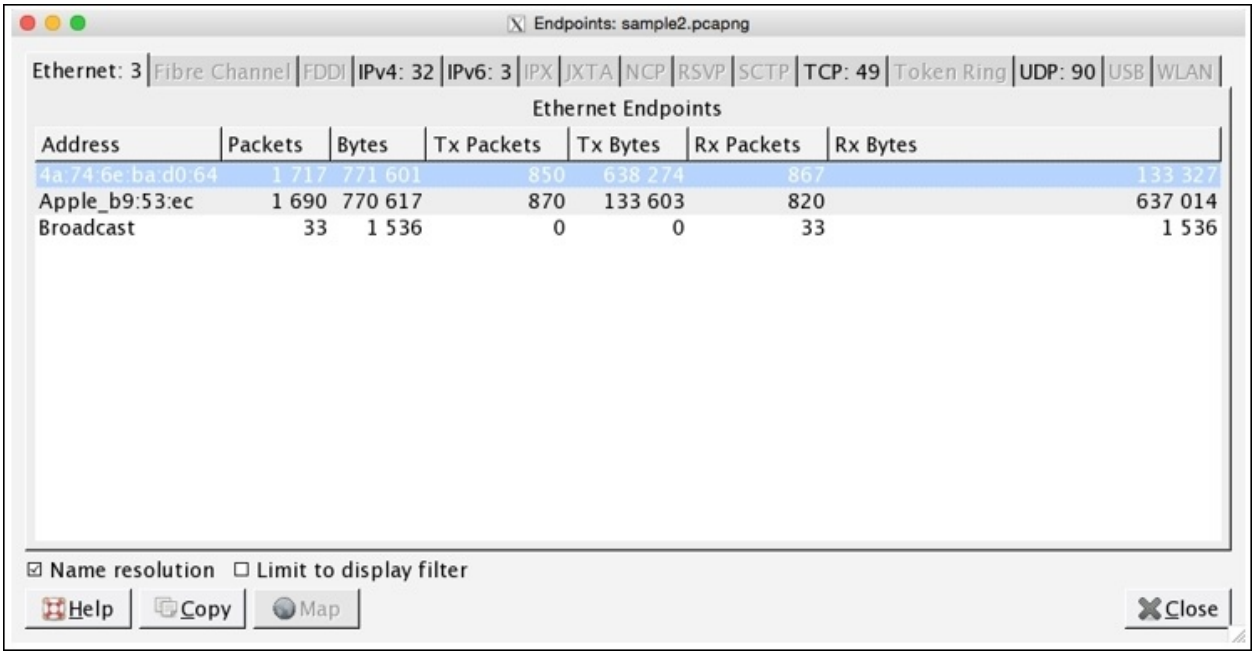


Figure 3.8: Endpoints window

Now, if you want to analyze other protocols, then simply click on any tab of your choice. I clicked on the **IPv4** tab and sorted the main pane using the **Packets** column, which looks like the one shown in the following screenshot: By just looking at the **Endpoints** dialog, I can now easily figure out that maximum data was transferred from IP **172.20.10.7**. This could be a one single IP talking to some server or probably a server talking to multiple machines on our network at a moderate rate.

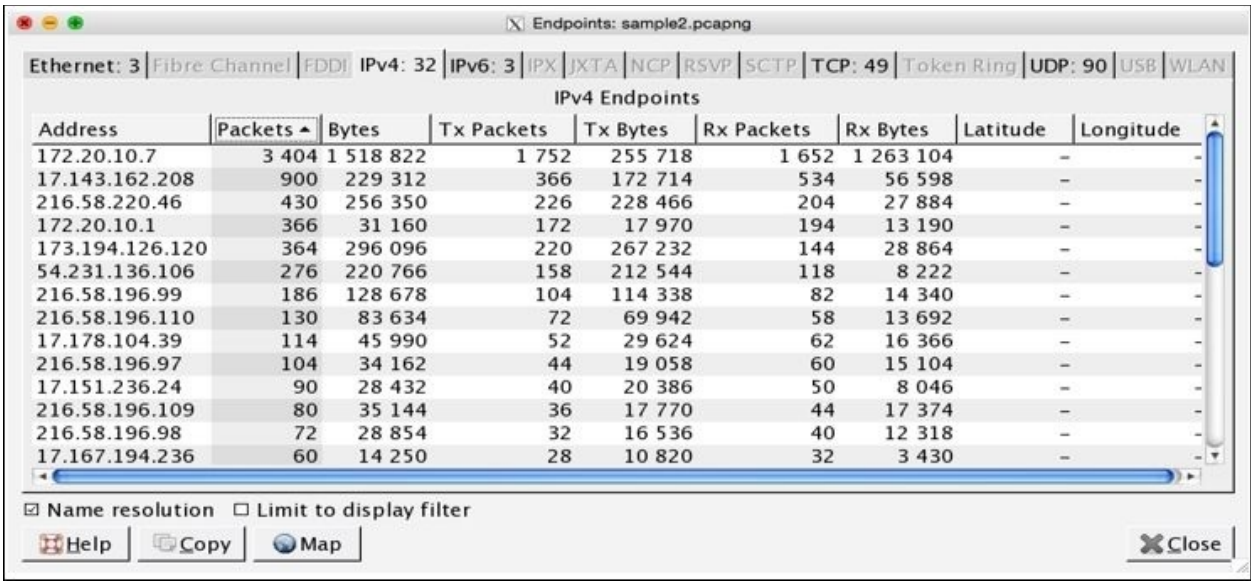
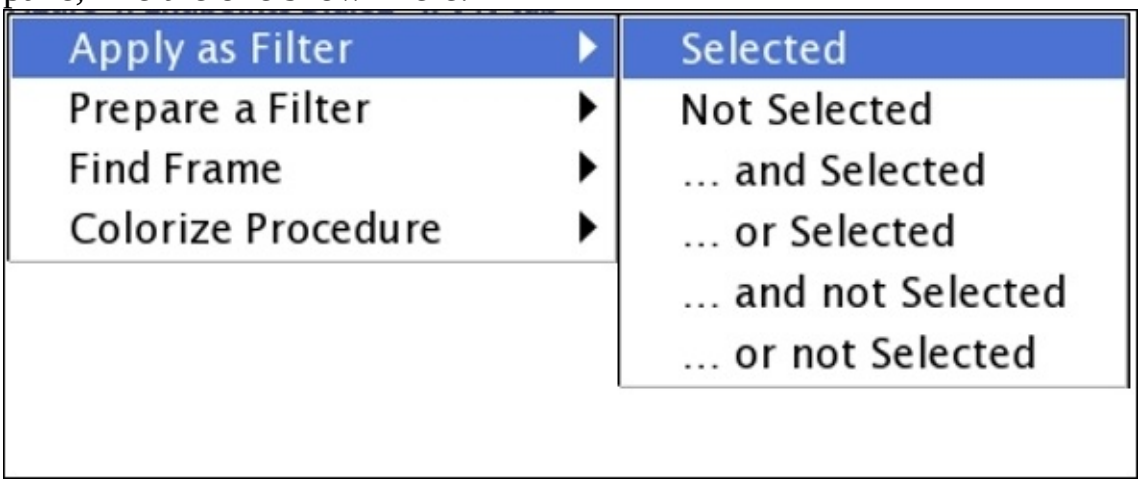
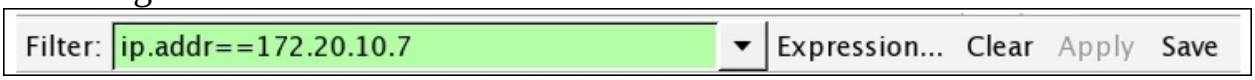


Figure 3.8: Endpoints dialog—IPv4v tab

If you would like to dig more into it, we have an interesting option that can be taken advantage of; simply create a display filter for the same. To do so, right-click on the first row with most packets transferred and choose **Selected** under **Apply as Filter**, as shown in the following screenshot: You will be able to see a display filter for the same **Endpoint** in the **Display Filter** dialog above the **List** pane, like the one shown here:



This facilitates us to quickly analyze traffic for a certain endpoint and hence increases the speed of analysis for users. Once you click on **Clear**, you will be presented with the same **Endpoint** dialog. At the bottom of the window, you will see two check boxes and a few buttons. The purpose of each is listed in the following:



- **Name Resolution**: This resolves the name of each of the Ethernet addresses listed in the **Ethernet** tab. But in some scenarios, it might affect the performance of the application adversely too, for example, when trying to resolve the unique IP addresses from a huge `pcap` file.
- **Limit to display filter**: This limits the results of the Endpoint window on the basis of a display filter that you already applied before accessing the Endpoints window.
- **Copy**: This copies the content of the current Endpoints window tab in a CSV format (comma-separated values).
- **Map**: This maps the selected endpoint's location in your browser on the basis of its actual geographical location.

Working with IO, Flow, and TCP stream graphs

Among various other reporting tools, Wireshark offers graphing capabilities too, which can present captured packets in an interesting format that makes the analysis process much more effective and easy to adapt. The graphing feature is much more effective in comparison to scrolling thousands of packets to figure out the cause of any network-related problem. If you have an overwhelming number of packets to be analyzed, then graphs can be seriously productive. There are multiple types of graphs available that we will discuss, starting with the IO graph.

IO graphs

This is one of the basic graphs that are created using the packets available in the capture file. To create the IO graph, select any TCP packet in your capture file and then click on **IO Graph** under **Statistics**. Refer to the following screenshot:

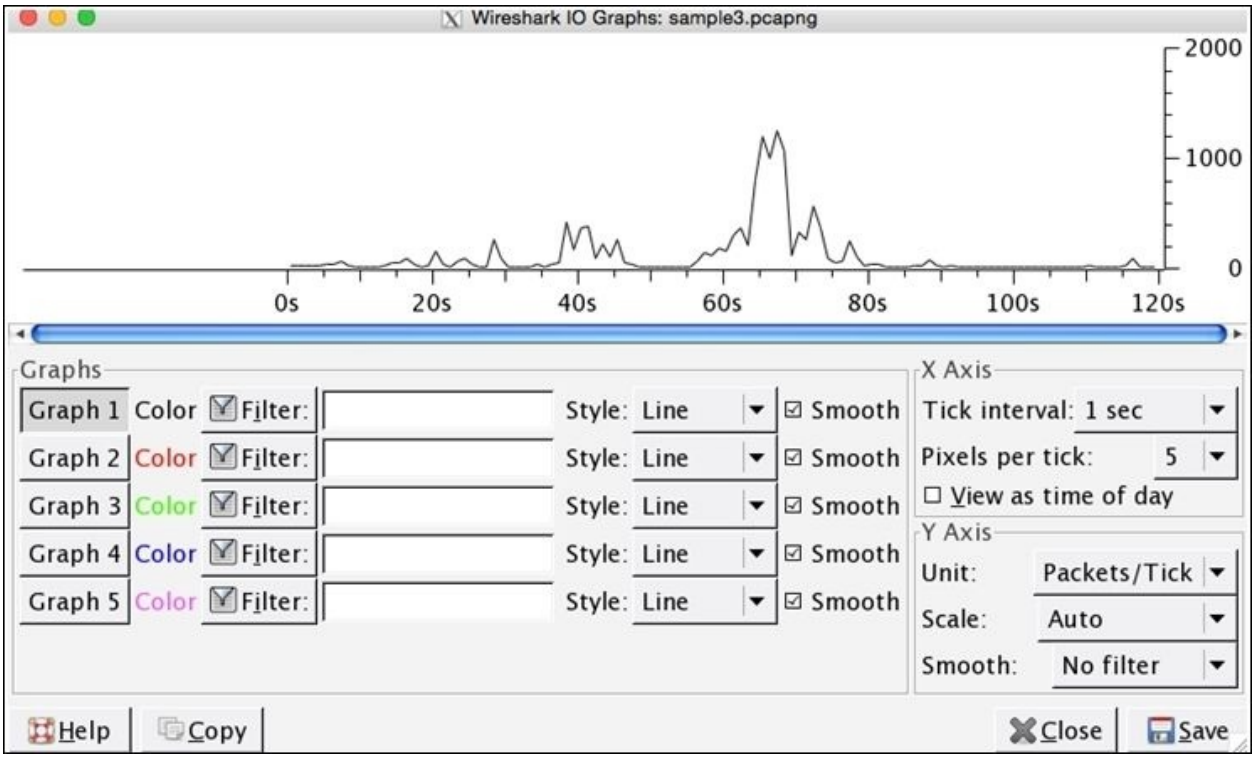


Figure 3.10: IO graphs

This way, you can see the highs and lows in your traffic, which can be used to rectify problems or can even be used for monitoring purpose. In the preceding graph, the data on the x axis represents the time in seconds and the data on y axis represents the number of packets per tick. The scale for the x and y axis can be altered if needed, where x axis will have a range between `10` and `0.001` seconds and y axis values will range between packets/bytes/bits.

From the preceding graph, we can easily depict that between sixtieth to eightieth second of the capture process, the network was most active, which generated

approximately 1000 packets each second of the capture process. Now, you will be realizing how easy it was to gather that specific information from thousands of packets in merely 4-5 seconds; this is what graphing makes you capable of.

Just below the plotted area, you can see the **Graph** section, which lists various tools, such as Graphs 1-5, several filters, and the line format, and various other details. Let's take an example and try to understand the functioning of each of them.

The preceding graph displays the generalized form of our network traffic. Now, my requirement is that I just want to see the frequency of the UDP traffic separately in the same graph plotted with a red line. For such specifications, follow these steps:

- Write `UDP` as a filter in the second filter box from the top
- Click on the **Graph 1** button to deactivate it
- Click on the **Graph 2** button to activate it
- Now, you will see the same window as shown in the following screenshot:

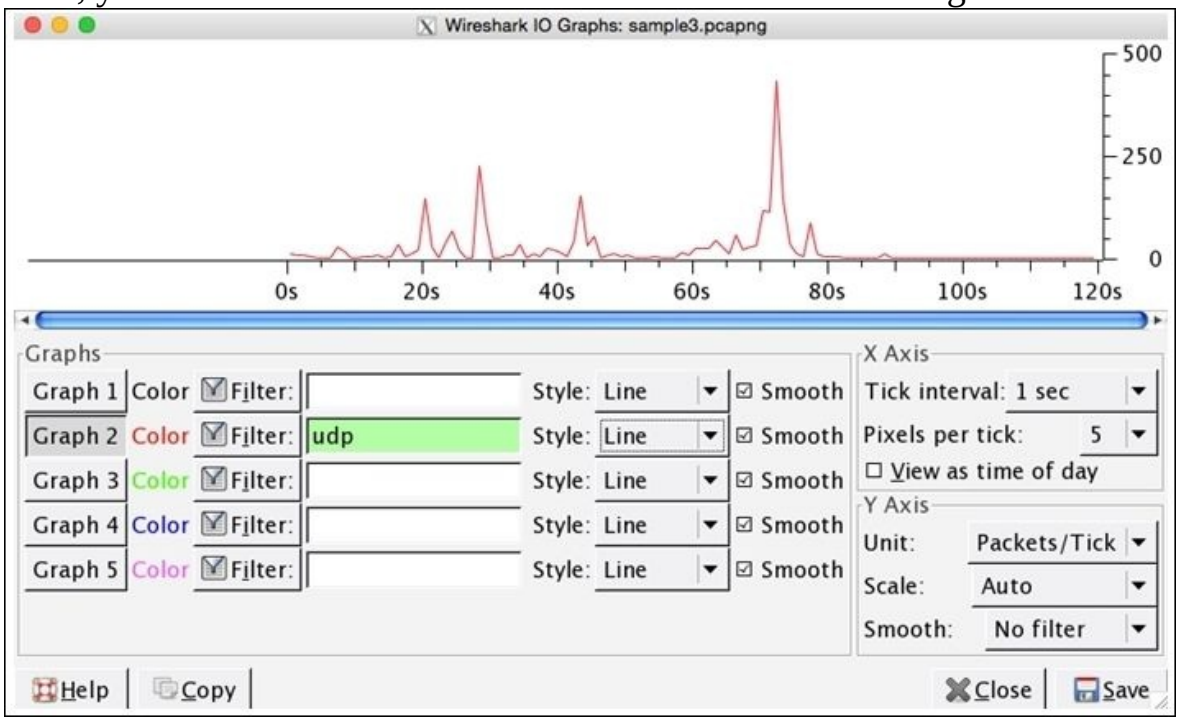


Figure 3.11 : IO graph-UDP traffic only

Analyzing specifically UDP traffic becomes easier in just a few steps. It is clearly visible from the preceding graph that most of the UDP traffic was generated between the seventieth to eightieth second of the capture process, and more than 250 packets were received during the capture process. If you want to compare both TCP and UDP traffic in the same graph, take a look at the following screenshot:

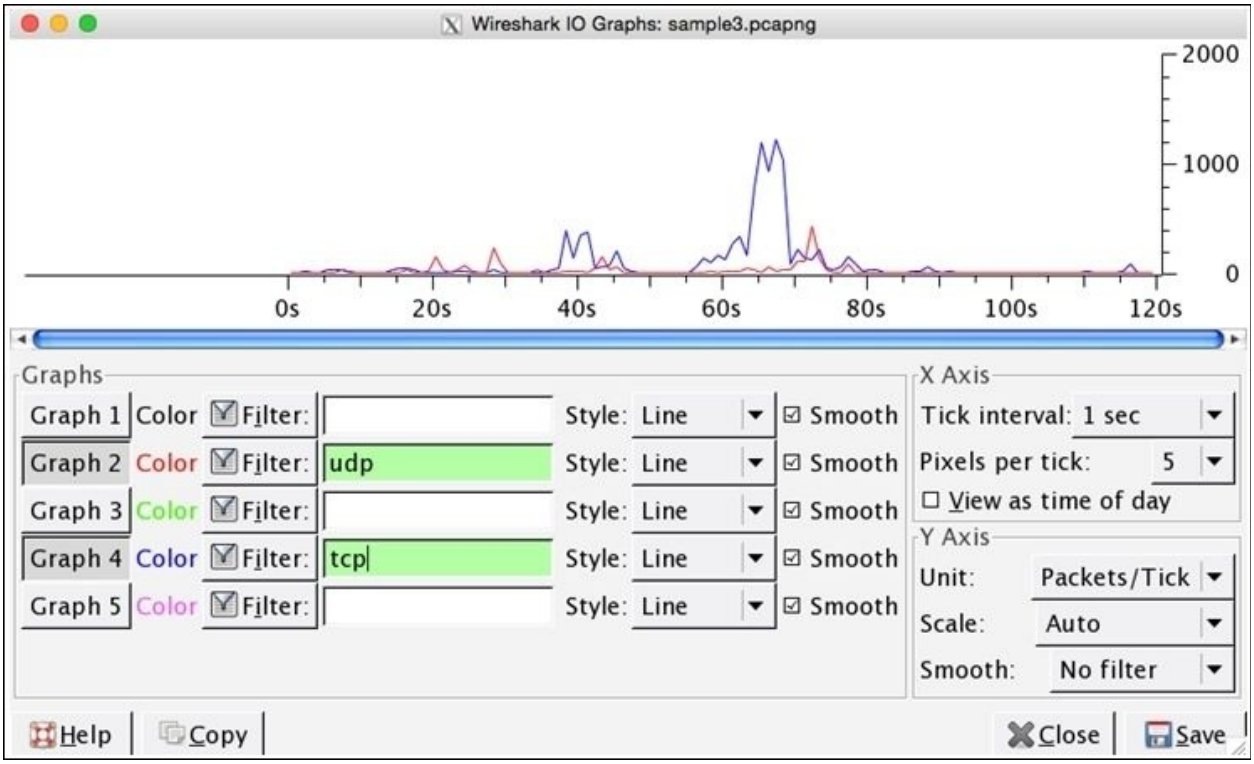


Figure 3.12: IO Graphs—TCP and UDP together

Comparing two things gives us a new angle to view regular things, and generally speaking, the learning process becomes better when we start comparing.

Flow graphs

This is one of the nicest features in Wireshark, where we are assisted with troubleshooting capabilities in scenarios like facing a lot of dropped connections, lost frames, retransmission traffic, and more. Flow graphs let us create a column-based graph, which summarizes the flow of traffic between two endpoints, and it even lets us export the results in a simple text-based format. This is the easiest way of verifying the connection between client and server.

For instance, I have a web server running at **172.16.136.1** and a client running at **172.16.136.129**. The client will request the web server for a certain resource. Let's see what the flow graph looks like for such kind of requests. There will be hundreds of packets generated, but we will look only at HTTP packets, just to make the results more confined and understandable. Click on **Flow Graph** under **Statistics**, and then from the pop-up dialog, choose **Displayed Packet**. Click on **OK**. Refer to the following screenshot that illustrates the same:

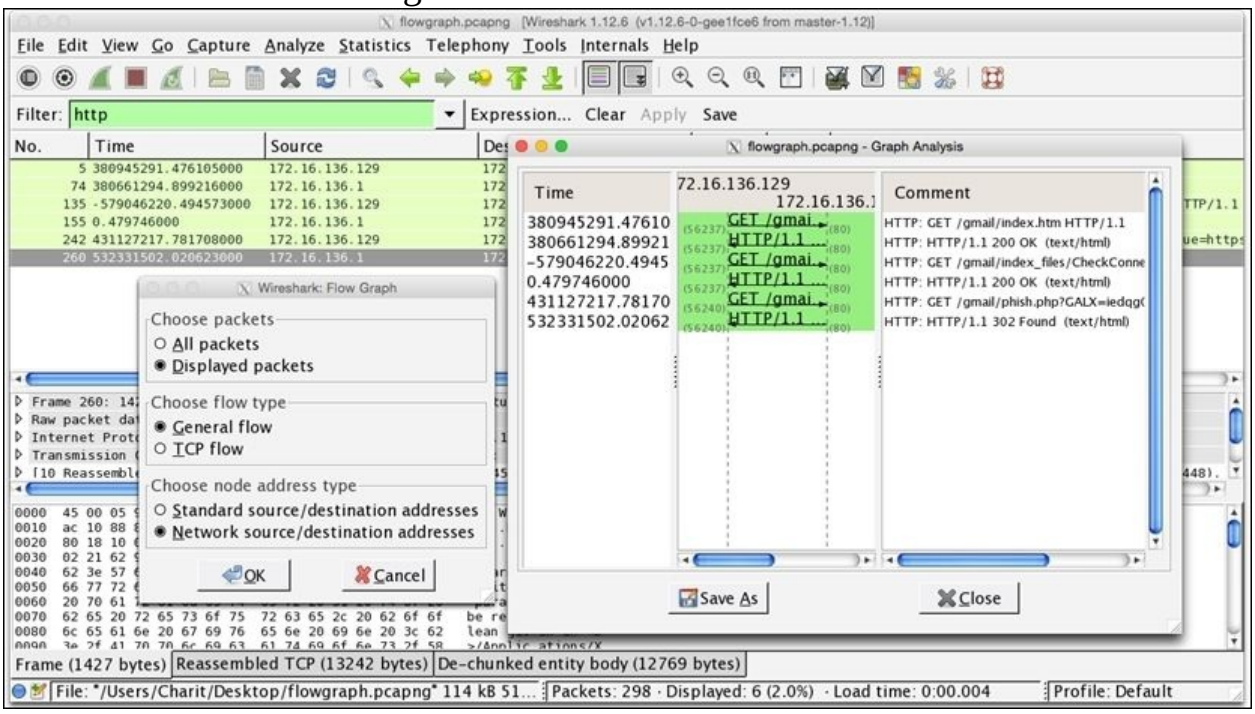


Figure 3.13: Flowgraph

Now, from the **Graph Analysis** window, we can see at what time a certain request was made and what response did we receive, which TCP port was used, along with some plain English comments, and the flow of traffic is also marked. This makes it simple for us to understand how TCP packets flow around.

TCP stream graphs

There are a couple of graphs that come in this section. Each of them depicts the network traffic in a graphical form differently. Let's start by taking a look at each one of them.

Round-trip time graphs

Round-trip time (**RTT**) is the duration in which the ACK for a packet that is sent is received, that is, for every packet sent from a host, there is an ACK received (TCP communication), which determines the successful delivery of the packet. The total time that is consumed from the transfer of the packet to the ACK for the same is called round trip time. Follow these steps to create one for yourself:

- Select any TCP packet in your **packet list** pane.
- Navigate to **Statistics** | **TCP Stream Graph** | **Round Trip Time Graph**.
- The x axis represents the TCP sequence number and the y axis represents the RTT in seconds.
- Each plotted point on the graph represents the RTT of a packet. If you are not seeing anything in your graph, then you might have selected an opposite directional packet.
- RTT graphs are often used by network admins to identify any congestion or latency that can make your network perform slowly.
- To investigate further, just click on any plotted RTT dot in your graph, and Wireshark will point you to that specific packet in the list pane.

The following RTT graph represents normal web traffic, and at some points in the graph, latency can be observed:

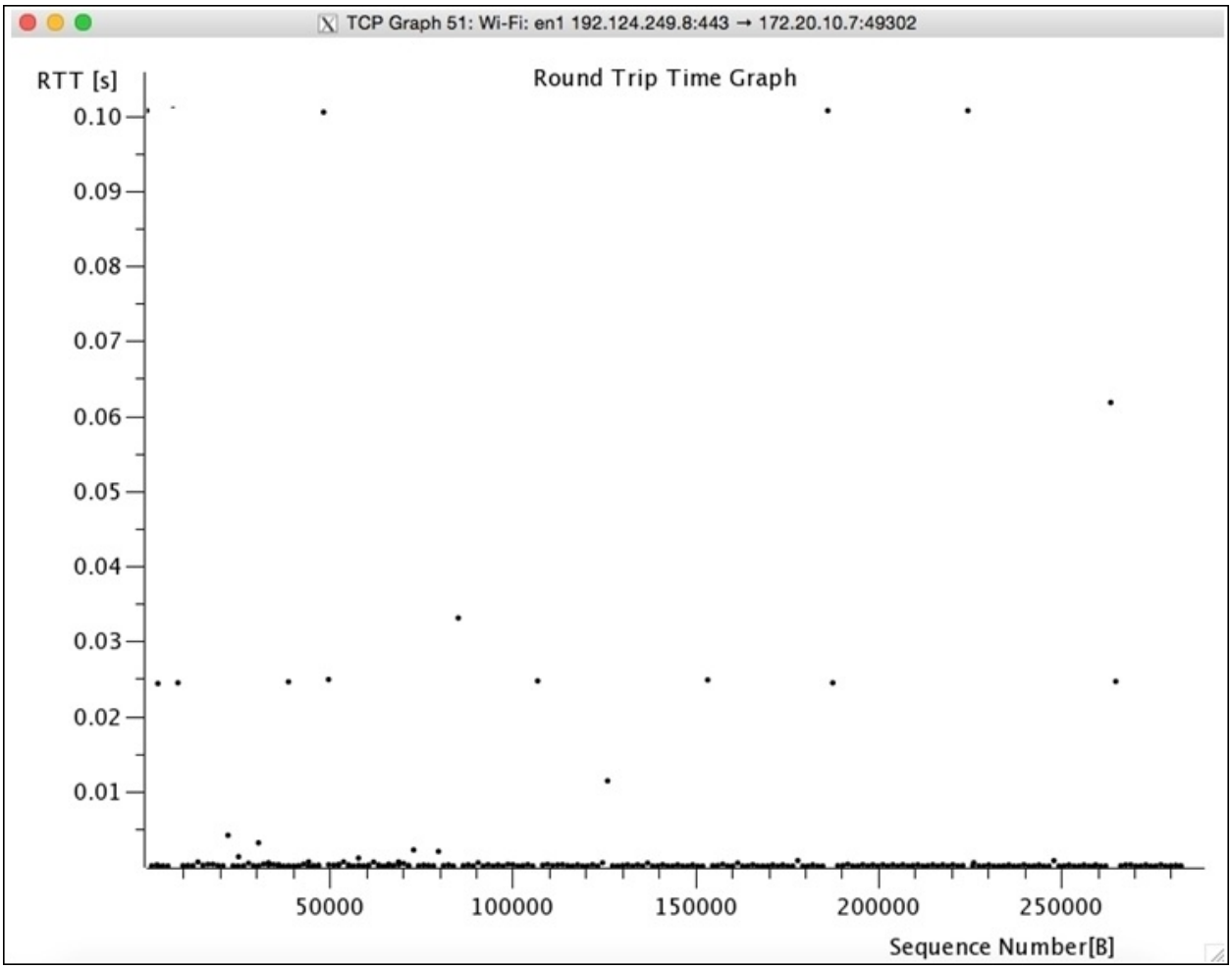


Figure 3.14: Round Trip time Graph

Bottleneck and latency can often be identified with a vertical line of plotted RTT dots, which depicts whether the packet from the sending device is first queued up and then sent all at once or whether the packets are suffering with duplicate ACKs or packet loss, where retransmission was required, thus increasing the RTT time.

Throughput graphs

This graph is very similar to the IO graph that depicts the traffic flow. However, it is different in one important aspect that Throughput graphs depict the unidirectional traffic whereas IO graphs depict the traffic in both directions. For every TCP packet that you select in the list pane, the Throughput graph can be different. If you are seeing a blank graph, then just select another TCP packet and try to create the graph again. Follow these steps to create one for yourself:

1. Open the trace file that contains your packets.
2. Apply a display filter if required.
3. Select any TCP packet from the list pane.
4. Navigate to **Statistics** | **TCP Stream graphs** | **Throughput graph**.
5. Voila! It's done.

In the title bar, the IP address of the communicating hosts is present, along with the direction of traffic. The x axis represents the time in seconds, and the y axis represents throughput in bytes/seconds. Refer to the following graph (*Figure 3.15*) that illustrates the same:

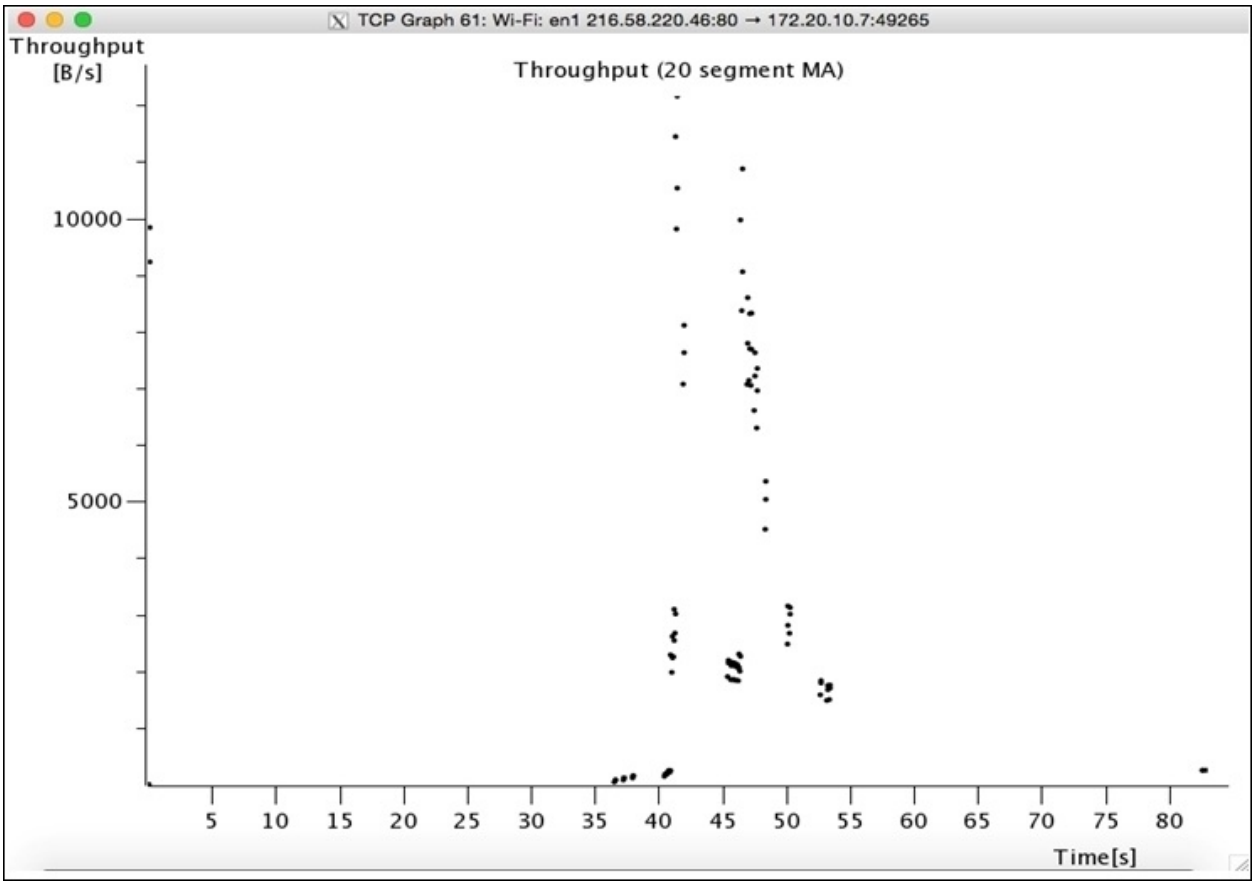


Figure 3.15: Throughput Graph

The Time-sequence graph (tcptrace)

This graph depicts the stream of TCP data over time. The traffic that will be presented is unidirectional (moving in one direction). Time-sequence graph gives us an idea about the segments that are currently traveling, the acknowledgements for segments that we've received, and the buffer area that the client is capable to hold. To create this graph, follow these steps:

1. Open the capture/trace file you want to work with.
2. Click on any TCP packet from the list pane.
3. Navigate to **Statistics | TCP Stream Graphs | Time sequence graph(tcptrace)**.
4. You must now see something like the following:

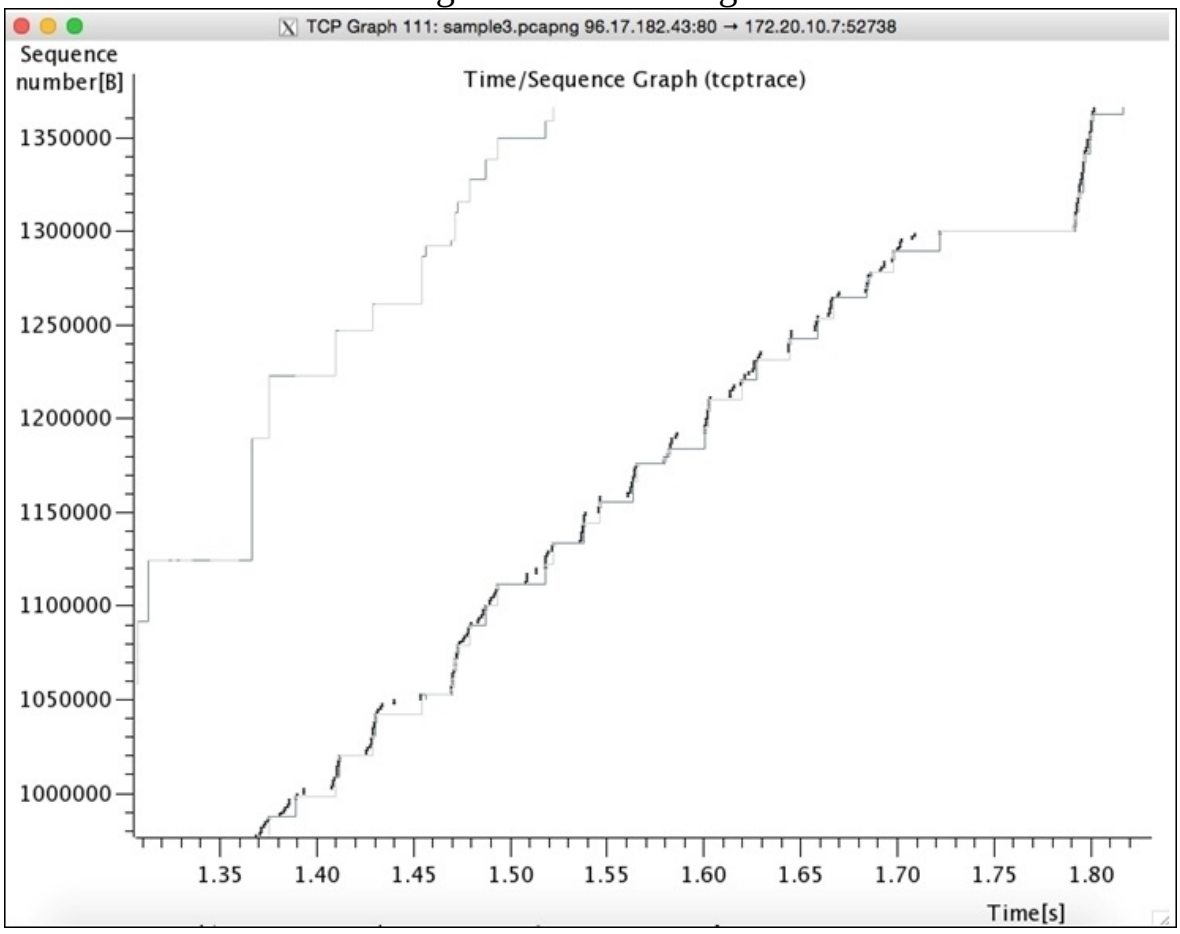


Figure 3.16 : Time Sequence graph (tcptrace)

The x axis of the graph represents the time in seconds and the y axis represents the TCP sequence number. TCP sequence numbers are incremented by the bytes of data sent with every packet, that is, if the sequence number is 1 and the packet we are sending holds 10 bytes of data, then the sequence number will be incremented by 10. Hence, the sequence number for the next packet to be sent will be 11. The throughput of the data is more when we have steeper lines plotted, normally, the graph plotting starts from the lower-left corner to upper-right corner.

There are actually three lines plotted on every graph. The line with multiple **I** written is the TCP data segment, and the longer the **I** stream, the more the data in the packet. The line below the TCP segment is the ACK stream for data sent, and the line at the top represents the calculated client-receiving window.

The distance between the client-receiving window line and the TCP segment line is the window size. The closer the line, the less data can be buffered, and vice versa. Consider the following zoomed-in screenshot for more understanding:

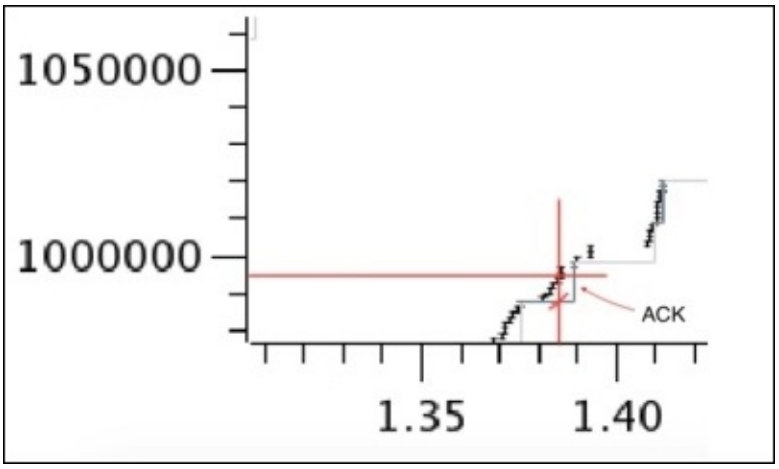


Figure 3.17: Throughput graph

Let's suppose that at 1.38 seconds Host A is sending byte 995,000, and at the same time, host A received an ACK for byte 990,000, which states that 5,000 bytes are still unacknowledged (in-flight). A point to be noted here is that the dark grey lines denote the ACKs received.

Follow TCP streams

Wireshark provides the feature of reassembling a stream of plain text protocol packets into an easy-to-understand format.

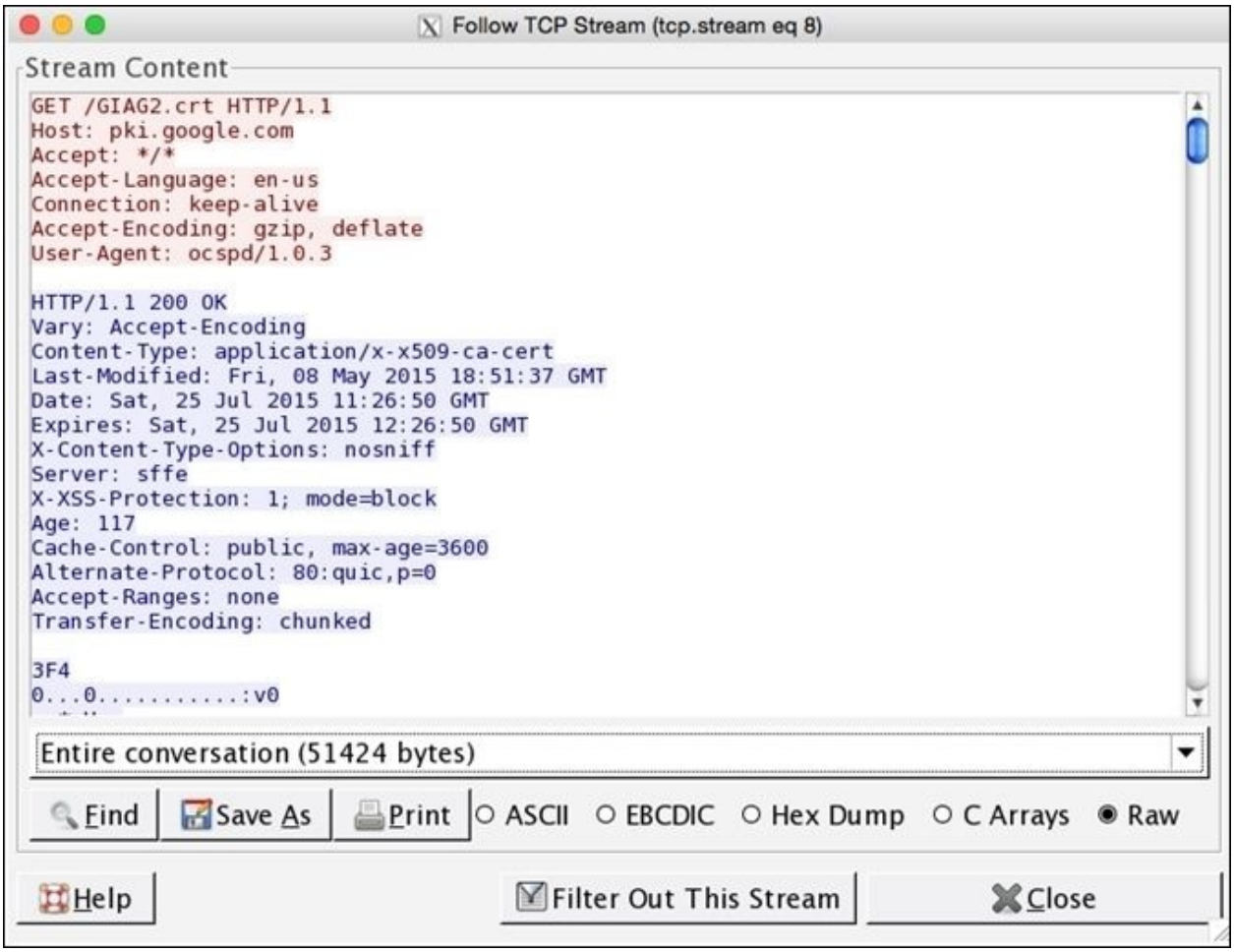


Figure 3.18: Follow TCP Stream window

For instance, assembling an HTTP session will show you the GET requests sent from the client and the responses received from the server accordingly. There is specific color coding that is followed by the requests and responses shown in the Follow TCP stream dialog. Any text in red color denotes a request that a client has sent, and any text in blue color denotes the response received from the server. If the protocol is HTTP, then you can view almost everything in plain

text; if the protocol is HTTPS, then most of the things will be encrypted, hence giving ambiguous text on the screen (there is a way to decrypt HTTPS traffic too, which we will discuss in the upcoming chapters). The Follow TCP stream option can be of great help while troubleshooting any HTTP session, which is the same with most of the application layer protocols.

At the bottom of the dialog, you have a drop-down menu from where you can choose to view either side of communication or you can choose the entire communication, consisting of requests and responses that are shared between the client and the server at the same time. Instead of just viewing the data in RAW format, you can choose between ASCII, EBCDIC, Hex dump, and C arrays format.

If you wish to save the content shown in the dialog, then click on **Save as**, which will save the content in a simple text format. Similarly, to print, you can click on **Print**. And if you want to view everything except the Follow TCP stream packets that you are viewing currently, then click on **Filter out this stream**. To close the dialog, click on **Close**.

To view the TCP stream, follow these steps:

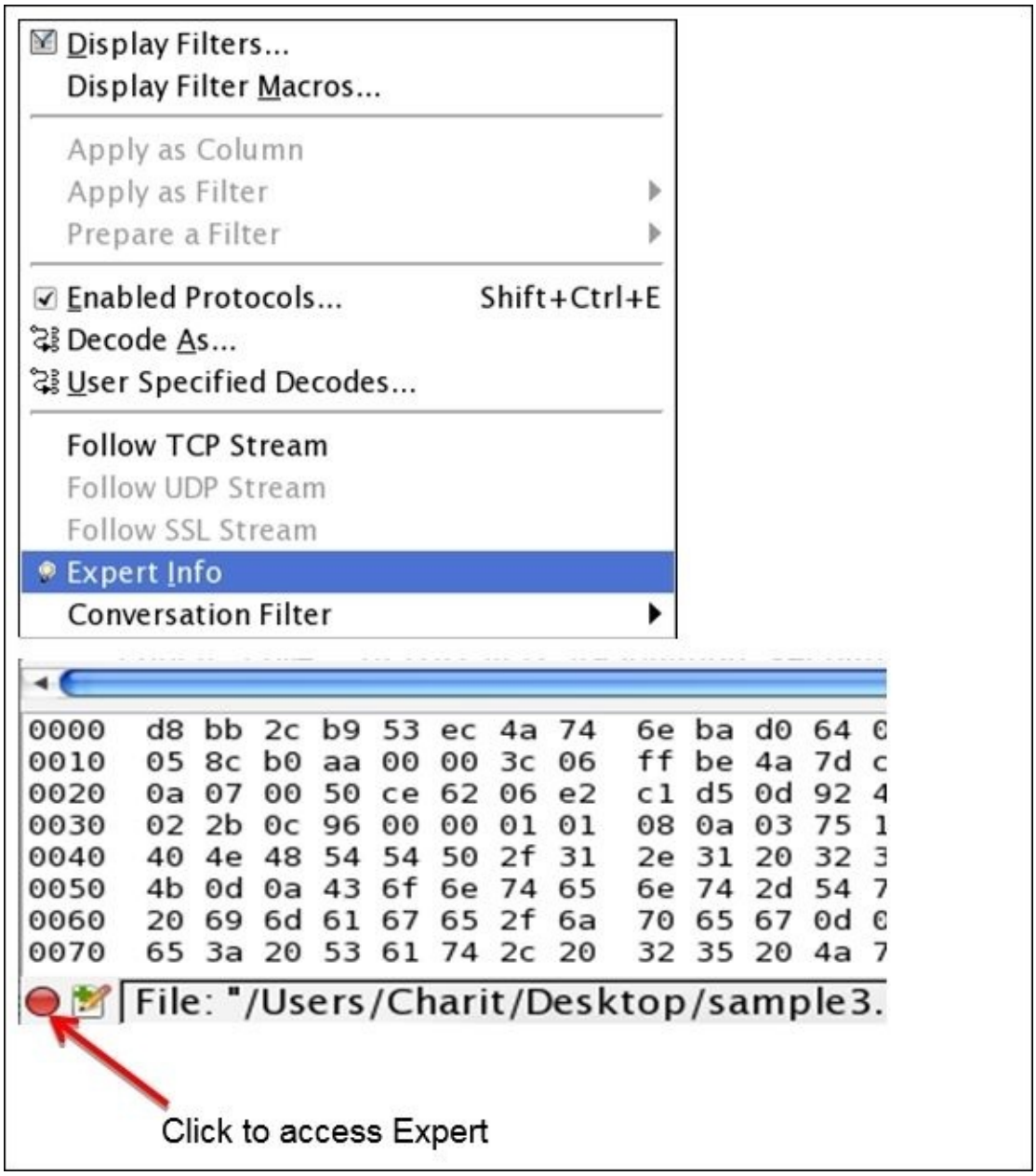
1. Open the capture/trace file.
2. Apply the display filter if required.
3. Select any packet from the list pane.
4. Right-click on the selected packet and click on **Follow TCP stream**.

Following the preceding steps gives a simple view of viewing data. Now, figuring out who initiated the connection will be quite easy.

Expert Infos

The information in the **Expert Infos** dialog is populated by the dissectors that enable the translation of every protocol that is well known to Wireshark. The **Expert Infos** dialog keeps you aware of the specific states that users should know about. Presently, expert infos is available only for TCP-based communication. Maybe for other protocols, the **Expert Info** dialog will be available by the time you read this.

You can access the **Expert Info** dialog by clicking on **Expert Info** under **Analyze**, or you can click on the bottom-left corner on the colored dot just before the status bar. Refer to the following screenshot, which illustrates the



same:

The red dot at the bottom-left corner can be colored with different colors, such as cyan, yellow, green, blue, and grey, where each of them has a specific meaning, which is listed as follows:

- **Red**: This indicates errors
- **Yellow**: This refers to warnings
- **Cyan**: This refers to a note

- **Blue**: This refers to chats
- **Green**: This refers to comments
- **Grey**: This means none

Now, let's have a look at the Expert Infos dialog and discuss various other elements residing within. Refer to the following screenshot for illustration purposes:

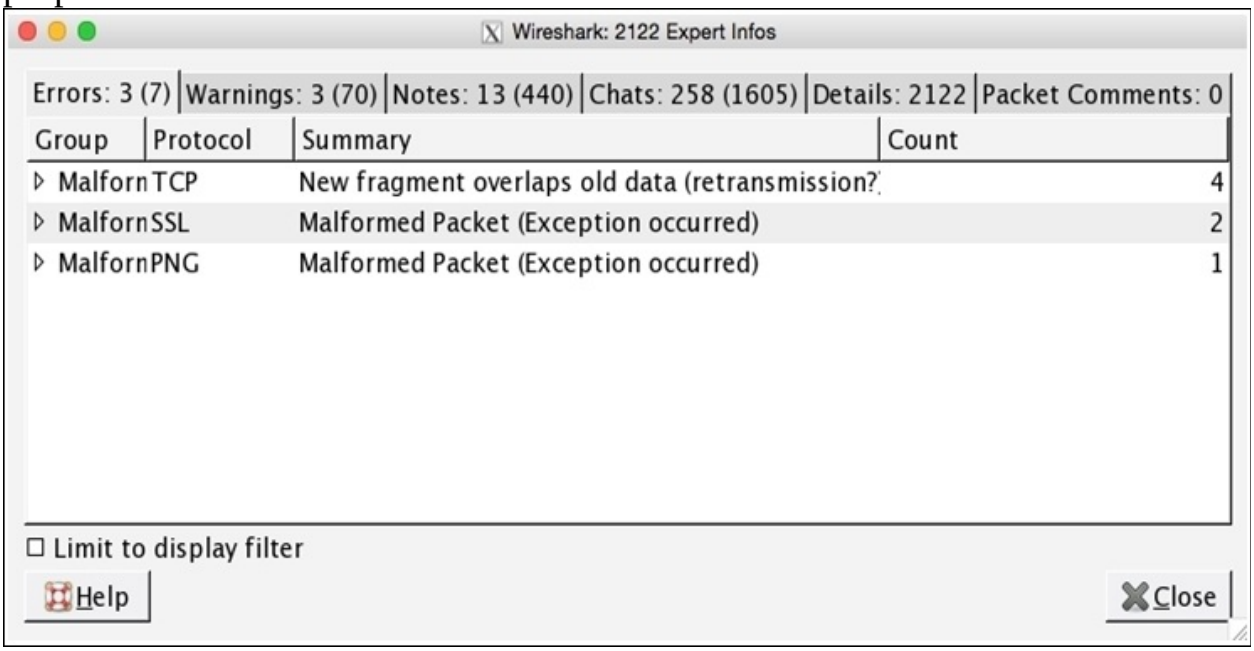


Figure 3.19: Expert Infos dialog

As you can observe, there are multiple tabs listed just below the title bar that consist of packets listed depending on their severity level and category of information. There are mainly four sections in the Expert Infos dialog that point to the likely cause of the problem, so double-checking it will be helpful. Each tab contains the name of the section and two numbers: one inside the parenthesis and one outside. The number inside the parenthesis denotes the total number of packets that have been flagged for the containing category, and the number outside denotes the total number of unique categories for the packets flagged.

We will go through each section one by one, and we will also summarize the criteria by which packets are flagged and listed under different categories, such

as chat, note, warnings, details, and so on:

- **Chat**: These are general messages concerning the current communication. A packet that falls under this section is listed as follows:
  - **Window Update**: This makes the sender aware that the TCP receive window size has been updated.
- **Note**: These are unusual messages that may or may not be part of the current normal communication. Packets that fall under this section are listed as follows:
  - **The Zero Window Probe**: Suppose that the server receiving the packets from the client is not able to process the packets received at the same speed that the client is sending them, thus causing packet loss. In such cases, a server will send a Zero Window packet to the client to halt the process of sending packets for sometime while keeping the connection alive.
  - **The Keep Alive ACK**: The receiver of the Keep Alive packets sends this ACK as a response.
  - **The Zero Window Probe ACK**: This relates to the Zero Window Probe example. The Zero Window Probe ACK will be sent by the client in response to the server's request.
  - **Window is full**: This notifies the sending host that the TCP-receiving window is currently full.
  - **TCP retransmission**: The TCP packet is retransmitted again because of a duplicate ACK, packet loss, or if the timer for retransmission expires.
  - **The duplicate ACK**: If you think about the TCP three-way handshake communication, for every packet received at the other end, the sender should get an ACK packet. If the receiver gets the packet with the sequence number that has already been received, then duplicate ACKs will be generated. This will happen in case of packet loss as well.
- **Warning messages**: These are unusual messages that are probably not a part of your general communication. Packets that fall under this section are listed as follows:
  - **Zero Window**: These messages have been observed when the receiving side tries to notify the sender to stop sending for a while as the TCP-receiving window is full.
  - **Keep Alive**: These messages will be observed when any Keep Alive messages have been captured in the communication.

    - **ACKed Lost Packet**: These messages will be observed when an ACK for some lost packet is received.
    - **Previous Segment Lost**: These messages will be observed when an unexpected packet is received out of sequence.
    - **Out of Order**: These messages will be observed when are packets received in some random sequence, thus signifying no sequence.
    - **Fast Retransmission**: These messages will be popped up when, in a short time of 20 milliseconds, duplicate ACKs have been transmitted again.
- **Error**: These are general error messages in the packets or are thrown by the dissector of a specific protocol translating it. There is no specific category in error messages.
- **Details**: Collectively, all Expert Info dialogs can be viewed in the details tab. However, it is advisable to look into each tab individually on the basis of their severity level. Pointing out the problems can be sometimes easy because the entries made in the **details** tab are lined up in the sequence as they were captured. Viewing anomalies through the details tab can be a bit time consuming and disadvantageous.
- **Packet Comments**: This refers to any annotations given regarding the trace file that can be used to share any interpretations further. Adding comments to the trace file can be really useful while documenting for future references. To add a comment to any packet of your choice, just right-click on the selected packet and click on **Packet Comment**. You will be presented with a dialog where you can add a comment of your choice, and the same comment will be visible in the Packet Comments section of the Expert Infos dialog. Adding a comment will also affect how a certain packet is shown in the Details pane. Generally, an extra field will be added to the details pane highlighted with a green background color.

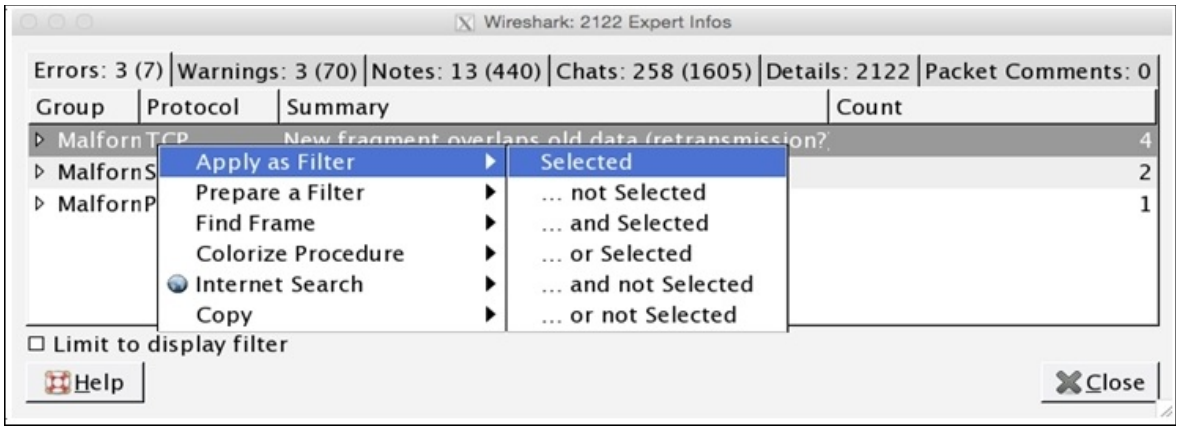
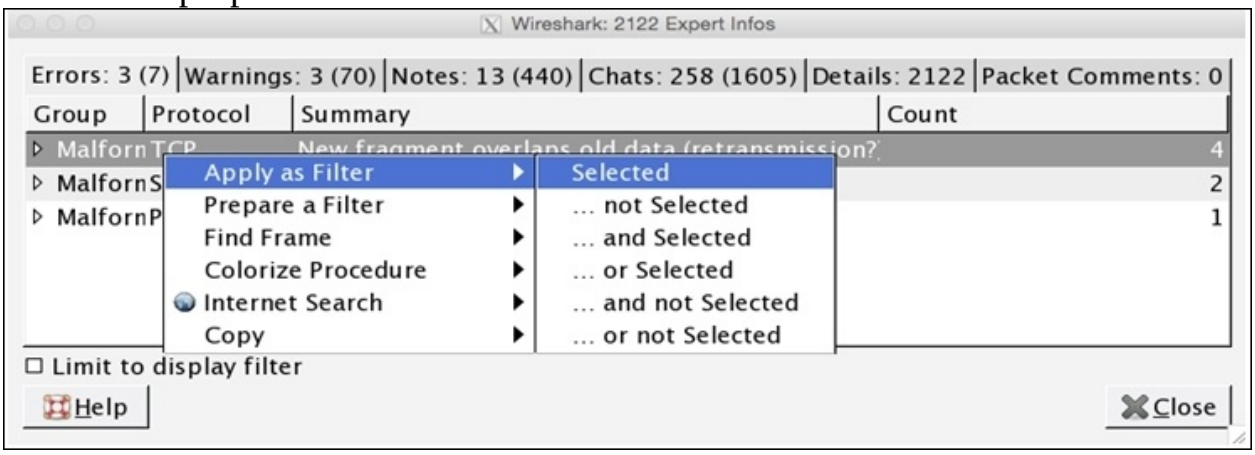


Figure 3.20: Create filter using Expert Infos dialog

Unique categories presented in every section can be expanded to get more information about a specific packet. When you expand and click on the packet listed in the **Expert Infos** dialog, Wireshark will point you to the corresponding packet in the list pane that can be investigated further. Creating a display filter for every category is also possible; just right-click on the selected category and choose the type of filter you want to create. Refer to the following screenshot for illustration purposes:



The main motive of the **Expert Infos** dialog is to find the anomalies present in a trace file. Finding the network problems in the trace file for a novice user becomes a lot easier and faster. Viewing the **Expert Infos** dialog can give a

better idea about the unusual behavior of network packets. As we already discussed, the **Expert Infos** dialog is available for protocols based on TCP/IP; for the rest, there is not much info available.

The best way to figure out juicy info is to look into the tabs separately instead of looking into the **details** tab because, as we discussed, it can be time consuming and can lead to various misunderstandings. Users like you are not supposed to rely completely on **Expert Infos**; sometimes, the file you trace will contain anomalies that won't be listed in the **Expert Infos** dialog. May be, manual analysis will be required as well.

The protocol field that is shown in the details pane of the selected packet will be colored as per the severity level of the **Expert Infos** dialog; take a look at the following screenshot for further reference:

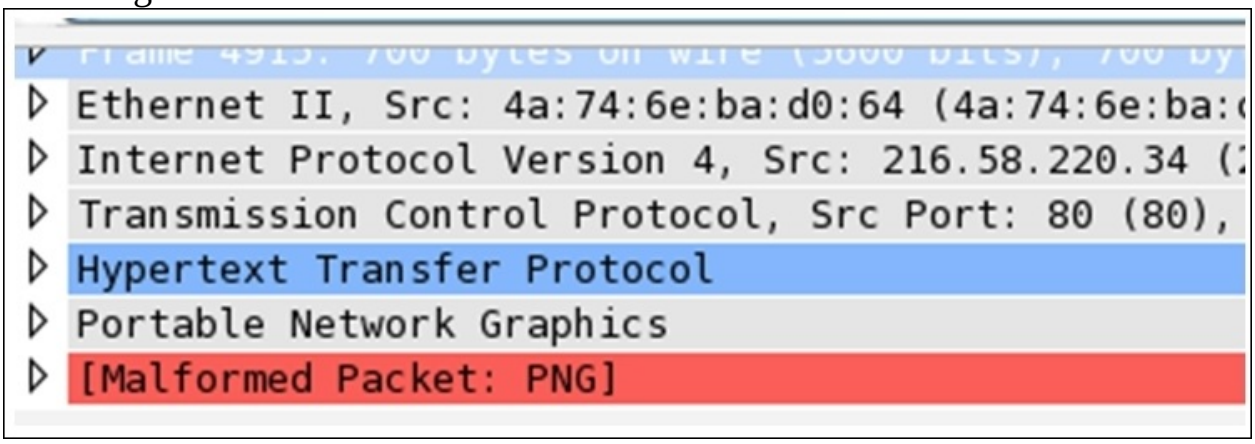


Figure 3.21: Colorization rules in protocol field

We can easily identify from the preceding screenshot that for this particular packet, there is an entry in the Error and Chat sections (red color denotes Error and blue denotes Chats). It is also possible that a single packet is listed in two sections of the Expert Infos dialog.

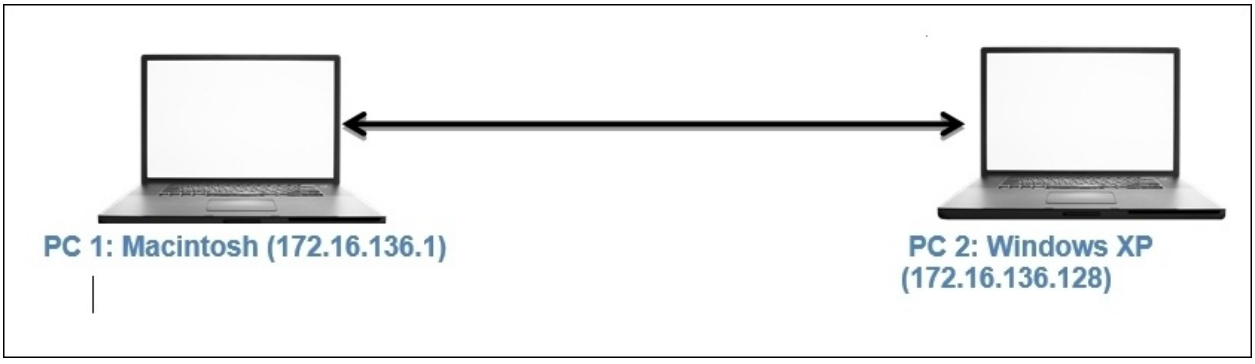
Command Line-fu

With the default installation of Wireshark, there are couple of command-line tools that get installed. These command-line tools are some sort of protocol analyzers, which can be taken advantage of when you don't have a GUI interface to work with or you don't have an option to install the GUI. There are good number of tools available in Wireshark to do this, which are Capinfos, Dumpcap, Editcap, Mergecap, Rawshark, Reordercap, Text2pcap, and Tshark.

The most common and widely used command-line tool for protocol analysis purposes is Tshark, which is capable of capturing data through listening to a live wire, and it can even analyze your already saved trace files. The captured packets are translated into an understandable form and printed to the standard output, or you can save them to the file of your choice. Dissectors that are used by Wireshark the same Tshark utilizes.

Tshark uses the pcap library to capture and translate the packets from the live wire or from the already saved files. Just like Wireshark's filtering option, we can enable filters in Tshark. There are multiple customizable options present in Tshark that can be leveraged to use it in a more advanced fashion.

Wireshark has a CLI version, which is almost similar to Tshark in terms of the syntax and various options that both of them support equally. Let's understand this topic better with an example. Say, for instance, we have an Apache web server and FTP running on a Windows XP box located at `172.16.136.128` and a Macintosh client running at `172.16.136.1`. Using our custom infrastructure, we will generate some network packets and try to use Tshark for capturing and analysis purposes.



When working on a Windows PC, you might have to create the environment variable before you can start using Tshark. The following screenshot belongs to Tshark, displaying `tshark -h (help options)` within the CLI:

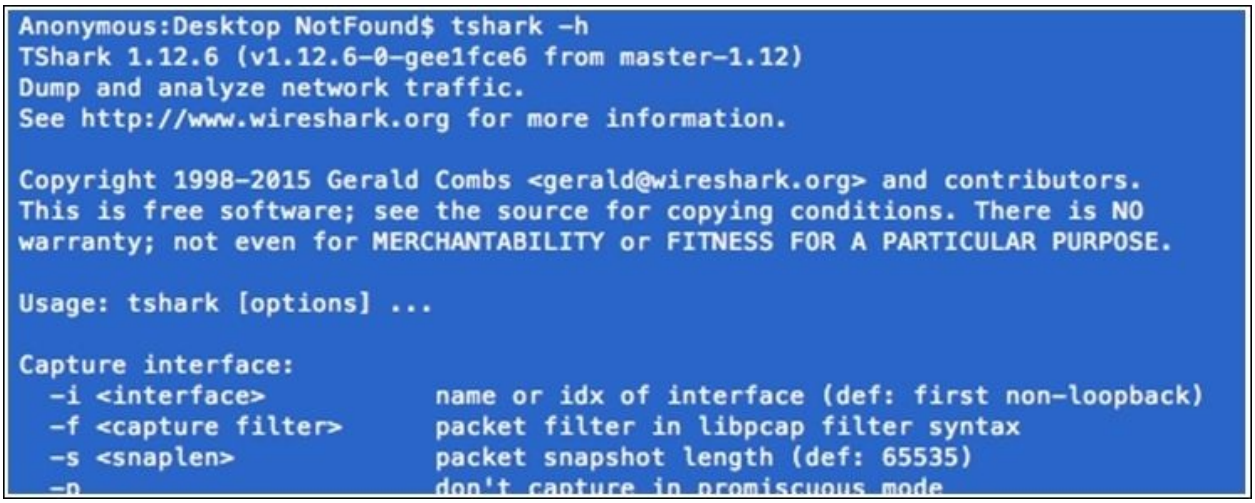


Figure 3.22: Tshark help

We will start with the basics and eventually move toward the creation of filters, and then we will collect statistics using the CLI-based tool Tshark:

- The first thing we should know is how many interfaces do we have available to capture packets. Use the following command to check `tshark -D`:

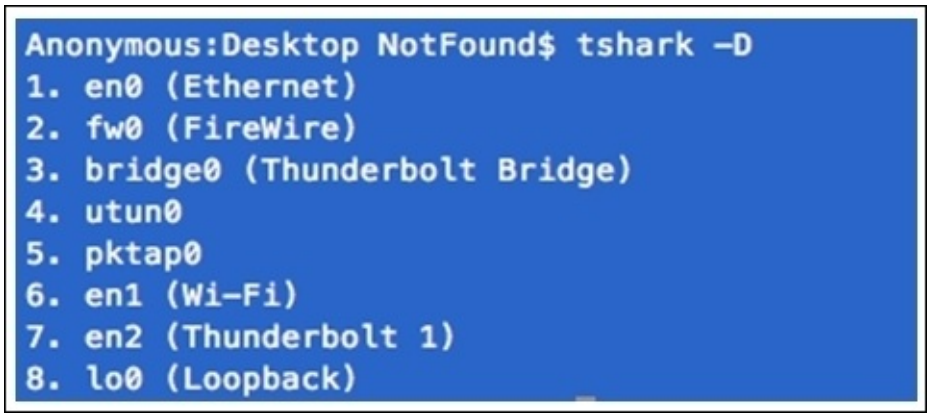


Figure 3.23: Interfaces available

If you do not specify any interface for capturing, `tshark` will choose the first interface that is available on its own. Interfaces can be chosen by their

names and also by the sequence number they appear in. Refer to the preceding screenshot, which shows all the interfaces that are available.

- I have a custom interface `pktap0` that will listen to the connection between my client and the server. So, the command to initiate the capture process will be `tshark –i pktap0 or tshark –i 5`:

```
Anonymous:Desktop NotFound$ tshark -i pktap0
Capturing on 'pktap0'
```

- Now, let's generate some HTTP traffic by visiting the web page hosted on our server from the client (I am using the `curl` command-line tool for browsing purpose):

```
Anonymous:Desktop NotFound$ curl http://172.16.136.128
```

- As soon as the preceding command has been issued, a couple of packets are captured by `tshark` on the `pktap0` interface. And a summary of translated packets for better understandability can be seen. Refer to the following screenshot that illustrates the same:

```
Anonymous:Desktop NotFound$ tshark -i pktap0
Capturing on 'pktap0'
  1   0.000000 172.16.136.1 -> 172.16.136.128 TCP 64 51816→80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS
  2 -745883619.604183 172.16.136.128 -> 172.16.136.1 TCP 64 80→51816 [SYN, ACK] Seq=0 Ack=1 Win=64240
  3 -733373297.062554 172.16.136.1 -> 172.16.136.128 TCP 52 51816→80 [ACK] Seq=1 Ack=1 Win=131744 Len
  4 -1830766245.431098 172.16.136.1 -> 172.16.136.128 HTTP 130 GET / HTTP/1.1
  5 -1830766245.129806 172.16.136.1 -> 172.16.136.128 HTTP 130 [TCP Retransmission] GET / HTTP/1.1
  6 -1664501840.066843 172.16.136.128 -> 172.16.136.1 TCP 52 80→51816 [ACK] Seq=1 Ack=79 Win=64162 Le
  7 -392509417.396438 172.16.136.128 -> 172.16.136.1 TCP 52 [TCP Dup ACK 6#1] 80→51816 [ACK] Seq=1 Ac
  8 -2027256734.439159 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
  9 -179068134.420122 172.16.136.1 -> 172.16.136.128 TCP 52 51816→80 [ACK] Seq=79 Ack=294 Win=131456
 10 -2067155579.763355 172.16.136.1 -> 172.16.136.128 TCP 52 51816→80 [FIN, ACK] Seq=79 Ack=294 Win=1
 11 -1830766248.828112 172.16.136.128 -> 172.16.136.1 TCP 52 80→51816 [ACK] Seq=294 Ack=80 Win=64162
 12 -392509283.614170 172.16.136.1 -> 172.16.136.128 TCP 52 [TCP Dup ACK 10#1] 51816→80 [ACK] Seq=80
 13 -1830766248.686849 172.16.136.128 -> 172.16.136.1 TCP 52 80→51816 [FIN, ACK] Seq=294 Ack=80 Win=6
 14 -392569681.317465 172.16.136.1 -> 172.16.136.128 TCP 52 51816→80 [ACK] Seq=80 Ack=295 Win=131456
```

Figure 3.24: Packets captured at pktap0

If you want to stop the capture process at any point, press *Ctrl + C*.

- To save the translated packets to a file, we need to specify the `-w` switch, along with the command that will save the raw data packets to the specified file:

```
Anonymous:Desktop NotFound$ tshark -i pktap0 -w http.txt
Capturing on 'pktap0'
11
```

A total of 11 packets have been captured, and a text file is being created on the desktop with the `name http.txt`, which will contain raw data as shown in the following screenshot:

```
Anonymous:Desktop NotFound$ cat http.txt


?M<+????????.Mac OS X 10.10.3, build 14D136 (Darwin 14.3.0)4Dumpcap


D136 (Darwin 14.3.0)``???@@E@f?@@k???????lP??f??????
???x``dA???_@@E@?@?},?????P?l?Φ¿?f????a??
@@q??????lP??f??Φ¿?
???xT??4??9??E??@@H???????lP??f??Φ¿?h
???xGET / HTTP/1.1
User-Agent: curl/7.37.1
Host: 172.16.136.128
Accept: */*
```

Figure 3.25: Raw data stored in file

- If you want to save the normal translated form (like the one shown in the list pane in Wireshark), as shown in the standard output, then just redirect the output of the tshark command to a file of your choice, as shown in the following screenshot:

```
Anonymous:Desktop NotFound$ tshark -i pktap0 >> http2.txt
Capturing on 'pktap0'
11
```

As you can see, `11` packets are captured and redirected to the text file `http2`. Let's see what is stored in the `http2.txt` file:

```
Anonymous:Desktop NotFound$ cat http2.txt
  1   0.000000 172.16.136.1 -> 172.16.136.128 TCP 64 51821→80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=32
  2 -1830767469.040043 172.16.136.128 -> 172.16.136.1 TCP 64 80→51821 [SYN, ACK] Seq=0 Ack=1 Win=64240 L
  3 -1830767469.040009 172.16.136.1 -> 172.16.136.128 TCP 52 51821→80 [ACK] Seq=1 Ack=1 Win=131744 Len=0
  4 -2016764535.847514 172.16.136.1 -> 172.16.136.128 HTTP 130 GET / HTTP/1.1
  5 -2027256734.427691 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
  6 -1830767469.037172 172.16.136.1 -> 172.16.136.128 TCP 52 51821→80 [ACK] Seq=79 Ack=294 Win=131456 Le
  7 -1830767469.037084 172.16.136.1 -> 172.16.136.128 TCP 52 51821→80 [FIN, ACK] Seq=79 Ack=294 Win=1314
  8 -1935145592.773838 172.16.136.128 -> 172.16.136.1 TCP 52 80→51821 [ACK] Seq=294 Ack=80 Win=64162 Len
  9 -1830767469.036949 172.16.136.1 -> 172.16.136.128 TCP 52 [TCP Dup ACK 7#1] 51821→80 [ACK] Seq=80 Ack
 10 -1935145592.773838 172.16.136.128 -> 172.16.136.1 TCP 52 80→51821 [FIN, ACK] Seq=294 Ack=80 Win=6416
 11 -1830767469.036570 172.16.136.1 -> 172.16.136.128 TCP 52 51821→80 [ACK] Seq=80 Ack=295 Win=131456 Le
```

Hopefully, by now you must have clearly understood the difference between both ways of saving the raw data packets and translated packets. Both of the techniques can be used in multiple scenarios.

- The next big thing you will learn is the different filters (Capture, Read, and Display) available in Tshark. We know about Capture and Display filters already, but here we have one more category, that is, the **Read filter**. The Read filter is closely similar to the Capture filter, as both of them can filter packets from the live network. However, the Read filter is also capable of filtering packets out of a saved file. Using the Read filter could be processor intensive, and things like packet loss can happen, so think twice before using it. To display the filter, the `-f` switch is used; `-R` is used for the Read filter; and `-Y` is used for the display filter. Now, I am going to capture only FTP packets using the following syntax:

```
Anonymous:Desktop NotFound$ tshark -i pktap0 -f "port 20"
Capturing on 'pktap0'
  1   0.000000 172.16.136.1 -> 172.16.136.128 TCP 64 51852→20 [SYN] Seq=0 Wi
  2   0.000151 172.16.136.128 -> 172.16.136.1 TCP 64 20→51852 [SYN, ACK] Seq
  3 -1438261061.117554 172.16.136.1 -> 172.16.136.128 TCP 52 51852→20 [ACK]
  4 -565845755.905104 172.16.136.128 -> 172.16.136.1 FTP-DATA 94 FTP Data: 4
  5   0.330476 172.16.136.1 -> 172.16.136.128 TCP 52 51852→20 [ACK] Seq=1 Ac
  6 -1438260168.702253 172.16.136.128 -> 172.16.136.1 FTP-DATA 97 FTP Data:
  7 -776735948.749363 172.16.136.1 -> 172.16.136.128 TCP 52 51852→20 [ACK] S
```

While applying a filter, there is a restriction that the filter expression must be specified as a single argument if it has spaces in between. Then, we need to write the expression within double quotes. Refer to the preceding screenshot that illustrates the same.

- Now, let's try to create one display filter using the `http.pcap` file. I want to filter all packets originating from the web server located at `172.16.136.128`

using the `http` protocol.

- First I captured the communication between the client and server. And save the traffic in file `HTTP.pcap`.

  Once I have enough packets to work with, I will apply display filters, as shown in the following screenshot:

```
Anonymous:Desktop NotFound$ tshark -r http.pcap -Y "ip.src==172.16.136.128 and http"
 31 -2027256734.408549 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
 42 -2027256734.408549 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
 71 -1899318681.597223 172.16.136.128 -> 239.255.255.250 SSDP 161 M-SEARCH * HTTP/1.1
 76 -1899318681.597223 172.16.136.128 -> 239.255.255.250 SSDP 161 M-SEARCH * HTTP/1.1
 81 -1899318681.597223 172.16.136.128 -> 239.255.255.250 SSDP 161 M-SEARCH * HTTP/1.1
 90 -1899318681.597223 172.16.136.128 -> 239.255.255.250 SSDP 161 M-SEARCH * HTTP/1.1
467 -2027256734.408549 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
619 -2027256734.408549 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
653 -2027256734.408549 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
1925 -1830772787.988137 172.16.136.128 -> 172.16.136.1 HTTP 345 HTTP/1.1 302 Found
```

Figure 3.26: Tshark display filter

- Suppose you want to quickly collect statistics about the http protocol from the `http.pcap` file. For such a requirement, we can use this command:
  `tshark –r <file-name> -q –z <expression>`

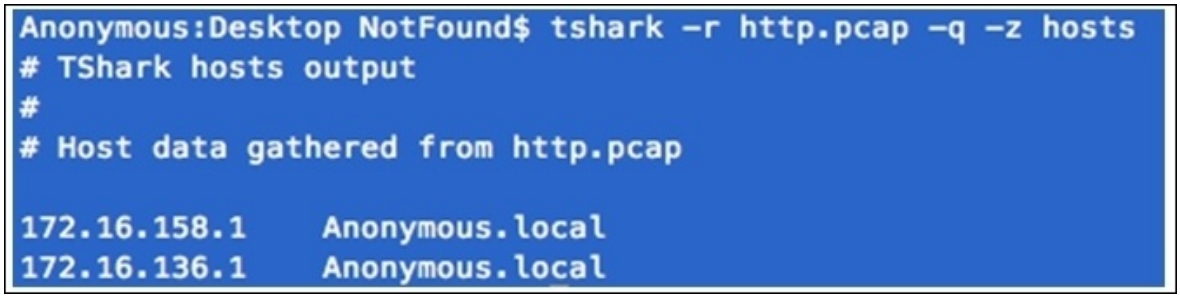
```
Anonymous:Desktop NotFound$ tshark -r http.pcap -q -z http,tree

================================================================================
HTTP/Packet Counter:
Topic / Item           Count         Average       Min val       Max val       Rate (ms)     Percent
--------------------------------------------------------------------------------
Total HTTP Packets     17                                                       100%
 HTTP Request Packets  11                                                       64.71%
  GET                  7                                                       63.64%
  SEARCH               4                                                       36.36%
 HTTP Response Packets 6                                                       35.29%
  3xx: Redirection     6                                                       100.00%
   302 Found           6                                                       100.00%
  ???: broken          0                                                       0.00%
  5xx: Server Error    0                                                       0.00%
  4xx: Client Error    0                                                       0.00%
  2xx: Success         0                                                       0.00%
  1xx: Informational   0                                                       0.00%
 Other HTTP Packets    0                                                       0.00%
```

  The `-q` switch keeps it silent over the standard output (this is generally used while working with statistics in Wireshark) and the `–z` switch for activating various statistics options available. Both of these switches are often used together.

- Let's take one more simple example before wrapping this up; from the http.pcap file, I want to figure out how many hosts there are in total during the whole capture time. For such a requirement, refer to the following

screenshot:



Here, you learned about the basic theoretical and practical concepts of the CLI utility Tshark, along with how to capture and filter data as per our requirements. With the help of Tshark, it becomes really easy to understand how protocols work; we saw various techniques to collect and analyze the packets. Statistical features in Tshark are rich, which helps a moderate user become advanced with an better understanding of how to analyze network packets.

Summary

The Statistics menu in Wireshark contains options that can give us insight from a unique perspective. In this chapter, we've discussed features such as Summary, Conversations, Endpoints, and Graphs.

Summary is an informational feature, which offers a granular form of data, filters, and the trace file that you are working with. The Conversations window details data regarding the communication that happens between two or more hosts. The Endpoints dialog gives an overview of the devices connected to the network and communicating. The Protocol Hierarchy window gives an idea about the protocols being used in the communication, that is, it gives us a picture of the distribution of protocols used by the hosts for communication.

Graphs are a pictorial way of representing the statistics regarding packets. We can easily figure out if something is wrong with our network; we can match network performances and troubleshoot general day-to-day problems that occur.

IO graphs tell us the basic status of a network, and let us create filters. Matching network performances and differentiating a specific protocol becomes easy due to these. The Flow graph depicts the flow of data in a column-based manner and creates a simple interface to understand the flow of packets in a network. TCP stream graphs are a couple of types, but their objective is to depict the throughput of our network, that is, to know how much data is traveling over a particular period of time.

Using the Follow TCP Stream option, you can reassemble the packets listed in a raw data form, which can be easily read. There are different options that are available to change the form to ASCII, Hex, and many others.

The Expert Infos dialog tells you the information that can be usual and unusual. All of them are related to your packets; information is generated with the help of protocol dissectors, which translate the packets to a normal form, and if they find something unusual, then it will be listed in a section and under a category inside the dialog.

Command-line tools also get installed when you install Wireshark. The most

common tool used is Tshark, which works in a similar way to Wireshark and tcpdump. It uses the pcap library that is used by other major protocol analyzers. With tshark, you can listen to live networks or work along with an already saved capture file. The Filtering and Statistical features are really efficient when dealing with any network analysis process. In the next chapter, we will dive into analyzing the commonly used application layer protocols.

Exercise

Q.1. What is the purpose of the Statistics menu and what tools does it contain?

Q.2. Using the Conversations dialog, can you figure out the busiest host on the network? If yes, how?

Q.3. Think of a scenario where using the Endpoints window can be useful.

Q.4. Is it possible to create a display filter using the Endpoints window?

Q.5. Switch the name resolution feature off while viewing the conversations window. What difference does it make if it is switched on?

Q.6. Can using the Summary option from an already saved capture file help you figure out the total number of ignored packets after you apply a display filter?

Q.7. Describe the benefits of using different graphing techniques while analyzing data.

Q.8. Using an IO graph, create a filter to plot the DNS traffic in a green line.

Q.9. Create an IO graph and show UDP traffic in red along with general TCP traffic. Then, change the y axis unit to per bytes.

Q.10. Create a display filter for FTP packets, and apply the same in a Flow graph. Then, customize it to check the SEQ number and ACKs instead of details.

Q.11. Using a previously captured file, create a Round Time Trip graph and figure out the packet whose RTT is the highest. Then, check the sequence number of that packet and verify its sequence number by comparing it with the graph.

Q.12. Create a Throughput graph between a server and your client. Try to figure out at what time the throughput was at its peak and also try to check the average throughput in bytes/seconds.

Q.13. If you have a requirement to view TCP packets in a raw data form, then

which option will you opt for to customize the same window in order to view just the responses from the server side?

Q.16. Point out at least 5 benefits of using the Follow TCP Stream dialog.

Q.17. Explain the significance of the Expert Info dialog and figure out how many categories are there in a Warnings section.

Q.18. Using a command-line protocol analyzer, start sniffing your currently working network interface and save all traffic to a file named `traffic.pcap` (capture traffic at least for a minute).

Q.19. Capture only DNS traffic using tshark and save all the capture packets to a file named `DNS.pcap`.

Q.20. Create a display filter to filter HTTP and SSL traffic from the traffic.pcap file we created earlier and save the filtered traffic to a new file called `HTTP.txt`.

Q.21. Using the statistical features available in tshark, figure out the total number of hosts in the `traffic.pcap` file and save all the IP addresses that belong to one single host of your choice (Google, Yahoo, Apple, and so on) to a file named `hosts.txt`.

Q.22. Using the statistical feature available in tshark, check the Ethernet address of the hosts participating in the communication process from the `traffic.pcap` file and figure out the most communicating host from the list.

Q.23. View the protocol distribution using tshark statistical functions for the `traffic.pcap` file.

Chapter 4. Inspecting Application Layer Protocols

This chapter will lead you through the common application layer protocols and will make it easy for you to find any anomalies. You will understand and analyze the normal behavior of application layer protocols by looking at the most common protocols and understand their usual and unusual behaviors.

- DNS—normal and unusual
- Lab Up—DNS
- FTP—normal and unusual
- Lab Up
- HTTP—normal and unusual
- Lab Up—HTTP
- SMTP—normal and unusual
- Lab Up—SMTP
- SIP—normal and unusual
- Lab Up—SIP
- VoIP—normal and unusual
- Lab Up—VoIP
- Decrypting encrypted traffic
- Practice questions

We will cover some of the most common application layer protocols that govern today's networks, whether small or big. Without spending too much time, let me take you on this wonderful journey of protocols.

Domain name system

Imagine a world of Internet where you have to type a random numerical value (IP address), instead of a name, to visit a website. Also, assume that each numerical figure is different. Considering this, how many IP addresses can you memorize? 5? 10? Perhaps, 50 at max? So, now, you are confined to visiting just 50 websites. This doesn't really sound feasible.

Suppose instead of just memorizing the IP addresses, you note down each of them, followed by the name that you want to give to the website to figure out which website is for what purpose. Now, you can create an Excel file for yourself, consisting of the IP addresses written next to the name of the website you gave. This way, probably, you can collect more than a thousand website addresses for later use.

For the sake of your unlimited web experience, DNS comes to your rescue, and it does exactly what you did in the preceding example. DNS creates a database of websites with their IP addresses, along with the name of the domain, A single row of record is often termed as resource records in a zone file. Each entry in the zone file is termed as a resource record. DNS uses TCP and UDP, both for different purposes, over the port `53` by default.

As a client, when you try to visit a website from your LAN environment, your request is being sent through an internal DNS server that looks up the resource records it contains. The request is termed as a DNS query. If your DNS server has already saved the IP address for the domain you are looking for, your client machine will get a reply from the internal DNS server that contains the IP address of the website you are trying to visit. Thus, you can form IP packets and start communicating. This reply is termed as a DNS response.

Dissecting a DNS packet

A DNS packet consists of a couple of unique fields that are briefly discussed here:

- **Transaction ID**: This is a number that keeps the dots connected between a particular domain query and it's corresponding response.
- **Query/response**: Every DNS packet is marked as a query or a response, depending on the details it contains.
- **Flag bits**: Each query and response contains different flag bits set, which are as follows.
    - **Response**: The message is a query or a response.
    - **Opcode**: This determines the type of query contained. Opcode ranges between 0–15. Refer to the following table:

Opcode	Description
0	Standard query
1	Inverse query
2	Server status request
3	Unassigned
4	Notify
5	Update
6-15	Unassigned

- **Truncated**: This determines whether the packet is truncated if its size is large (greater than 512 bytes).
- **Recursion desired**: The query sent by your client is supposed to go on a recursive search procedure from one DNS server to another if the resource record you are looking for is not present.
- **Recursion available**: If this bit is set, then it means the recursion that your

client requested is available, and if what you are looking for is not present on one server, then your query would be transferred to another DNS for lookup procedure.

- **Reserved (z)**: .As defined by RFC 1035; Reserved for future use, must be set to zero for all queries and responses.
- **Response code**: The values in this field signifies the response.

- **Response code**: This field is used to signify whether errors and the type of error. Here are the possible code values that you can receive:

Code	Description
0	No error
1	Format error
2	Server failure
3	Name error
4	Not implemented
5	Refused

- **Questions**: Indicates the number of queries present in the packet.
- **Answers**: Indicates the number of answers in response to the query sent.
- **Authority RRs**: Indicates the number of authority resource records sent as response.
- **Additional RRs**: Indicates the number of additional resource records sent as response.
- **Query section**: The query sent to the DNS Server, it should be the same in the response received as well.
- **Answer section**: The answer that came as a response to our query. The response can be multiple too. The answer basically consists of the resource records that came in response to our query.
- **Type**: This field indicates the type of query sent. Refer to the following table for common query types.

Type	Description
A	Host address
NS	Name server
MX	Mail exchange
SOA	Start of zone authority
PTR	Pointer record
AAAA	IPv6 address
AXFR	Full zone transfer
IXFR	Incremental zone transfer

- **Additional info**: This field includes additional info containing resource records. It is not required to answer the query.

Dissecting DNS query/response

A client sends a query to the DNS server that possesses the name resolution information. Using this information, the client can start IP-based communication. Sometimes, the information the client is looking for is not available with the DNS server it requested. In this case, the DNS server itself transfers the query to any neighbor DNS it knows about, if recursion is desirable. The whole query and response thing is completed within two packets only. Refer to the following *Figure 4.1* where I am trying to visit https://www.google.co.in. A request from my client located at `192.168.1.103` is sent to the default gateway at `192.168.1.1`. This gateway will forward my query to the DNS server it knows about:

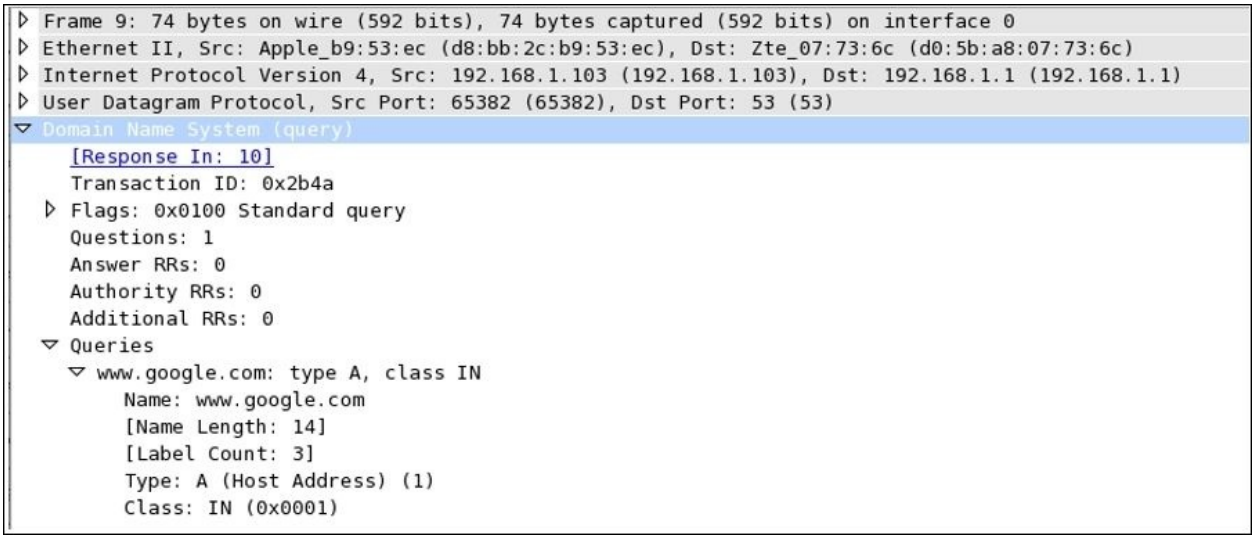


Figure 4.1: DNS query

If you notice, here, DNS is using UDP as an underlying protocol. If you want to know more about the DNS query being generated, just expand the flags section. This section will list various details such as whether recursion is available, whether recursion is desired, whether the query is truncated, what the response code is, what the Opcode for the query is, and so on. Please refer to the following screenshot.

```
▽ Flags: 0x0100 Standard query
    0... .... .... .... = Response: Message is a query
    .000 0... .... .... = Opcode: Standard query (0)
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...1 .... .... = Recursion desired: Do query recursively
    .... .... .0.. .... = Z: reserved (0)
    .... .... ...0 .... = Non-authenticated data: Unacceptable
```

The expanded `Flags` section depicts that the type of DNS packet is a query, the packet data is not truncated, and recursion is desirable if available.

In response to this query, you will be seeing one more packet with the same transaction ID that denotes the association of a particular query. It is the response packet. Response for our query will usually consist of IPv4 address for the domain we are trying to look for. We'll be returned with a single IP, or maybe multiple IPs available to it. If the domain we are looking for is not available, then its probable CNAME's will be returned in as favor.

Refer to *Figure 4.2* to understand this:

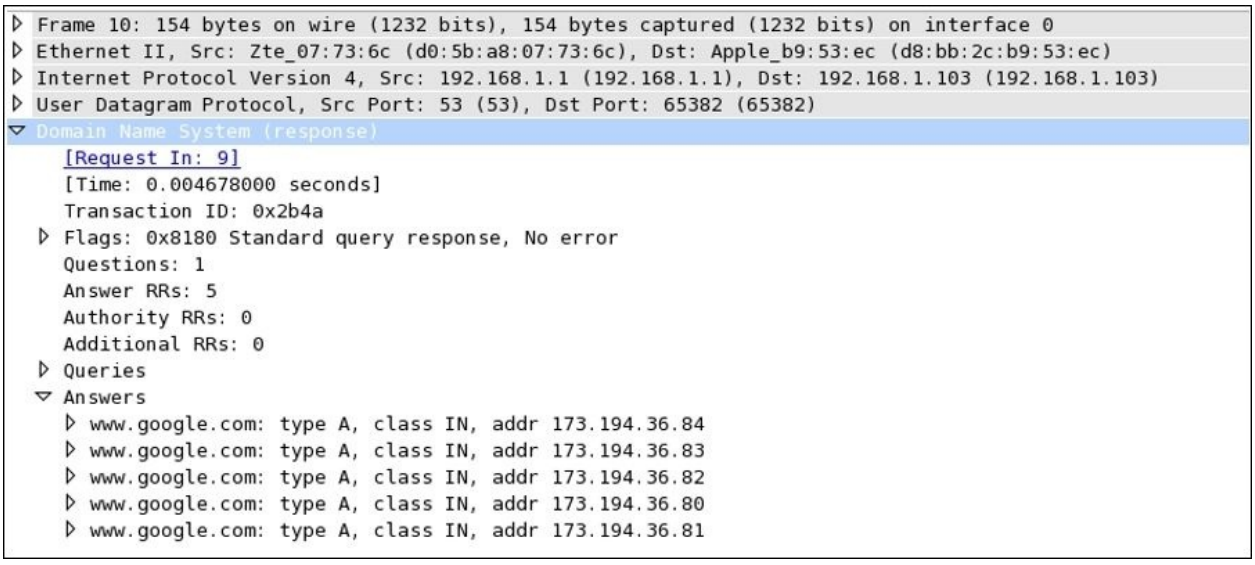
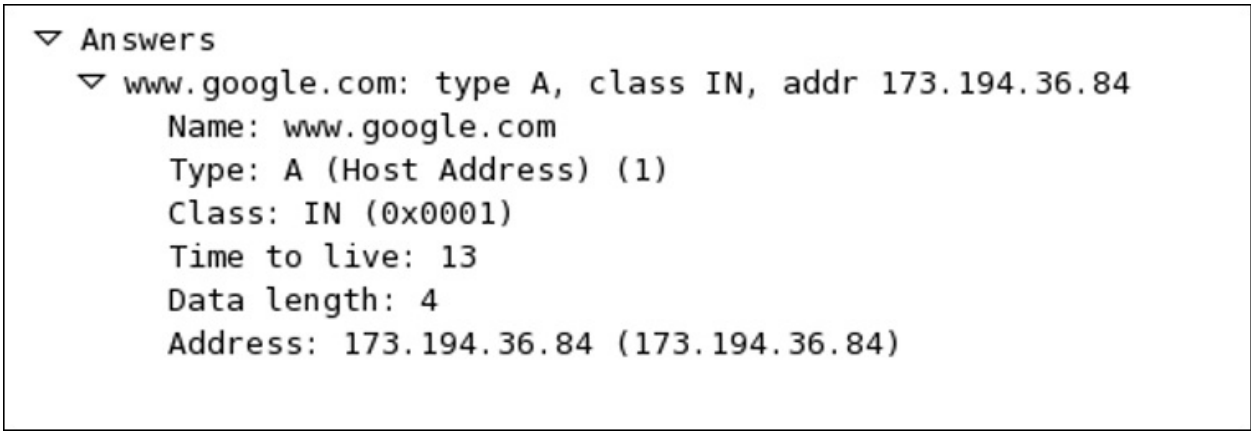


Figure 4.2: DNS response

As I said, we could get multiple replies. If you notice the **Answer RRs** section, we have received 5 replies for the www.google.com domain. For verification that the response received belongs to the previous query only, just match the `Transaction ID`. Expand any section in the answers category to view more details. Refer to the following image:



Unusual DNS traffic

Name resolution problems can have a significant impact on the performance of a network. One of the most common DNS problems you can face is when looking for something that does not exist in the DNS server's database. Sometimes, you are trying to visit a website that exists, but your DNS server is not able to resolve the domain you gave. It could also be a timed-out situation where your client waited more than the expected time for a DNS response.

In the following *Figure 4.3*, I am trying to check the type A record for the http://google.com domain, which is actually an incorrect syntax. Hopefully, it won't be resolved:

```
Anonymous:~ NotFound$ host -t a http://google.com
Host http://google.com not found: 3(NXDOMAIN)
```

Figure 4.3: Type A record for http://google.com

As expected, we got a `Not Found` error. I only tried once, but the client tried it twice to resolve the domain given. What got captured is depicted in *Figure 4.4* here:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.103	192.168.1.1	DNS	77	Standard query 0xcdc1 A http://google.com
2	0.009283000	192.168.1.1	192.168.1.103	DNS	77	Standard query response 0xcdc1 No such name
3	0.053794000	192.168.1.103	192.168.1.1	DNS	77	Standard query 0xbb93 A http://google.com
4	0.056583000	192.168.1.1	192.168.1.103	DNS	77	Standard query response 0xbb93 No such name

Figure 4.4: DNS Response-No Such Name

There can be multiple situations where you can get stuck. The best option is to first have a benchmark set for your own network, and then try comparing your problem with the benchmark you created. For example, check the name you are trying to resolve, launch a protocol analyzer, and dig into the name resolution queries and responses. Understand how long it is taking to complete the query,

the response process, and so on. Every device on the network maintains a local DNS cache (host file), which is initially used to resolve any domain you request. If the local DNS cache does not have the entry for that domain, then the request will be forwarded to the local network's DNS server, which will perform the lookup. If found, their response will be sent. Otherwise, the request from the local DNS server will be forwarded to an external DNS server, which the local DNS server is configured to look for.

File transfer protocol

Since the Internet came into existence, we have been working with FTP. It was in the limelight even when the Internet was still a closed network used by the government and other corporate organizations.

FTP uses the TCP protocol to initiate and transfer files over a designated channel. There will be two channels created; one is the command channel, and the other one is specifically a data channel. The command channel will be used to send and receive the commands and their responses. The data channel is used to send data between the client and the server.

Commonly, port `21` is used by the FTP server to listen for the connection, and any random port on the client to send and receive data. As per the standard, port `21` will be used for the command channel and port `20` for the data channel. However, you will observe random port numbers used to transfer TCP data segments.

Dissecting FTP communications

There are two types of mode a client uses to communicate with the server: active and passive. Both of them have a different approach to send and receive data. In earlier versions, active mode was in use by default, but these days, you can see passive mode in use by default. I will discuss each of them using my own virtual network where I have a FTP server (VSFTPD) configured on the `172.16.136.129` IP and a client at `172.16.136.1`. The following sections described the flow and show how the client and server will behave in the active and passive modes.

Passive mode

- The client sends a `SYN` request to the server running at port `21`.
- The client receives `SYN/ACK` from the server over a temporary port used.
- The client sends `ACK` to the server to confirm that the channel will be used for sending commands. Refer to the following screenshot:

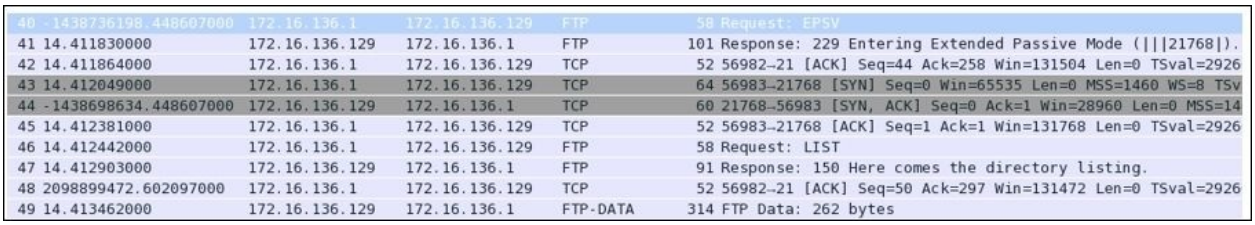
1	0.000000000	172.16.136.1	172.16.136.129	TCP	64 56982→21 [SYN] Seq=0 Win=65535
2	0.000187000	172.16.136.129	172.16.136.1	TCP	60 21→56982 [SYN, ACK] Seq=0 Ack=1
3	1846322634.413041000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=1 Ack=1 Win=

- Now, the client will be shown a welcome banner and will be asked for the assigned credentials:

4	0.018723000	172.16.136.129	172.16.136.1	FTP	88 Response: 220 Welcome to Charit's FTP se
5	555032032.287455000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=1 Ack=37 Win=131728 L
6	-952210303.718297000	172.16.136.1	172.16.136.129	FTP	62 Request: USER abc
7	-143593220.746255000	172.16.136.129	172.16.136.1	TCP	52 21→56982 [ACK] Seq=37 Ack=11 Win=29696 L
8	4.629189000	172.16.136.129	172.16.136.1	FTP	86 Response: 331 Please specify the passwor
9	4.629206000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=11 Ack=71 Win=131696
10	5.732635000	172.16.136.1	172.16.136.129	FTP	62 Request: PASS abc
11	-1086390884.249094000	172.16.136.129	172.16.136.1	FTP	75 Response: 230 Login successful.
12	2070317539.792672000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=21 Ack=94 Win=131672

Figure 4.5: Server showing welcome banner and asking for credentials

- Normally, passive mode must be on by default. Performing a directory listing will tell you that the **Extended passive** (**ESPV**) mode is in use. In this mode, the client requests the server to listen on the data port and wait for the connection. In return, the server informs the client about the TCP port number used for the connection. Please refer to the below screenshot.



40 -1438736198.448607000	172.16.136.1	172.16.136.129	FTP	58	Request: EPSV
41 14.411830000	172.16.136.129	172.16.136.1	FTP	101	Response: 229 Entering Extended Passive Mode (|||21768|).
42 14.411864000	172.16.136.1	172.16.136.129	TCP	52	56982→21 [ACK] Seq=44 Ack=258 Win=131504 Len=0 TSval=2926
43 14.412049000	172.16.136.1	172.16.136.129	TCP	64	56983→21768 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 TSv
44 -1438698634.448607000	172.16.136.129	172.16.136.1	TCP	60	21768→56983 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=14
45 14.412381000	172.16.136.1	172.16.136.129	TCP	52	56983→21768 [ACK] Seq=1 Ack=1 Win=131768 Len=0 TSval=2926
46 14.412442000	172.16.136.1	172.16.136.129	FTP	58	Request: LIST
47 14.412903000	172.16.136.129	172.16.136.1	FTP	91	Response: 150 Here comes the directory listing.
48 2098899472.602097000	172.16.136.1	172.16.136.129	TCP	52	56982→21 [ACK] Seq=50 Ack=297 Win=131472 Len=0 TSval=2926
49 14.413462000	172.16.136.129	172.16.136.1	FTP-DATA	314	FTP Data: 262 bytes

Figure 4.6: client sends ACK to the server

In frame 42, the server informs about the IP address and the port number that the client has to use while creating any data connection to the server.

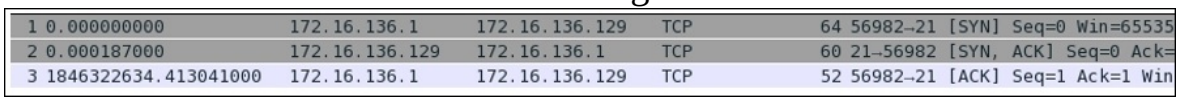
- In frame 42, the server informs us about the IP address and the port number that the client has to use while creating any data connection to the server.

Followed by a sequence of SYN, SYN/ACK, and ACK, packets which us required to create a data channel between both the devices. After this, the LIST command is executed as seen in frame 46. Then data is transferred using the temporary ports used by both the client and the server.

- As soon as the data transfer is complete, the sending host closes the connection by transmitting a FIN packet which is addressed by the receiving side using an ACK packet. The receiving side also sends a FIN packet that is acknowledged too. If both the devices want to share more data, then a new data channel will be created using random port numbers.

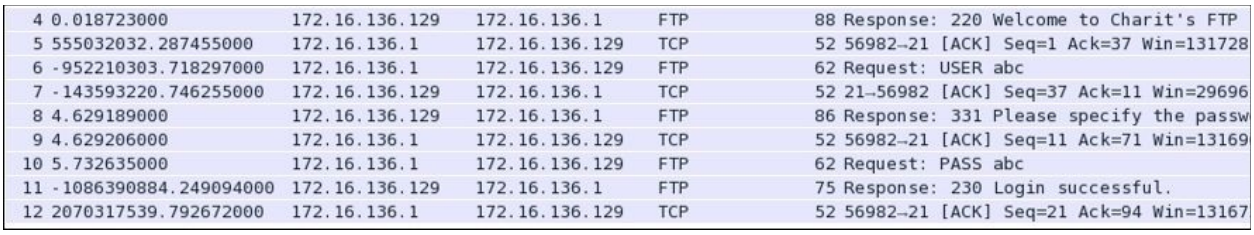
Active mode

- The client sends a SYN request to the server running at port 21.
- The client receives SYN/ACK from the server over a temporary port used by the client.
- The client sends ACK to the server to confirm that the channel will be used to send commands. Refer to the following screenshot:



1 0.000000000	172.16.136.1	172.16.136.129	TCP	64	56982→21 [SYN] Seq=0 Win=65535
2 0.000187000	172.16.136.129	172.16.136.1	TCP	60	21→56982 [SYN, ACK] Seq=0 Ack=
3 1846322634.413041000	172.16.136.1	172.16.136.129	TCP	52	56982→21 [ACK] Seq=1 Ack=1 Win

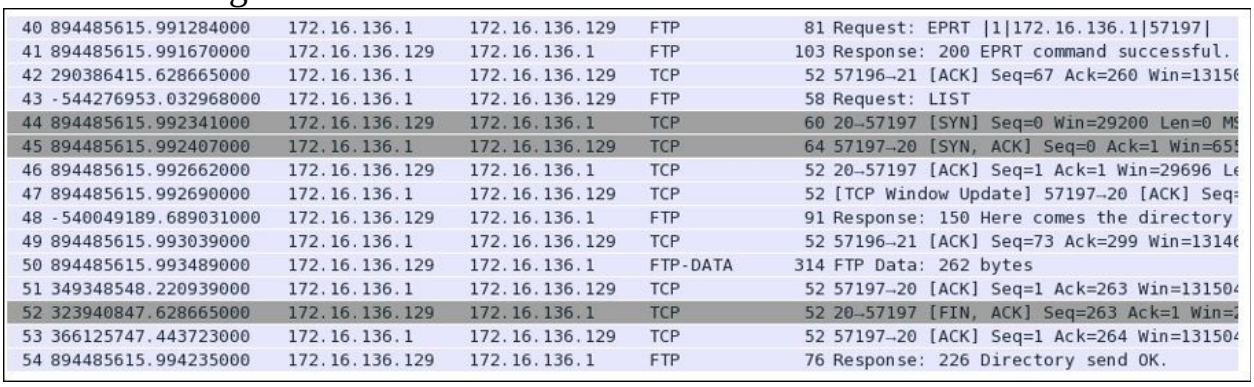
- Now, the client will be shown a welcome banner and will be asked for the assigned credentials:



4	0.018723000	172.16.136.129	172.16.136.1	FTP	88 Response: 220 Welcome to Charit's FTP
5	555032032.287455000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=1 Ack=37 Win=131728
6	-952210303.718297000	172.16.136.1	172.16.136.129	FTP	62 Request: USER abc
7	-143593220.746255000	172.16.136.129	172.16.136.1	TCP	52 21→56982 [ACK] Seq=37 Ack=11 Win=29696
8	4.629189000	172.16.136.129	172.16.136.1	FTP	86 Response: 331 Please specify the passw
9	4.629206000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=11 Ack=71 Win=13169
10	5.732635000	172.16.136.1	172.16.136.129	FTP	62 Request: PASS abc
11	-1086390884.249094000	172.16.136.129	172.16.136.1	FTP	75 Response: 230 Login successful.
12	2070317539.792672000	172.16.136.1	172.16.136.129	TCP	52 56982→21 [ACK] Seq=21 Ack=94 Win=13167

Figure 4.7: Client is shown a welcome banner and asked for credentials

- Now, we have to turn passive mode off, because, as usual, it will be on by default. Once done, we can create a data channel for transferring purposes, refer to the following screenshot:



40	894485615.991284000	172.16.136.1	172.16.136.129	FTP	81 Request: EPRT |1|172.16.136.1|57197|
41	894485615.991670000	172.16.136.129	172.16.136.1	FTP	103 Response: 200 EPRT command successful.
42	290386415.628665000	172.16.136.1	172.16.136.129	TCP	52 57196→21 [ACK] Seq=67 Ack=260 Win=1315
43	-544276953.032968000	172.16.136.1	172.16.136.129	FTP	58 Request: LIST
44	894485615.992341000	172.16.136.129	172.16.136.1	TCP	60 20→57197 [SYN] Seq=0 Win=29200 Len=0 M
45	894485615.992407000	172.16.136.1	172.16.136.129	TCP	64 57197→20 [SYN, ACK] Seq=0 Ack=1 Win=65
46	894485615.992662000	172.16.136.129	172.16.136.1	TCP	52 20→57197 [ACK] Seq=1 Ack=1 Win=29696 Le
47	894485615.992690000	172.16.136.1	172.16.136.129	TCP	52 [TCP Window Update] 57197→20 [ACK] Seq=
48	-540049189.689031000	172.16.136.129	172.16.136.1	FTP	91 Response: 150 Here comes the directory
49	894485615.993039000	172.16.136.1	172.16.136.129	TCP	52 57196→21 [ACK] Seq=73 Ack=299 Win=1314
50	894485615.993489000	172.16.136.129	172.16.136.1	FTP-DATA	314 FTP Data: 262 bytes
51	349348548.220939000	172.16.136.1	172.16.136.129	TCP	52 57197→20 [ACK] Seq=1 Ack=263 Win=13150
52	323940847.628665000	172.16.136.129	172.16.136.1	TCP	52 20→57197 [FIN, ACK] Seq=263 Ack=1 Win=
53	366125747.443723000	172.16.136.1	172.16.136.129	TCP	52 57197→20 [ACK] Seq=1 Ack=264 Win=13150
54	894485615.994235000	172.16.136.129	172.16.136.1	FTP	76 Response: 226 Directory send OK.

Figure 4.8 Creating data channel for transferring purpose

Frame `40` shows that the client is requesting to switch the passive mode off using the `EPRT |1|172.16.136.1|57197|` command. **Extended Port** (**EPRT**) helps in specifying an extended address that can be used for data connection. The command accepts three arguments: network protocol, network address, and the port number.

- Now, whenever the client tries to initiate a connection, it has to be destined for the particular address specified by the EPRT command. Before, every data connection server informed the client about the temporary port to be used.

You learned about the active and passive modes of communication that the FTP servers support. You also learned how they behave. Whenever troubleshooting any FTP connection, checking the mode will be useful and saves time.

Dissecting FTP packets

In general, every request sent from the client is a specific command set to which the server responds with a numerical value followed by a text message. See the following screenshot:

```
62 Request: PASS abc
75 Response: 230 Login successful.
```

As you can see, the server requested for the password, which the client provides. It can be seen over the wire in plain text in the list pane itself. Once the server receives and verifies that the password is correct, the respective message will be shown. In our case, the password is correct, so the client receives `230` as a response code followed by a `Login Successful` message.

The command issued from the client side can have arguments or no arguments, and the data flowing across between the devices can be simply seen in the TCP header of the packet. Refer to the following *Figure 4.9*:

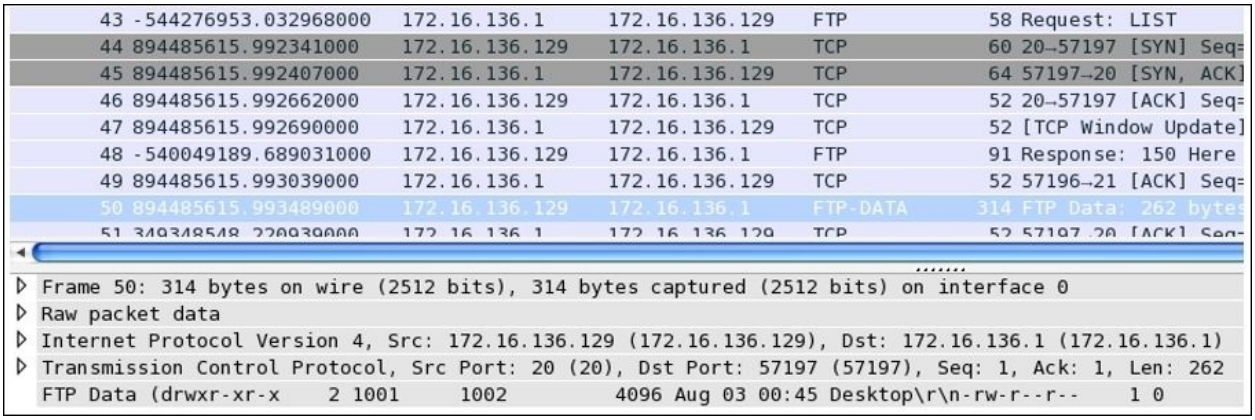


Figure 4.9: FTP-DATA returned

Frame `43` shows that the client issued the LIST command that was processed by the server, and 262 bytes of data was returned back to us. Select frame `50` to

further investigate the contents of the TCP header. One of the biggest disadvantages of using FTP is that all data travels in plain text, even the usernames and passwords.

Reassembling the FTP data stream is easy because except the data, there is nothing that travels around. There is no code or command that gets appended to the packets travelling, thus making it easy for Wireshark and the user to understand things easily. To reassemble the TCP stream of FTP packets, just right-click on the selected packet, choose the **Follow TCP Stream** option, and view it in raw form. Refer to the following *Figure 4.10*:

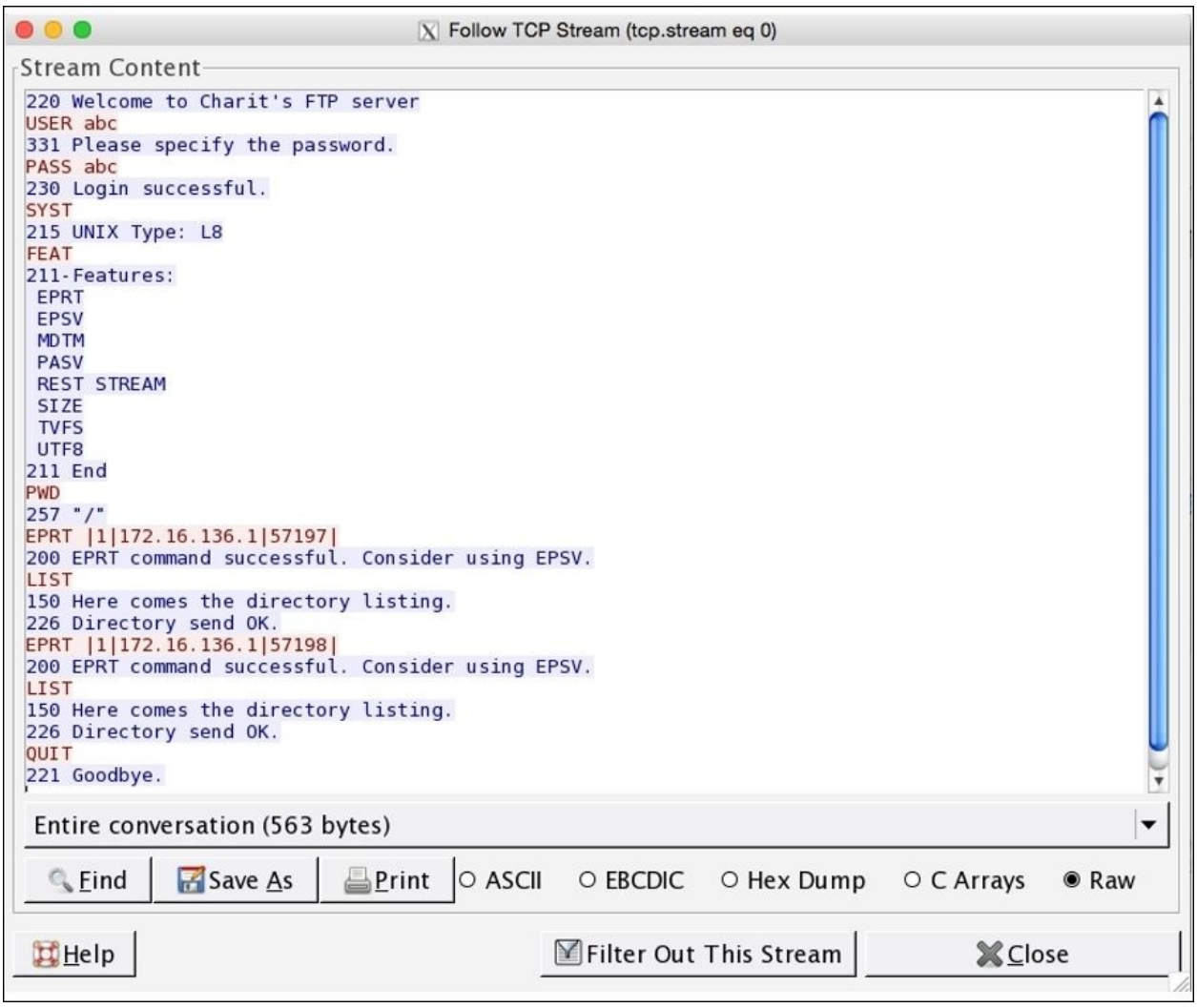


Figure 4.10: FTP stream

The entire communication between the client and the server that happened over the data and command channels is translated into human-readable format. Text in red color is what the client sent, and text in blue color is what the client received. These days, we have a couple of advanced protocols that can create an encrypted channel. One of them is **Secure File Transfer Protocol** (**SFTP**).

Unusual FTP

There can be multiple scenarios, which generate FTP traffic of an unusual type. I will use a couple of scenarios to explain this and will show you how a certain traffic type looks. An example would be brute force attacks where a malicious user tries different passwords again and again, until the exact password is matched. This is the most common traffic type that you will see while working with FTP. Applying a `ftp.request.command=="PASS"` filter will show all the password attempts that have been made to your server. If you see an unusual number of attempts in a short span of time, then it can be a brute-force attempt against your server. Refer to the following screenshot:

Figure 4.11: FTP brute force

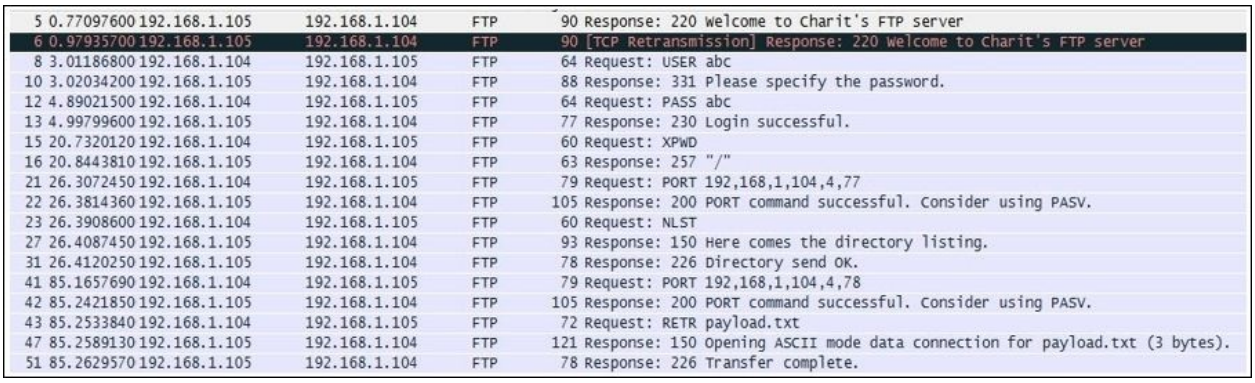
I applied the same display filter mentioned earlier, and you can see the results. Someone was trying to brute force my FTP server. To secure your server from such brute force or dictionary attacks, you can limit the server to maximum login attempts, after which the server should lock down the respective account for a particular amount of time.

You could also colorize the brute force traffic if you want. This will eventually give you a better overview of your capture file or live traffic. Try it out using the code that the server sends back to the clients in response.

Another example is a malicious device that is infected by some malware. Due to the malware, the device is trying to contact a command and control-center server

to download some payload, perhaps for privilege escalation purpose or to launch further attacks. There is even a possibility where an attacker sitting on the other side is trying to download or upload something. Let me take an example to explain. I have a Kali Linux box running at `192.168.1.105` and a Windows box at `192.168.1.104`. Through Kali, I created a small malware that was downloaded and installed by the victim (Windows). Once executed, we will get the shell from the device. Then, we can launch FTP from within the shell to connect our Kali box for privilege escalation purposes.

Refer to the following screenshot that captures the FTP traffic between the attacker and the victim:



5	0.77097600	192.168.1.105	192.168.1.104	FTP	90 Response: 220 Welcome to Charit's FTP server
6	0.97935700	192.168.1.105	192.168.1.104	FTP	90 [TCP Retransmission] Response: 220 Welcome to Charit's FTP server
8	3.01186800	192.168.1.104	192.168.1.105	FTP	64 Request: USER abc
10	3.02034200	192.168.1.105	192.168.1.104	FTP	88 Response: 331 Please specify the password.
12	4.89021500	192.168.1.104	192.168.1.105	FTP	64 Request: PASS abc
13	4.99799600	192.168.1.105	192.168.1.104	FTP	77 Response: 230 Login successful.
15	20.7320120	192.168.1.104	192.168.1.105	FTP	60 Request: XPWD
16	20.8443810	192.168.1.105	192.168.1.104	FTP	63 Response: 257 "/"
21	26.3072450	192.168.1.104	192.168.1.105	FTP	79 Request: PORT 192,168,1,104,4,77
22	26.3814360	192.168.1.105	192.168.1.104	FTP	105 Response: 200 PORT command successful. Consider using PASV.
23	26.3908600	192.168.1.104	192.168.1.105	FTP	60 Request: NLST
27	26.4087450	192.168.1.105	192.168.1.104	FTP	93 Response: 150 Here comes the directory listing.
31	26.4120250	192.168.1.105	192.168.1.104	FTP	78 Response: 226 Directory send OK.
41	85.1657690	192.168.1.104	192.168.1.105	FTP	79 Request: PORT 192,168,1,104,4,78
42	85.2421850	192.168.1.105	192.168.1.104	FTP	105 Response: 200 PORT command successful. Consider using PASV.
43	85.2533840	192.168.1.104	192.168.1.105	FTP	72 Request: RETR payload.txt
47	85.2589130	192.168.1.105	192.168.1.104	FTP	121 Response: 150 Opening ASCII mode data connection for payload.txt (3 bytes).
51	85.2629570	192.168.1.105	192.168.1.104	FTP	78 Response: 226 Transfer complete.

Figure 4.12: victim FTP capture

As you can clearly see, the attacker connected to the FTP server and downloaded the `payload.txt` file, which might be used to gain root privileges over the box.

If something of this nature is able to bypass your firewalls and other security appliances in place, then consider improvising the configuration you created and try to avoid these things in future. Sometimes, activity of this kind can be legitimate as well, but it should not stop you from investigating further. A small file of a few kbs is enough to compromise your whole network.

Hyper Text Transfer Protocol

Data on the web is transferred using the HTTP application layer protocol. Normal communication in HTTP is a request/response model where the communication between a client and a server is coordinated by a set of rules. The client requests for a certain resource to the server and then receives a status code that specifies the current status of the requested resource. If available then, the resource is also sent along with the status code. HTTP is one of the most popular and most widely used protocols to transfer data requested by browsers from the respective servers. The world of Internet is mostly governed by HTTP that runs on the transport layer.

How it works – request/response

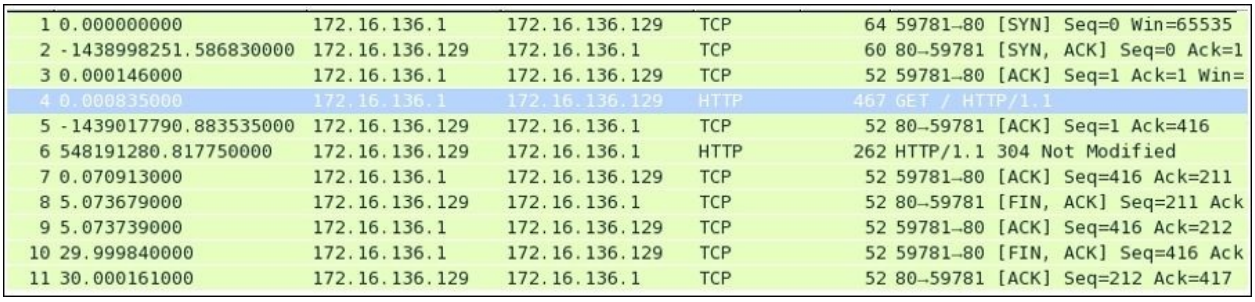
Every time you visit a website, this smart protocol takes care of your web-browsing experience. Web server utilizes the HTTP protocol to serve web pages they contain to the requesting clients. At the beginning of every HTTP session, the TCP three-way handshake takes place. It creates a dedicated channel between the communicating hosts followed by HTTP and data packets, which are sent in and received while the session is active. For instance, you are visiting a web server located at `http://172.16.136.129` and the client at `172.16.136.1`. Using our client-server infrasrtucture, we will try to capture the requests sent and responses received.

I will try to visit the home page located at the server mentioned earlier and will capture the traffic generated for the whole session, that is, requests sent and responses received. Follow the actions mentioned here to replicate the scenario.

Request

- Open your browser, and type the **Uniform Resource Locator** (**URL**) of any website that you want to visit. In my case, the website is located at `http://172.16.136.129` (Don't get confused because of the IP address I am using to visit a webserver. While studying DNS remove, we discussed that it is just a way to locate a webserver that is assigned with an IP address.). Press *Enter* to go to the home page. Here is the screenshot of the home page I am visiting:

- Due to the our preceding actions, a couple of packets are generated that are captured by Wireshark. Let's have a look at the list pane shown in the following screenshot:



1	0.000000000	172.16.136.1	172.16.136.129	TCP	64	59781→80 [SYN] Seq=0 Win=65535
2	-1438998251.586830000	172.16.136.129	172.16.136.1	TCP	60	80→59781 [SYN, ACK] Seq=0 Ack=1
3	0.000146000	172.16.136.1	172.16.136.129	TCP	52	59781→80 [ACK] Seq=1 Ack=1 Win=
4	0.000835000	172.16.136.1	172.16.136.129	HTTP	467	GET / HTTP/1.1
5	-1439017790.883535000	172.16.136.129	172.16.136.1	TCP	52	80→59781 [ACK] Seq=1 Ack=416
6	548191280.817750000	172.16.136.129	172.16.136.1	HTTP	262	HTTP/1.1 304 Not Modified
7	0.070913000	172.16.136.1	172.16.136.129	TCP	52	59781→80 [ACK] Seq=416 Ack=211
8	5.073679000	172.16.136.129	172.16.136.1	TCP	52	80→59781 [FIN, ACK] Seq=211 Ack
9	5.073739000	172.16.136.1	172.16.136.129	TCP	52	59781→80 [ACK] Seq=416 Ack=212
10	29.999840000	172.16.136.1	172.16.136.129	TCP	52	59781→80 [FIN, ACK] Seq=416 Ack
11	30.000161000	172.16.136.129	172.16.136.1	TCP	52	80→59781 [ACK] Seq=212 Ack=417

Figure 4.13: Packets captured by Wireshark

All these packets get generated as soon as you press *Enter*. As you can see, the first three packets are TCP three-way handshake packets where our client is requesting the server to create a dedicated channel. In our case, the connection was successful. However, if the server daemon wasn't running or because of any reason the server is not accepting our requests, then we could have seen `RST ACK` packets, like the one shown here:

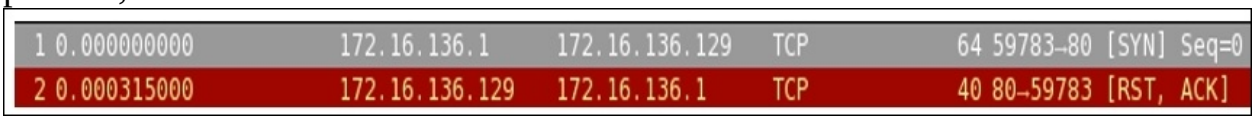


Figure 4.14:RST and ACK packets, as server not accepting the requests

This error states that the server is out of service or perhaps the server is not supposed to respond to our requests.

- After the TCP packets, you can see the first HTTP request sent by our client. Every request comprises a couple of elements that are sent to the server:

```
GET / HTTP/1.1\r\n
Host: 172.16.136.129\r\n
If-None-Match: "12625d-bc-51c6ab45063d1"\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
If-Modified-Since: Mon, 03 Aug 2015 16:31:40 GMT\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/600.6.3
Accept-Language: en-us\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
```

Figure 4.15: HTTP request

- This is how a request looks. In the first line, there are three things passed on to the server as the arguments, which are HTTP method and requested resource location "/" (root directory)
- The second line specifies the `Host` argument that is required by the `HTTP/1.1` protocol requests. The value of this field is the webserver's address that you typed in the address bar of the browser.
- The fourth line is the `ACCEPT` parameter that mentions what kind of content is acceptable by the requesting client in response.

- The `If-modified-since` parameter is sent from the client to the server, which includes the date and time of your previous request made to the server. If the server contents have been changed since your previous request, then you will receive the new updated page. Otherwise, your system will present you with the locally cached page that will eventually save some resources.
- The next field is `User-Agent`, which specifies the browser-related information that you are using to visit the webpage. This information will be used by the server to present you with browser-compatible content.
- Parameters such as `Accept-Language` and `Accept-Encoding` are passed on to the server to inform us of what type of content is acceptable to the client. So, while the server prepares the response material, these things should be taken into consideration.
- The `Connection-Alive` parameter specifies that the client wishes to keep the connection working after this particular request has been processed.

All the HTTP packets are sent most commonly to the webserver at port `80` (other common webserver ports are `8080`, `3132`, `8088` and so on. which are being dissected by Wireshark as per HTTP protocol preferences).

Response

- As you can see, after the fourth packet, the server acknowledges the client's request to get to the server's web root directory. The server starts transmitting the resource that client requested for. The sixth packet in the list pane is what the client received, a status code followed by a short message, including the content of the resource requested. Refer to the following *Figure 4.16* illustrating the HTTP response:

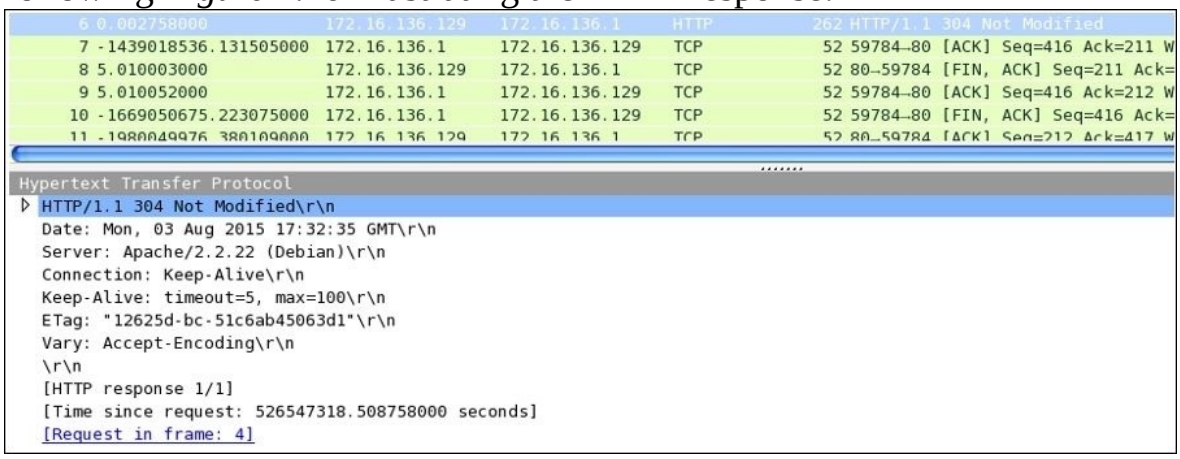


Figure 4.16: HTTP response

- As a part of TCP communication, the client will acknowledge every packet sent by the server. It can be seen in the seventh packet that the client is trying to send an `ACK` for the resource it received.
- Let's dissect the response elements for packet number six. The first line consists of three arguments sent in response. They denote the HTTP protocol version in use, the status code (`304` in our case, which specifies that the requested resource did not change since the time mentioned in the `Date` parameter), and finally, a brief description about the status code (`Not Modified` in our case).
- In the third line, the `Server` parameter mentions the name and version of the web server running. We can see that `Apache/2.2.22` is the server that is located at `172.16.136.129`.
- The fourth and fifth lines state that the server wishes to keep the connection alive. The duration for which the server wishes to do so is also mentioned in the next line of the parameters sent in response to us. Rest of the content is

mentioned in the next few lines are some configuration parameters.

This is a very basic example to check out the request and responses exchanged between the client and the server. However, this basic thing is what actually happens every time you visit a website. As stated earlier, we receive a status code followed by a brief description in response. With every tab you open in your browser, there will be a new socket created between a client and a server connected through an IP address and the port number on which the web server runs.

Unusual HTTP traffic

All the details mentioned earlier are part of a normal traffic pattern. What we are about to witness is some unusual traffic pattern that you might face while dealing with HTTP. I will try to mention some do's and don'ts, which might prove helpful to you while troubleshooting and analyzing HTTP. Most of the HTTP problems revolve around errors such as `404`, some kind of redirection, DNS resolution problems, and server-related issues. Let me explain each scenario in detail.

For instance, you are visiting a web server, and you are looking for something that is currently not available or the requested resource's location has been changed. In such cases, you will receive a `404` status code, which denotes that the requested resource is not found on the server. Refer to the following screenshot where I tried to request for a file named `abc.txt` on a web server that does not exist:

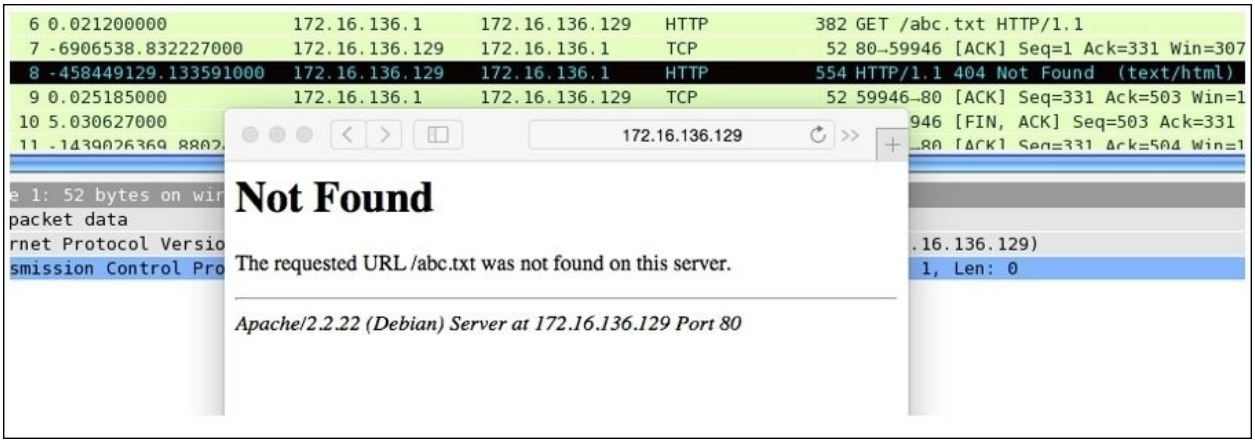
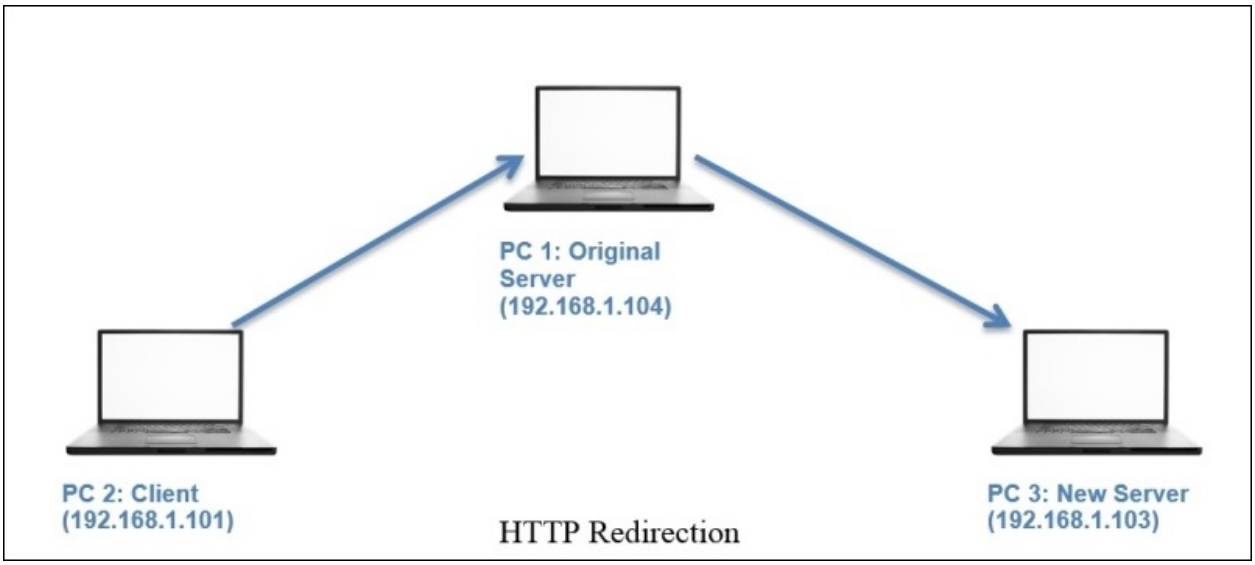


Figure 4.17 : HTTP 404

On the list pane, you can see that the requested resource is not available. So, we get `404 Not Found Error`. Such errors could be malicious too if someone is trying to perform directory listing on your webserver. Changing the coloring rules of such 404 packets to something different other than the normal HTTP packets rules will get our attention quickly. As you can see, packet number eight is a HTTP packet, applied with a different coloring scheme.

Redirection of the user's request is often done when a certain requested resource location has been changed to another address or the resource isn't available. Now, to make you understand redirection, I have made some changes in our infrastructure that can be easily seen in the diagram shown here:



Now, the request from the client sent to the original server at `192.168.1.104` will be redirected to a new server located at `192.168.1.103` without any further efforts by the client. To configure redirection, you have to modify your server's configuration file. The following captured packets depict the redirection happened. Refer to the next list pane in *Figure 4.18*:

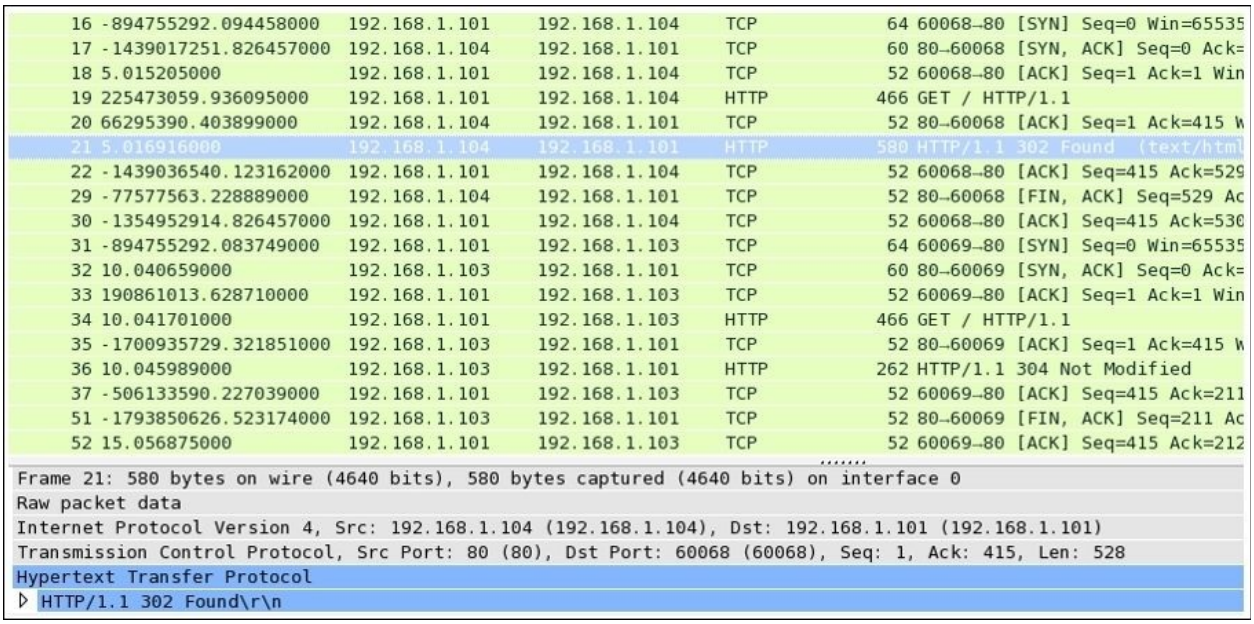
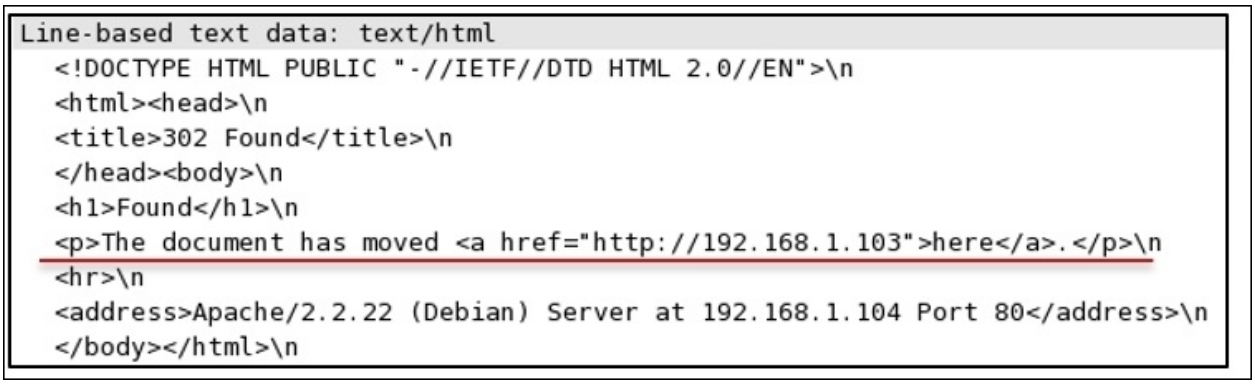


Figure 4.18: HTTP redirection

As you can see, a TCP handshake was initiated with the old server at `104` followed by an HTTP GET request. The server at `104` responded with a `302 Found` response in packet `21`, which is an indication of redirection. Our request was sent to the new server located at `103` with whom we again initiated the TCP three-way handshake (packet `31`). After packet `31`, the destination field was changed to the new server's address.

On investigating packet `21` further, we can see the content that redirected our request to the new server. Expand the `Line-based text data` section under the HTTP section of the details pane for packet `21`. Refer to the following screenshot:

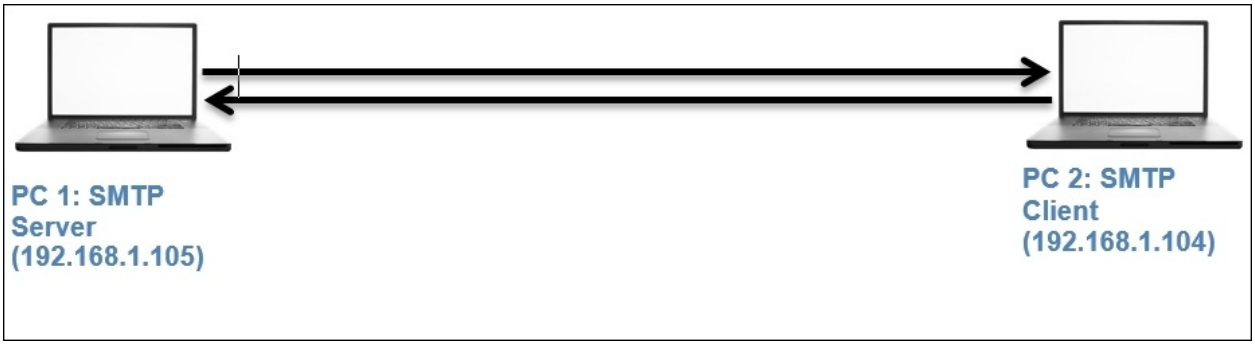


We have already discussed DNS resolution problems in the DNS protocol section. For example, if the requested web server is not able to resolve your request using your internal DNS server as well as other external servers, then you won't be able to visit the website. Even if the DNS servers are working fine and you are not able to visit the site, then congestion can be the problem, where a server is not able to process multiple requests at the same time. This will result in errors such as `408 time-out requests`, `429 Too Many requests`, or even `404 not found`. The world of HTTP is enormous, and day-to-day situations can differ from person to person. The most important fact that you should keep in mind is that if all your basic-level concepts are clear, then only it would be an easy to do the job you have been assigned. Nothing can beat common sense with out-of-the-box thinking.

Simple Mail Transfer Protocol

SMTP is used widely to send and receive emails over small, as well as large, infrastructures (can be public or private). The protocol uses the Sender-SMTP process to send e-mails and the Receiver-SMTP process to receive emails. This makes SMTP a client-server-based protocol that runs over port `25`. However, many mail server admins follow the secure practice of changing the default port number for SMTP to any other random port that prevents the server from sending any spams out there in the wild and even keep the server out-of-reach from malicious users.

Most commonly, an SMTP channel for mail transfer is created using a TCP three-way handshake that happens between two hosts, which is followed by a series of SMTP packets. For illustration purpose, I configured one SMTP server on `192.168.1.105` and a client on `192.168.1.104`. The client will request the server to send an e-mail to an address known to the client. The server will respond to this request with numerical code, followed by a brief response parameter. For understanding the real functioning of the protocol, I will be using the following architecture.

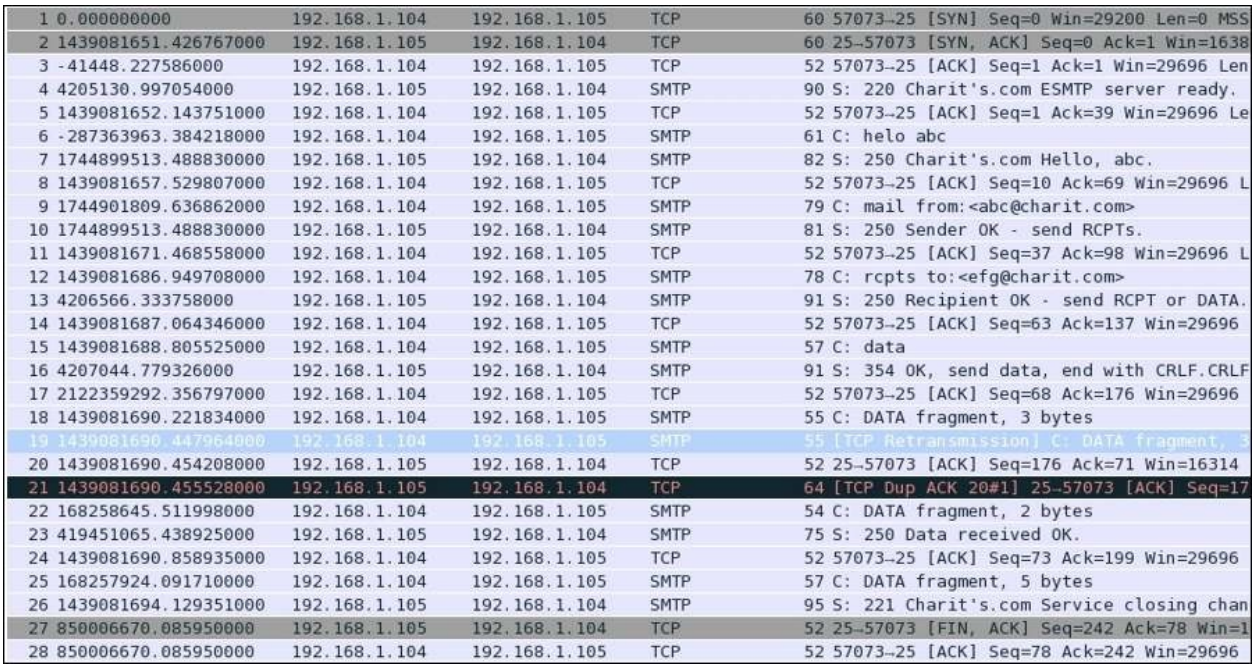


Usual versus unusual SMTP traffic

Using the netcat client from Kali Linux, I will try connecting to the SMTP mail service running on a Windows machine. Once a dedicated channel is created between the server and the client, the server indicates that it is ready to accept any commands sent in. Also, the server will respond with numerical codes with a short summary. I followed these steps to connect and send an e-mail:

1. Open a connection using netcat `nc –nv 192.168.1.105 25`.
2. Initialize an SMTP session using the `HELO testmail` command.
3. Specify the `from` address using the `MAIL FROM:<abc@charit.com>` command.
4. Specify the recipient's address using the `RCPTS TO:<efg@charit.com>` command.
5. To enter data into the mail body, type `DATA` and press *Enter*. Now, type the message you wish to send. Once you are finished writing your email, type a `.` to mark the ending and press *Enter*.
6. Now, your message will be sent. If you wish to send more emails, follow the same procedure; or else, you can close your connection with the mail server. Type `QUIT` to do so.

The series of commands I followed generated a couple of packets that contain details about the session in a very granular form. I also created a capture filter, which captured only the packets associated with the client and server that would help me in closely analyzing the packets related to the session; and preventing other packets entering the list pane. All of these commands mentioned will only work when the server is configured to permit clear text message communication without any authentication, refer to the following screenshot depiction for similar behavior.



1	0.000000000	192.168.1.104	192.168.1.105	TCP	60 57073→25 [SYN] Seq=0 Win=29200 Len=0 MSS
2	1439081651.426767000	192.168.1.105	192.168.1.104	TCP	60 25→57073 [SYN, ACK] Seq=0 Ack=1 Win=1638
3	-41448.227586000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=1 Ack=1 Win=29696 Len
4	4205130.997054000	192.168.1.105	192.168.1.104	SMTP	90 S: 220 Charit's.com ESMTP server ready.
5	1439081652.143751000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=1 Ack=39 Win=29696 Le
6	-287363963.384218000	192.168.1.104	192.168.1.105	SMTP	61 C: helo abc
7	1744899513.488830000	192.168.1.105	192.168.1.104	SMTP	82 S: 250 Charit's.com Hello, abc.
8	1439081657.529807000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=10 Ack=69 Win=29696 L
9	1744901809.636862000	192.168.1.104	192.168.1.105	SMTP	79 C: mail from:<abc@charit.com>
10	1744899513.488830000	192.168.1.105	192.168.1.104	SMTP	81 S: 250 Sender OK - send RCPTs.
11	1439081671.468558000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=37 Ack=98 Win=29696 L
12	1439081686.949708000	192.168.1.104	192.168.1.105	SMTP	78 C: rcpts to:<efg@charit.com>
13	4206566.333758000	192.168.1.105	192.168.1.104	SMTP	91 S: 250 Recipient OK - send RCPT or DATA.
14	1439081687.064346000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=63 Ack=137 Win=29696
15	1439081688.805525000	192.168.1.104	192.168.1.105	SMTP	57 C: data
16	4207044.779326000	192.168.1.105	192.168.1.104	SMTP	91 S: 354 OK, send data, end with CRLF.CRLF
17	2122359292.356797000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=68 Ack=176 Win=29696
18	1439081690.221834000	192.168.1.104	192.168.1.105	SMTP	55 C: DATA fragment, 3 bytes
19	1439081690.447964000	192.168.1.104	192.168.1.105	SMTP	55 [TCP Retransmission] C: DATA fragment, 3
20	1439081690.454208000	192.168.1.105	192.168.1.104	TCP	52 25→57073 [ACK] Seq=176 Ack=71 Win=16314
21	1439081690.455528000	192.168.1.105	192.168.1.104	TCP	64 [TCP Dup ACK 20#1] 25→57073 [ACK] Seq=17
22	168258645.511998000	192.168.1.104	192.168.1.105	SMTP	54 C: DATA fragment, 2 bytes
23	419451065.438925000	192.168.1.105	192.168.1.104	SMTP	75 S: 250 Data received OK.
24	1439081690.858935000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=73 Ack=199 Win=29696
25	168257924.091710000	192.168.1.104	192.168.1.105	SMTP	57 C: DATA fragment, 5 bytes
26	1439081694.129351000	192.168.1.105	192.168.1.104	SMTP	95 S: 221 Charit's.com Service closing chan
27	850006670.085950000	192.168.1.105	192.168.1.104	TCP	52 25→57073 [FIN, ACK] Seq=242 Ack=78 Win=1
28	850006670.085950000	192.168.1.104	192.168.1.105	TCP	52 57073→25 [ACK] Seq=78 Ack=242 Win=29696

Figure 4.19: SMTP session

Packets from `1-3` are TCP-handshake packets. The handshake is happening between the client and the server. In the fourth packet, the client receives a message stating `220` as the response code. This means the server is ready and available to respond to the client's request. In the sixth packet, the client initializes the standard SMTP session using the `HELO` command (You must be wondering why most of the packets listed in the list pane start with `C` or `S`. Requests sent from the client are marked with the character `C`, and server responses are marked with character `S`.). Then, enter the sender's and recipient's e-mail addresses, which were confirmed to be correct by the server, with response code `250` in packets `10` and `13`. After that, enter the e-mail body using the `DATA` command, which was successfully received by the server in packet `23`. In the end, the user gracefully closes the connection by issuing the `QUIT` command, which the server confirmed in packet `26`, thus sending the `FIN, ACK`.

Now, I will introduce you to the dark side of SMTP that you might have witnessed, or you will someday. By dark side, I meant the packets that are not supposed to pop up inside the list pane usually. However, if they do, then you

have to look into your protocol configuration. For this, I would like to introduce you to some quite common scenarios that you should be aware of.

The first and foremost case I can think of is when the server and the client are not able to create a dedicated channel for communication; in short, the TCP handshake did not go well. This can happen because of many reasons, such as the mail server daemon is not running, the mail server is not running on the default port, the mail server daemon has reached the maximum simultaneous client connections allowed or connections from a particular subnet are not allowed there can be multiple scenarios related to this. The following list pane depicts two kinds of traffic abnormalities:

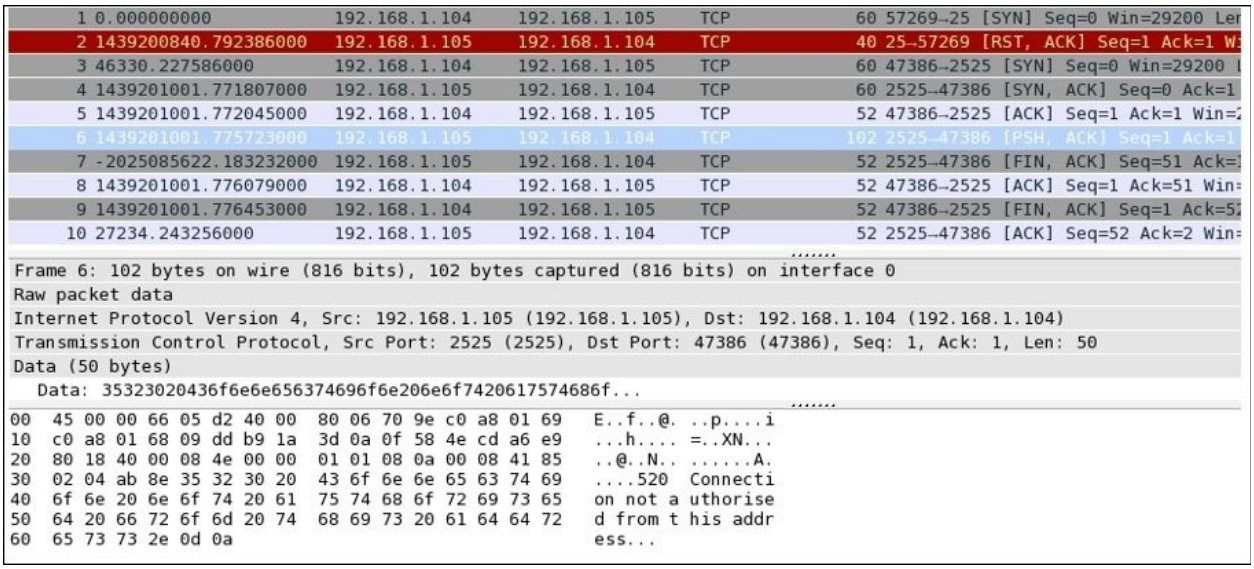


Figure 4.20: SMTP unusual traffic

The first two packets were generated due to an error, which stopped the TCP handshake from occurring. This error can be generated due to many factors, some of which are mentioned here:

- Mail server daemon is not running
- Mail server daemon default port is changed
- Mail server daemon has reached the maximum simultaneous connections limit (DDoS attack).
- Mail server's configuration has been tampered with

Let's suppose now, that the client came to know about the correct port number to which the connection should be initiated, but still, the session was not created successfully. Observe the traffic starting from packet `3` to the packet `10`, the last packet. A TCP three-way handshake happened, but then, suddenly, the client was kicked off from the session. What could be the possible reason for such a response from the server? Perhaps the client is not allowed to get connected because of some restrictions in place, such as IP or MAC filtering.

1	0.000000000	192.168.1.104	192.168.1.105	TCP	60 57230→25 [SYN] Seq=0 Win=29200 Len=0 MSS=1460
2	-1439191021.671720000	192.168.1.105	192.168.1.104	TCP	60 25→57230 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len
3	-299332529.969384000	192.168.1.104	192.168.1.105	TCP	52 57230→25 [ACK] Seq=1 Ack=1 Win=29696 Len=0 TS
4	-1435002675.066153000	192.168.1.105	192.168.1.104	SMTP	90 S: 220 Charit's.com ESMTP server ready.
5	0.144765000	192.168.1.104	192.168.1.105	TCP	52 57230→25 [ACK] Seq=1 Ack=39 Win=29696 Len=0 T
6	2.062258000	192.168.1.104	192.168.1.105	SMTP	61 C: helo abc
7	2.199304000	192.168.1.105	192.168.1.104	SMTP	82 S: 250 Charit's.com Hello, abc.
8	2.199772000	192.168.1.104	192.168.1.105	TCP	52 57230→25 [ACK] Seq=10 Ack=69 Win=29696 Len=0
9	212289295.064342000	192.168.1.104	192.168.1.105	SMTP	76 C: mail from:<efg@abc.com>
10	12.450170000	192.168.1.105	192.168.1.104	SMTP	81 S: 250 Sender OK - send RCPTs.
11	12.450646000	192.168.1.104	192.168.1.105	TCP	52 57230→25 [ACK] Seq=34 Ack=98 Win=29696 Len=0
12	22.846623000	192.168.1.104	192.168.1.105	SMTP	75 C: rcpts to:<abc@abc.com>
13	-566788423.768283000	192.168.1.105	192.168.1.104	SMTP	96 S: 553 We do not relay non-local mail, sorry.
14	23.255494000	192.168.1.104	192.168.1.105	TCP	52 57230→25 [ACK] Seq=57 Ack=142 Win=29696 Len=0
15	53.669236000	192.168.1.105	192.168.1.104	TCP	52 25→57230 [FIN, ACK] Seq=142 Ack=57 Win=16328
16	-700612319.058152000	192.168.1.104	192.168.1.105	TCP	52 57230→25 [FIN, ACK] Seq=57 Ack=143 Win=29696
17	53.671389000	192.168.1.105	192.168.1.104	TCP	52 25→57230 [ACK] Seq=143 Ack=58 Win=16328 Len=0

Figure 4.21: Client not allowed to get connected due to some restrictions

Another type of abnormal traffic that can be seen widely these days is harvesting of e-mails used by spammer and spamming botnets roaming in the wild. A spammer tries to harvest emails from the publicly accessible mail servers to verify which email address is valid and which isn't. For example, look at the following screenshot (*Figure 4.15*) where a malicious user tries to verify the existence of an e-mail ID using the **E-mail From** field, verification of e-mail addresses can alos be done using `VRFY` command. Depending on the response, the user will come to know whether the email is valid or not. Observe packet number `13` for the server's response. These kinds of attacks are done using a custom-made dictionary file, which matches the current domain requirements. Once an email is verified, the spammer can perform various forms of social-engineering attacks. A response code greater than 350 in SMTP protocol is probably some kind of error that can reduce your network performance, thus increasing the latency.

Session Initiation Protocol and Voice Over Internet Protocol

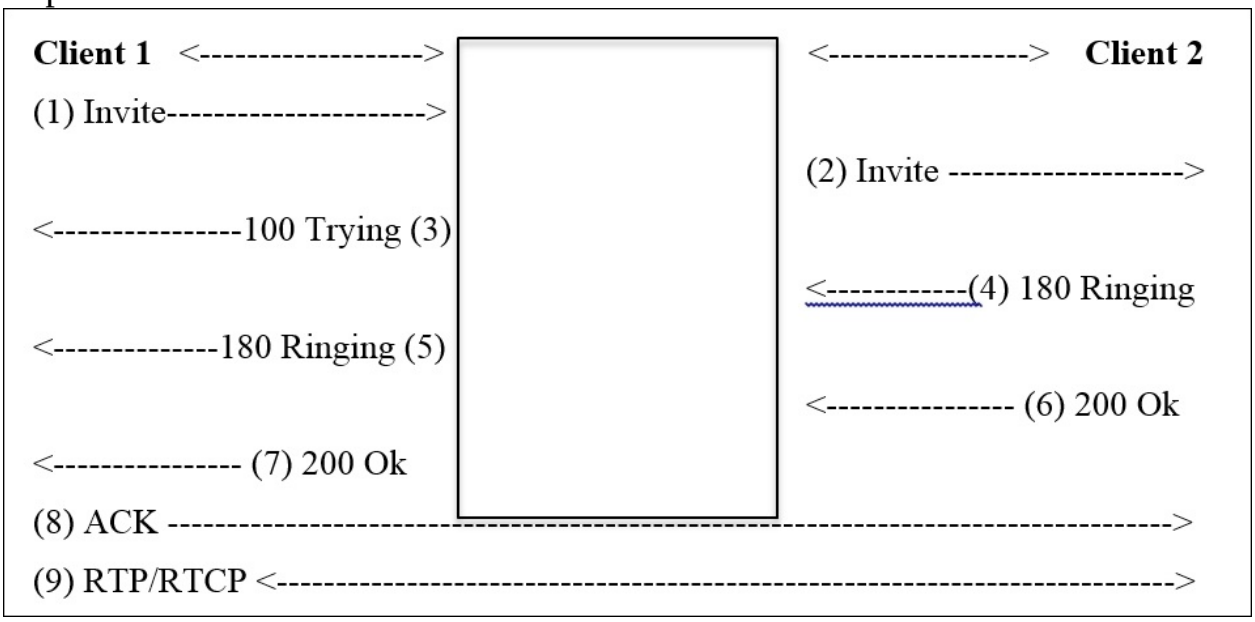
SIP is a part of the VOIP protocol family that is just a signaling protocol used to create, manage, and terminate voice over IP sessions in a networking environment. Examples of SIP can be a two-way phone call or a conference call, including multimedia sessions where multiple hosts can be present. This protocol is generally discussed in regards to the initiation of the session between the remove parties ; hosts/nodes that intend to communicate. After the initiation is completed, the data is transferred over the dedicated channel where the **Real time Transport Protocol** (**RTP**) helps. Basically, the family of RTP governs the transport and the flow control of all of the multimedia items (RTCP controls the flow).

The two most used tools while working with this protocol are the Statistics menu, under which we will cover Protocol Hierarchy, Packet Lengths, and flow graphs, which will give you an idea of data travelling back and forth between two hosts. Under the **Telephony** menu, you will see the RTP and VOIP Calls options that can facilitate us in assembling the VOIP call streams. We can then play them back to hear the conversation, this is what makes me really excited about Wireshark.

SIP runs over the UDP protocol and commonly uses port `5060`. All of this together in an IP-based environment makes it possible for us to dial instantly to our friends over a VoIP-enabled device. SIP makes it easy for the VOIP telephony server to establish user locations. It facilitates us with different call-managing features such as initiating calls, disconnecting calls, adding someone to a conference call, transferring calls, and various others. SIP is not going to help you maintain the quality of calls, yet SIP is one of the most important standards used by various services. Before we jump directly into looking and listening to the traffic, let's get ourselves acquainted with how the traffic moves in a voice over IP call.

There will be three parties we will consider: two of them are clients and one is the IP telephony server that helps in transferring the required and necessary packets back and forth between the two communicating hosts. The following

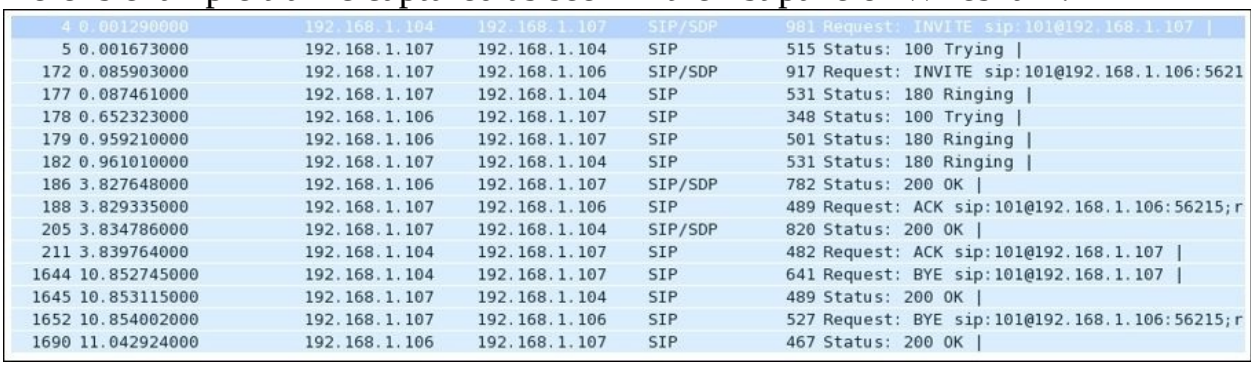
figure depicts a small infrastructure telephony architecture and lists the various steps taken:



- **Client 1** sends an **Invite** request to initiate the session using SIP.
- The telephony server in between, transfers the request to **Client 2**.
- The telephony server acknowledges **Client 1** with the **100 TRYING** packet.
- **Client 1** receives a **180 RINGING** packet as soon as **Client 2** starts ringing. When **Client** 2 on the other side received the call, it sends the **200 OK** packet, which is forwarded to **Client 1**.
- Now, the client sends the **ACK** packet to acknowledge the receipt of the **200 OK** packet.
- Now, both parties are connected with a dedicated channel over which the RTP/RTCP packet starts flowing back and forth.
- Once both of them are done, there will be a **BYE** packet sent from by the hosts communicating, which is acknowledged by the other end.
- If you observe, most of the packets are passing through the telephony server. Because the telephony server only knows about the exact location of the connected hosts.
- Once the connection is successfully created, all the packets are sent and received directly by the clients without the server's intervention.

I have configured a small VoIP telephony infrastructure using Asterisk PBX that you can download freely from the vendor's website. VOIP server is located at `192.168.1.107`, client 1 at `192.168.1.104`, and client 2 at `192.168.1.107`. Then, I downloaded X-Lite client using which, I tried calling client 2 from client 1. Now, using the real SIP traffic captured, it becomes easy for us to analyze and learn. Interestingly, there is an option using which, we can play back the communication captured (this can be really dangerous and more amazing).

Here is example traffic captured as seen in the list pane of Wireshark:



4	0.001290000	192.168.1.104	192.168.1.107	SIP/SDP	981 Request: INVITE sip:101@192.168.1.107 |
5	0.001673000	192.168.1.107	192.168.1.104	SIP	515 Status: 100 Trying |
172	0.085903000	192.168.1.107	192.168.1.106	SIP/SDP	917 Request: INVITE sip:101@192.168.1.106:5621
177	0.087461000	192.168.1.107	192.168.1.104	SIP	531 Status: 180 Ringing |
178	0.652323000	192.168.1.106	192.168.1.107	SIP	348 Status: 100 Trying |
179	0.959210000	192.168.1.106	192.168.1.107	SIP	501 Status: 180 Ringing |
182	0.961010000	192.168.1.107	192.168.1.104	SIP	531 Status: 180 Ringing |
186	3.827648000	192.168.1.106	192.168.1.107	SIP/SDP	782 Status: 200 OK |
188	3.829335000	192.168.1.107	192.168.1.106	SIP	489 Request: ACK sip:101@192.168.1.106:56215;r
205	3.834786000	192.168.1.107	192.168.1.104	SIP/SDP	820 Status: 200 OK |
211	3.839764000	192.168.1.104	192.168.1.107	SIP	482 Request: ACK sip:101@192.168.1.107 |
1644	10.852745000	192.168.1.104	192.168.1.107	SIP	641 Request: BYE sip:101@192.168.1.107 |
1645	10.853115000	192.168.1.107	192.168.1.104	SIP	489 Status: 200 OK |
1652	10.854002000	192.168.1.107	192.168.1.106	SIP	527 Request: BYE sip:101@192.168.1.106:56215;r
1690	11.042924000	192.168.1.106	192.168.1.107	SIP	467 Status: 200 OK |

Figure 4.22: SIP traffic

One thing you should consider is place the analyzer close to the telephony server so that you can easily capture every bit of packet-level information moving around. While capturing, if you cannot see any SIP packets, then you won't be able to capture VOIP packets as well. You would end up capturing UDP packets only in the list pane, which won't prove very fruitful for your analysis.

Analyzing VOIP traffic

Just for the sake of curiosity, I want to show you the protocol distribution for SIP traffic that can be seen using the **Protocol Hierarchy** dialog from the **Statistics** menu. Refer to the following *Figure 17*:

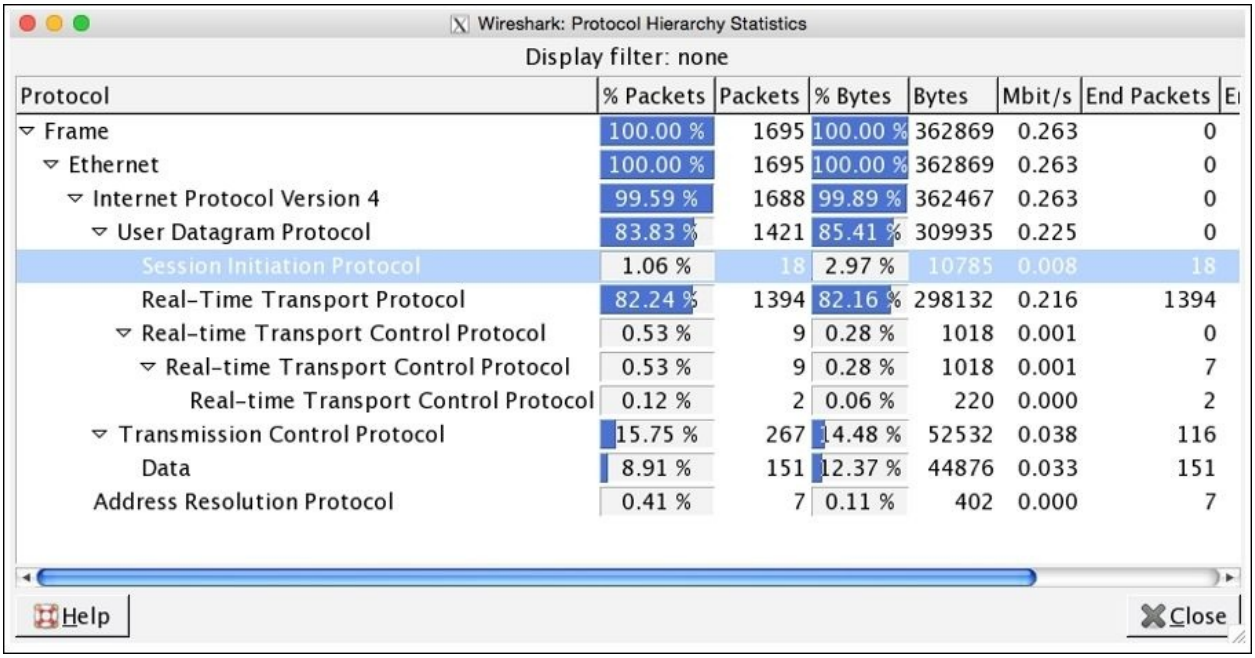


Figure 4.23: Protocol Hierarchy

Major traffic generated during the session is UDP based, and as seen in the preceding screenshot, SIP traffic is a very small part of it. If you observe closely, it is just 1 percent roughly, whereas RTP has a major role here with 82 percent. This gives an overview about the session we captured and tells us which protocol participates in what percentage. As we already know, SIP is used only to create and manage sessions that occur between two users, or it can be a multiuser conference call.

Flow graphs are one more way of getting a summary of the traffic. They help in understanding the movement of request and acknowledgements sent or received. Refer to the following *Figure 4.24*:

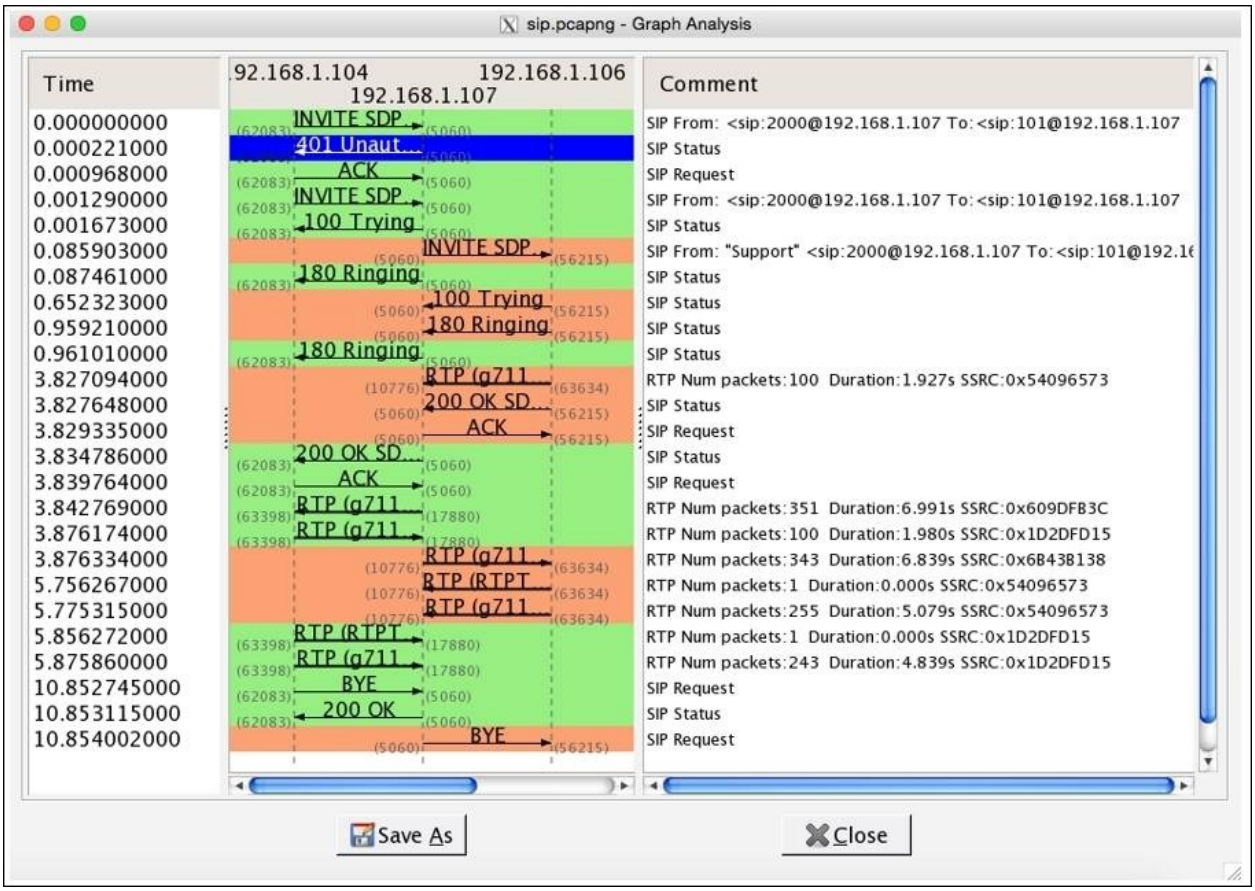


Figure 4.24: Flow graph

There are three IPs listed just below the title bar in the center section. These IPs belong to the server and the two clients that are trying to communicate. The entire request and the responses with their status codes and summary messages can be seen clearly here. Requests sent are colored in orange and the responses with green. This makes every element look more precise and easy to understand.

Reassembling packets for playback

Yes, this is possible. You can assemble the VOIP packets back to listen to either, or both sides of the communication in parallel. Let's suppose I want to listen what message client 1 sitting at `192.168.1.104` sent to the client 2. We can use the **Telephony** menu in Wireshark to reassemble the packets and choose the **VOIP Calls** option from the list. The following screenshot illustrates the

resulting dialog.

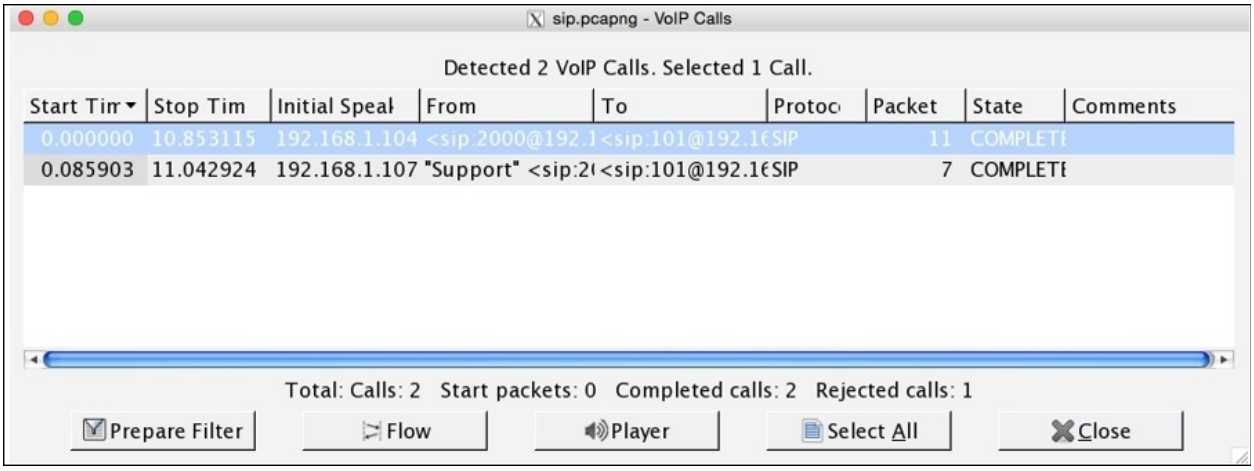


Figure 4.25 : VOIP Calls dialog

Now, choose which side of communication you want to listen to. Then, click on the **Player** button, which will then ask you to provide maximum **Jitter** (Jitter is the variance in packet rate at which the packets are being sent and received. If jitter is high, then there is a chance your network is dealing with congestion. Calls having high jitter values are not feasible to listen to.) in our communication session. The maximum jitter value is `22`. So, by default, there will be **50** ms value given in the box. You can change this value if your jitter is higher than that; otherwise, just click on **Decode**:

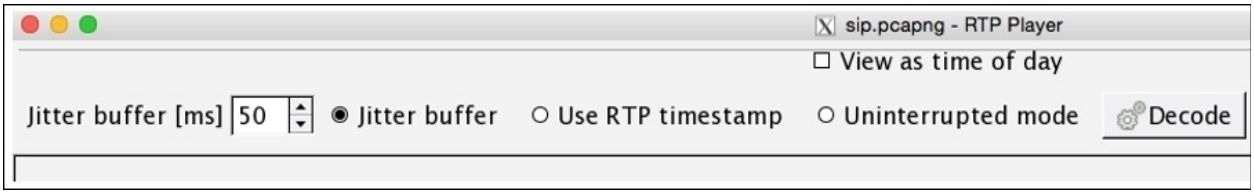


Figure 4.26: Player dialog

I did not change the default value and clicked directly on the **Decode** button, which reassembled all the VoIP packets for the side of communication I chose. Refer to the following screenshot:

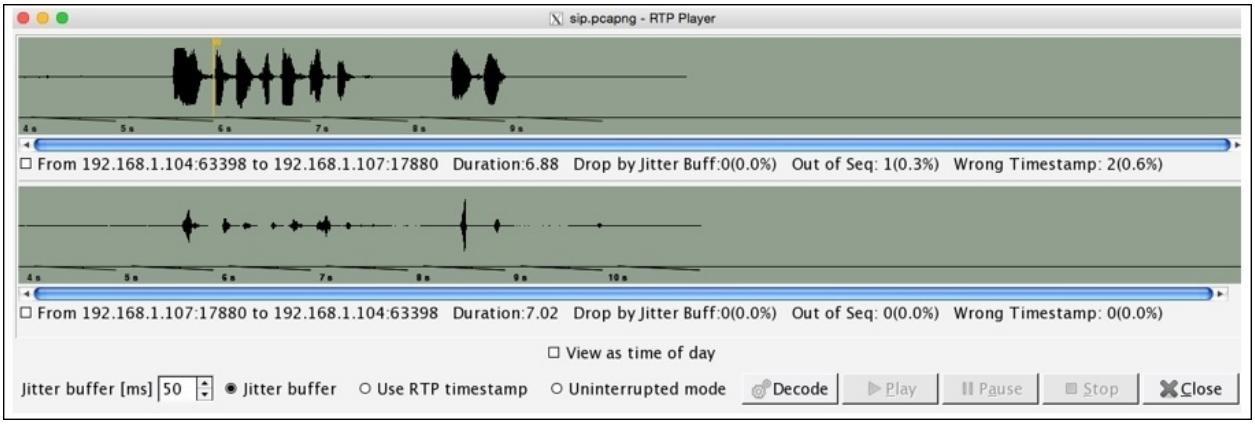


Figure 4.27: RTP Player

If you want to play the message, check the box just below the scrollbar and click on **Play**. Various useful details related to the assembled VOIP stream are listed.

Unusual traffic patterns

Wireshark has numerous tools that help a user in maintaining QS for a certain networking infrastructure and also consists of a tool that helps in identifying various day-to-day traffic anomalies. A common type of traffic when dealing with an SIP server is `INVITE` requests that are sent from one client to initiate the connection with another client. As you might already know, this process is a three-way handshake where the client who initiated the request is supposed to acknowledge when the session creation is completed. What if the client who requested does not respond with `ACK` and sends another `INVITE` request? Normally, the server will try to connect the client to the requested client machine, meanwhile waiting for the `ACK` response for the previous request. Now, let's suppose the client sent 100 `INVITE` requests through different clients on the network and did not even bother to send `ACK` for any one of those sessions created. This can result in a DOS attack (`INVITE` flood attack) where the SIP server won't be able to process any further requests (the buffer size for `INVITE` is `100`). To resolve this, you can apply a display filter to view the `INVITE` requests sent from a client or apply a filter where the status code is `200:OK`.

Other than DOS attacks, there is a chance that your network may slow down due to packet congestion, or you might not be able to get connected to another client on your network. In other words, your call cannot get through, if there is lag in setting up the call (the average call setup time is high). You will witness multiple cases once you work in a production environment. So, Wireshark and the various powerful tools it contains comes to our rescue.

For instance, if some client is trying to make a call to an invalid extension, they will get an error, and the call won't get through. Such a scenario will generate packets as shown here:

12	3.100381000	192.168.1.104	192.168.1.107	SIP/SDP	981	Request: INVITE sip:100@192.168.1.107 |
13	3.100794000	192.168.1.107	192.168.1.104	SIP	515	Status: 100 Trying |
167	3.366362000	192.168.1.107	192.168.1.104	SIP	574	Status: 503 Service Unavailable |
199	3.481824000	192.168.1.104	192.168.1.107	SIP	364	Request: ACK sip:100@192.168.1.107 |

I would suggest that you filter SIP packets consisting of error codes greater than

`399` and create a display filter using `sip.Status-Code > 399`. See the following screenshot that lists multiple errors generated while client 1 was trying to call:

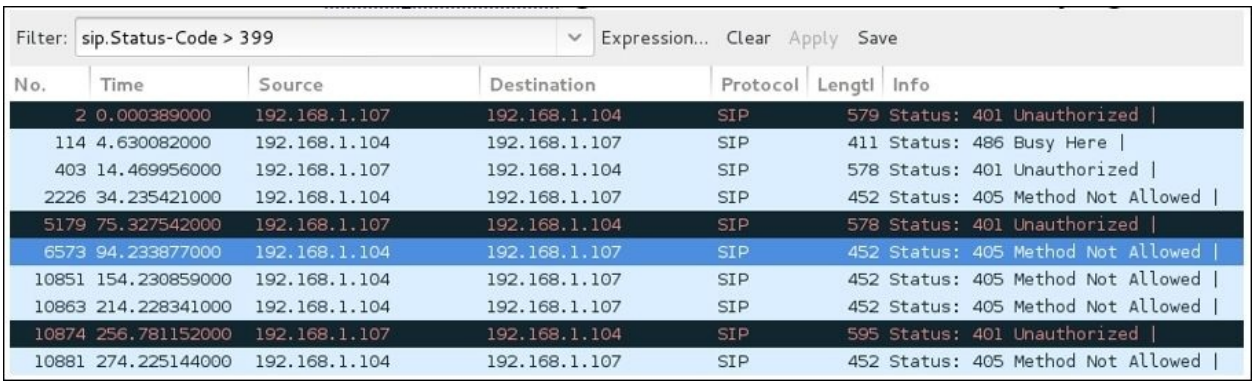


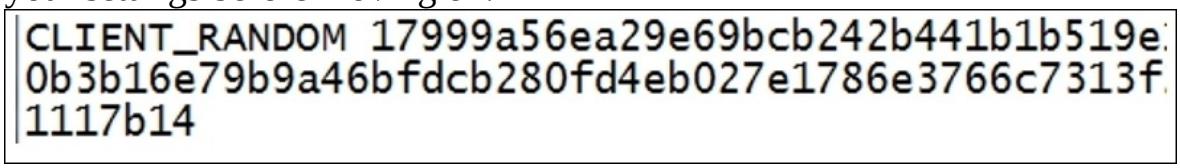
Figure 4.28: SIP error

Decrypting encrypted traffic (SSL/TLS)

Yes, it is possible to decrypt your online TLS traffic into a plain text SSL stream using Wireshark. Google Chrome and Firefox look for a log file, which stores the TLS session keys. Follow these steps to decrypt encrypted traffic:

1. Create an environment variable with the name `SSLKEYLOGFILE` that will point to a text file. Your browser will look for this file every time it starts up. To create environment variables, right-click on **My Computer** | Go to **Advanced Settings** | **Environment Variables** | **New** | **Specify Name**: `SSLKEYLOGFILE` and **Value**: `C:/Users/username/sslkeylog.txt` and click on **Ok**.
2. I have created a blank text file, `C:/Users/username/sslkeylog.txt` (make your new environment variable point to this file).
3. Now, open your browser and visit a website enabled with TLS/SSL. For demonstration purpose, I have my own SSL webserver located at `192.168.1.105` and a client located at `192.168.1.105`.

The certificate I created is self-signed; that's why you are seeing a red diagonal line across `https` in the address bar. After you visit any secure website enabled with SSL, your **sslkeylog.txt** will be populated with some random numbers, as shown in the following screenshot. If not, cross check your settings before moving on:



- I captured the whole traffic between my client and server in Wireshark. Now, go to **Edit** | **Preferences** | **Protocol tree** | **SSL** | **(Pre)-Master-Secret log**

filename | **/path/to/sslkeylog.txt** | **Ok**. Then, right-click on the SSL packet (Make sure you select **Decrypt packet data**. The option should be present in the bytes pane) and follow the SSL stream. Now, you will see something like *Figure 4.29* here:

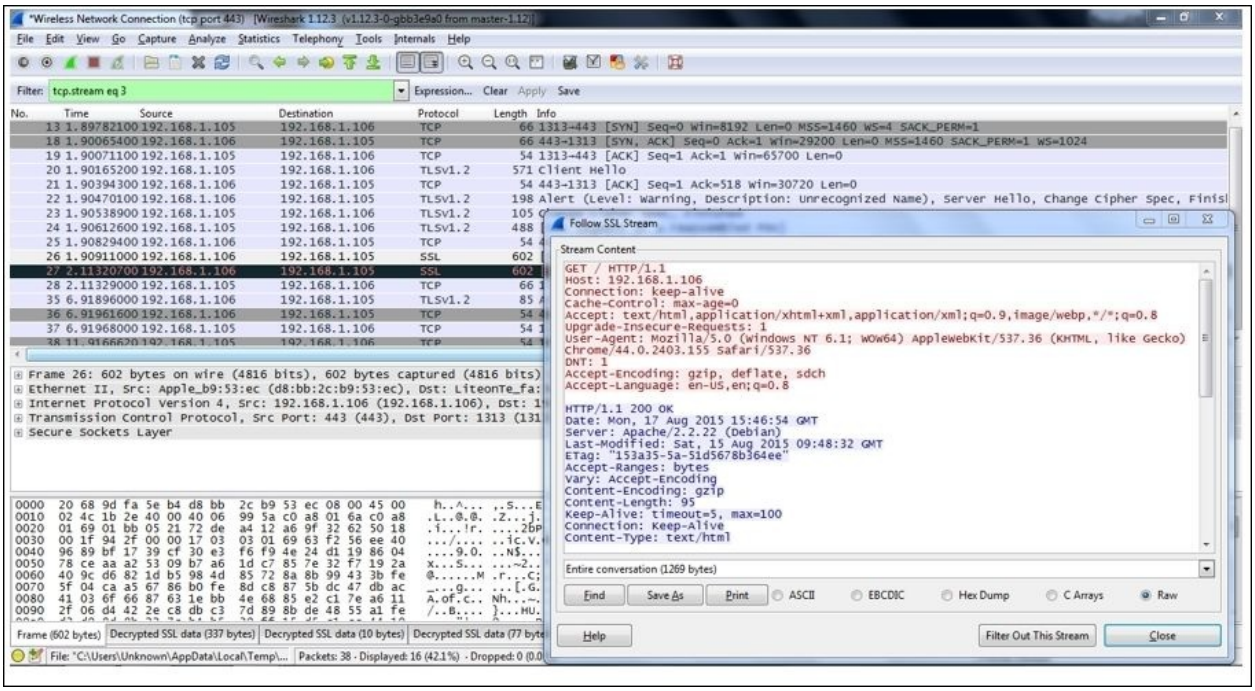


Figure 4.29: Decrypt SSL traffic

This is one of the easiest ways by which you can go ahead and decrypt SSL traffic with just a few clicks. One more way is to feed the RSA private key of the server into the Wireshark SSL preferences, which will give you the same result.

Summary

Domain name system/Service is a protocol used to resolve website names to an IP address. Using this domain name service, your machine can communicate on an IP-based network. Using zone transfer (if enabled), unauthenticated malicious users can ask for zone data form name servers, which is considered highly malicious and dangerous.

File transfer protocol has been used to transfer files from one machine to another since the Internet came into existence and is still being used in today's modern networks. The most unsecure part about FTP is that the data is passed in plain text and can be easily captured using protocol analyzers, unless you are using some encrypted form of the FTP client-server infrastructure.

The web browsers are used to present and transfer the web-based content back and forth uses hypertext transfer protocol. It is commonly also referred to as the request/response model, where a host requests for a certain resource and the server responds with a status code and the resource if available. Status codes greater than `399` should be watched closely, I would suggest is to apply different colorization schemes.

SMTP protocol is used to send e-mails. It is an unencrypted protocol where commonly authentication mechanism is not used. Every SMTP command and its corresponding arguments are passed over the wire in plain text that can be easily sniffed using Wireshark.

VoIP traffic is made up of two things: RTP for data transfer and SIP protocol used to create the session. Signaling protocol creates and manages a session where real-time transport protocol is used to carry the voice itself. Using Wireshark, anyone can capture and reassemble the packets back to listen to a communication session. One should take care of congestion, jitter, lag, and echoing problems while dealing with these protocols in order to maintain the quality of service.

Practice questions

Q.1 What is the significance of the DNS protocol while you surf the Internet?

Q.2 How would you define zone transfers and recursive DNS queries?

Q.3 What is the difference between recursion desired and recursion available in DNS queries?

Q.4 How many DNS record types exist? Explain the purpose of the AAAA record type and what does non-authoritative answer mean?

Q.5 Differentiate between active and passive modes of FTP. Explain which mode is better.

Q.6 What solution can you come up if you are being asked to make your FTP session encrypted? Explain the difference it would make.

Q.7 Using a virtual infrastructure or a physical one, install the FTP server on any of the machines and then try to communicate with it while capturing live packets in Wireshark.

Q.8 Find out how you can limit the maximum number of login attempts. How can such limitation affect the overall security of your FTP server?

Q.9 Why do we refer to HTTP communication as a request/response approach and what is the purpose of the three-way handshake while initiating the connection?

Q.10 Which version of HTTP are we currently using and what is the difference between the old and new ones?

Q.11 While your browser makes an HTTP request, various other parameters are also sent in your request. Why is it so? What is the purpose of `Accept-Encoding` and `Accept-Language` parameters sent with your request?

Q.12 Visit websites of your choice and browse a couple of pages while capturing all the packets in Wireshark. Then, create a display filter to check whether any

redirection was present in your whole session.

Q.13 For what purpose is SMTP on client side used? To send e-mails or receive them? Which protocols are popularly used to receive e-mails?

Q.14 Is it possible to perform a brute force attack on an SMTP server? If yes, then how and how do you identify such traffic pattern?

Q.15 What do you understand by e-mail harvesting and how you can perform an e-mail harvesting attack on an SMTP server? Is there any kind of specific response you will look for?

Q.16 Read about the difference between various email protocols and SMTP?

Q.17 What is the significance of SIP in a VOIP session? What percentage of traffic do you think SIP will have in a whole VOIP session?

Q.18 What is the difference between RTP and RTCP protocols?

Q.19 Download a SIP traffic capture file (`sippcap`) from Wireshark's website and analyze the session using a flow graph. Are you able to the see the process flow we discussed?

Q.20 Filter out all the wrong password attempts using specific code for such responses and apply a different coloring scheme (use the `aaa.pcap capture` file).

Chapter 5. Analyzing Transport Layer Protocols

This chapter will help you understand TCP and UDP protocols, how they communicate, the problems you can face with these protocols, and how you can use Wireshark to assist them. You will also learn how to analyze TCP and UDP protocols and look for any anomalies that may follow. The following are the topics that we will cover in this chapter:

- Understanding the TCP header and how it communicates
- Understanding the TCP analysis flags
- Lab up—TCP
- How to check for different analysis flags in Wireshark
- Understanding UDP traffic
- Lab up—UDP
- Practice questions

We will discuss TCP and UDP protocols using various practical examples that can give you an insight about how low-layer protocol packets communicate and travel in your network in order to transmit data successfully. We will also look at some common anomalies that you might witness in your day-to-day operations.

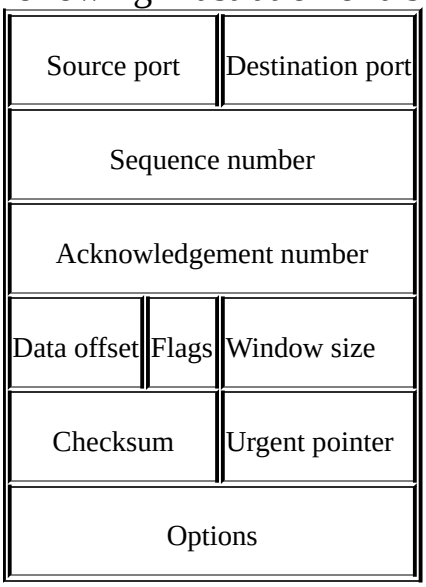
The transmission control protocol

A TCP is a connection-oriented protocol used by various other application-layer protocols to ensure data delivery without any loss of packets during transition. On the basis of sequence numbers and acknowledgement numbers, a TCP ensures fail-proof delivery of packets between the hosts that intend to communicate. A TCP is supposed to provide an end-to-end, reliable form of communication, which should be robust at all times. It sits in between the network layer and the application layer and uses the IP datagram to transfer data packets between the sender and receiver. Because of this approach, the TCP and IP are used by various application layer protocols for their reliable delivery.

A TCP is like a two-way communication process where not only the sender is involved in the communication, but even the receiver actively works to make it a successful connection. You can imagine it to be like a landline connection, where you dial a number; if the number you dialed is correct, you will hear a ringtone (if the other side is open to communicate). Only when the receiver responds by picking up the receiver, you can start talking. Likewise, in TCP-based communication, a process called **three-way handshake** takes place between the parties that are involved in the communication to create an independent channel between the two hosts.

Understanding the TCP header and its various flags

The TCP header is normally 20 bytes long, but at times, due to the presence of the options field, the TCP header size can vary up to 60 bytes. Refer to the following illustration of a simplified TCP header:



Now, let's get acquainted with the header fields to get a stronger grasp over the basics of a TCP:

- **Source port**: This is the port number associated with the sender side of the communication or you can say the port responsible for listening on the sender side.
- **Destination port**: This is the port number associated with the recipient side of the communication or you can say the port responsible for receiving the transmitted packets.
- **Sequence number**: These are the unique values that are used to ensure reliable delivery of data. TCP tracks each segment using sequence numbers.
- **Acknowledgement number**: These values are sent in response from the receiver side as part of the confirmation process that the packet was successfully received.
- **Data offset:** This indicates where the data packet begins and the length of the TCP header. The size can vary due to the presence of the options field.
- **Flags**: There are various types of flag bits present; each of them has its own

significance. They initiate connection, carry data, and tear down connections, and on the basis of their assigned purpose, we've named them as follows:

- **SYN (synchronize)**: These are the packets that are used to initiate a connection that is commonly known as the handshake process.
- **ACK (acknowledgement)**: These packets are used to confirm that the data packets have been received, and this also confirms the initiation and tear down of the connections.
- **RST (reset)**: These packets signify that the connection you were trying to create has been shut down or may be the application we were trying to communicate with is not accepting connections.
- **FIN (finish)**: These packets indicate that the connection is being torn down after the successful delivery of data packets. Both the sender and receiver send the FIN packets to gracefully terminate the connection. If they want to communicate again, they will start from the beginning, that is, from the three-way handshake process.
- **PSH (push)**: These packets indicate that the incoming data should be passed on directly to the application instead of getting buffered. To state this simply, the other host should receive data without waiting for it.
- **URG (urgent)**: Marked packets indicate that the data that the packet is carrying should be processed immediately by the TCP stack and the urgent pointer field should be examined if it is set.
- **CWR (congestion window reduced)**: These packets are used by the sender to inform the receiver that due to the transmit, the buffer is getting overfilled, and because of congestion, both the parties should slow down the transmission process to avoid any packet loss that might happen.

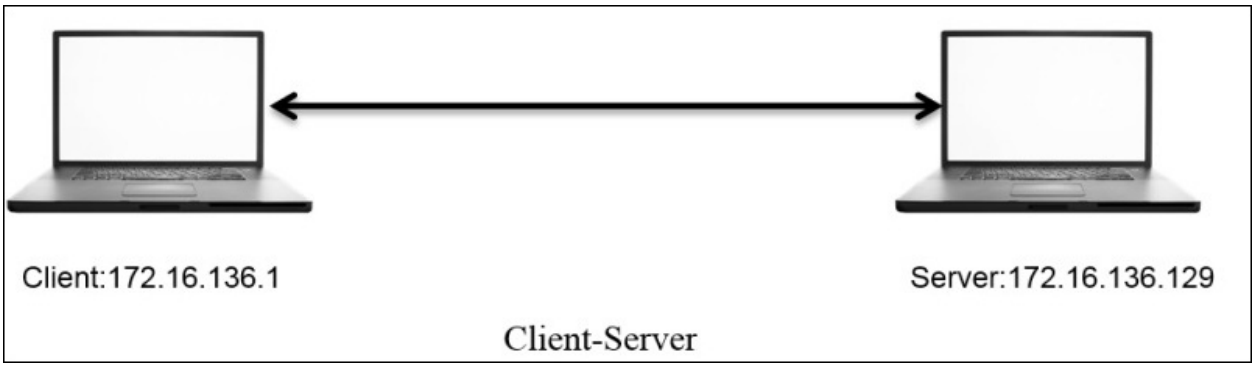
- **Window size**: This field in the header indicates the amount of data that the sender can send, . The amount is decided during the handshake process where both the hosts that communicate match the buffer size compatible for transmission. Flow control can be achieved through this field.
- **Checksum**: To cross check the integrity of the data that is being received, this field is used, where the contents of the TCP segments are validated.
- **Urgent pointer**: This field tells us about the value that the urgent pointer contains. It specifically indicates the sequence number of the octet that lies before the data.

- **Options**: This field length can vary due to the presence of various options. This field has three parts: the first part specifies the length of the option field, the second part denotes the options being used, and the third actually contains the options in use. One of the important options **maximum segment size** (**MSS**) is also part of this field.
- **Data**: The last part in the TCP header is the real data that travels around.

The preceding information gives us an overview regarding TCP headers and the significance of various parts of the header. While analyzing TCP sessions, it becomes quite important to know about these details.

How TCP communicates

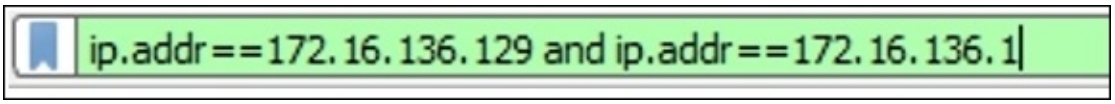
To understand and analyze the packets in real time, I have configured a server that runs at `172.16.136.129` and a client that runs at `172.16.136.1`, as shown in the following figure. Using Wireshark, I will try to illustrate the three-way handshake process, which happens before the actual data transfer as well as the tear down process (graceful termination). The three-way handshake ensures that the server and client are open to making connections and are ready with resources to create a dedicated channel between each other for a reliable delivery of packets.



Client-Server

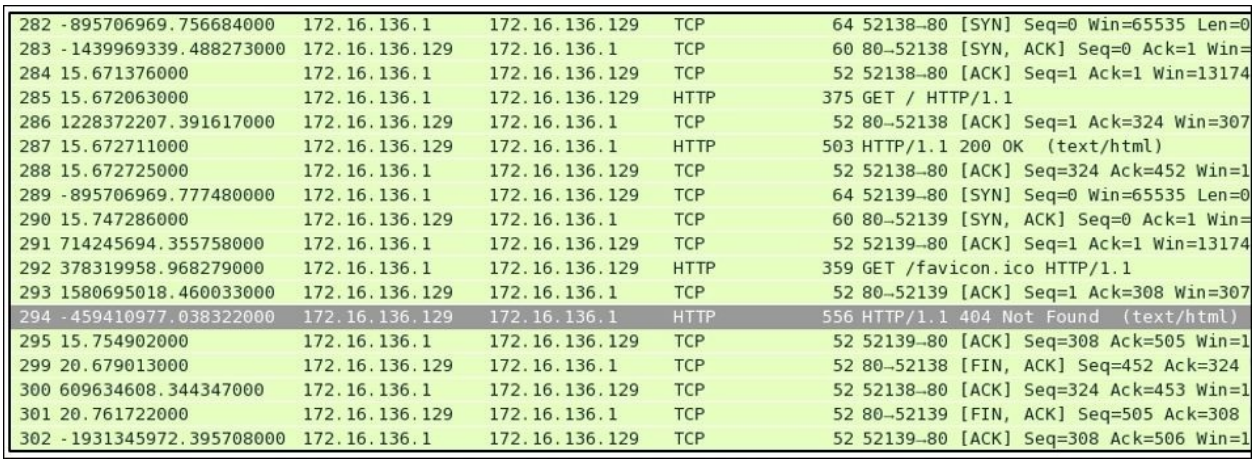
How it works

The server runs an HTTP server daemon at port `80`. On the client, I will visit the default webpage hosted at `http://172.16.136.1` while capturing all the packets taking part in the communication process.



Note

For the sake of visibility and ease, I've created a display filter to display the traffic between these two hosts specifically.



282	-895706969.756684000	172.16.136.1	172.16.136.129	TCP	64 52138→80 [SYN] Seq=0 Win=65535 Len=0
283	-1439969339.488273000	172.16.136.129	172.16.136.1	TCP	60 80→52138 [SYN, ACK] Seq=0 Ack=1 Win=
284	15.671376000	172.16.136.1	172.16.136.129	TCP	52 52138→80 [ACK] Seq=1 Ack=1 Win=13174
285	15.672063000	172.16.136.1	172.16.136.129	HTTP	375 GET / HTTP/1.1
286	1228372207.391617000	172.16.136.129	172.16.136.1	TCP	52 80→52138 [ACK] Seq=1 Ack=324 Win=307
287	15.672711000	172.16.136.129	172.16.136.1	HTTP	503 HTTP/1.1 200 OK (text/html)
288	15.672725000	172.16.136.1	172.16.136.129	TCP	52 52138→80 [ACK] Seq=324 Ack=452 Win=1
289	-895706969.777480000	172.16.136.1	172.16.136.129	TCP	64 52139→80 [SYN] Seq=0 Win=65535 Len=0
290	15.747286000	172.16.136.129	172.16.136.1	TCP	60 80→52139 [SYN, ACK] Seq=0 Ack=1 Win=
291	714245694.355758000	172.16.136.1	172.16.136.129	TCP	52 52139→80 [ACK] Seq=1 Ack=1 Win=13174
292	378319958.968279000	172.16.136.1	172.16.136.129	HTTP	359 GET /favicon.ico HTTP/1.1
293	1580695018.460033000	172.16.136.129	172.16.136.1	TCP	52 80→52139 [ACK] Seq=1 Ack=308 Win=307
294	-459410977.038322000	172.16.136.129	172.16.136.1	HTTP	556 HTTP/1.1 404 Not Found (text/html)
295	15.754902000	172.16.136.1	172.16.136.129	TCP	52 52139→80 [ACK] Seq=308 Ack=505 Win=1
299	20.679013000	172.16.136.129	172.16.136.1	TCP	52 80→52138 [FIN, ACK] Seq=452 Ack=324
300	609634608.344347000	172.16.136.1	172.16.136.129	TCP	52 52138→80 [ACK] Seq=324 Ack=453 Win=1
301	20.761722000	172.16.136.129	172.16.136.1	TCP	52 80→52139 [FIN, ACK] Seq=505 Ack=308
302	-1931345972.395708000	172.16.136.1	172.16.136.129	TCP	52 52139→80 [ACK] Seq=308 Ack=506 Win=1

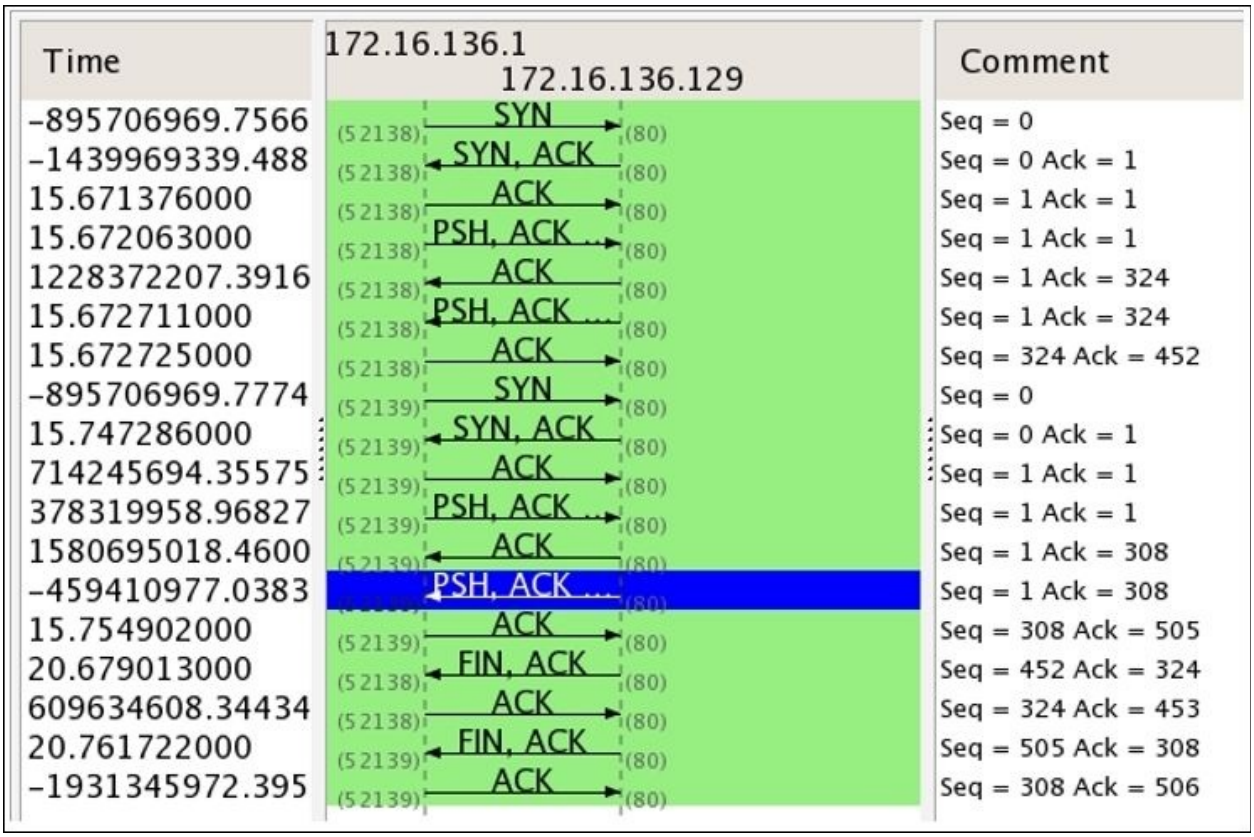
Figure 5.1: Connection Process:Three-way handshake, data transfer and tear down process

In the packets `282`, `283`, and `284`, it is clearly visible that the client and server are trying to create a dedicated channel. The client initiated the creation by sending a `SYN` packet in the `282` packet with the SEQ set to `0`. Since the server was open for communication, the server responded with a `SYN/ACK` packet with `ACK` set to `1` and `SEQ` set to `0`. This is followed by a confirmation sent from the client side that is seen in the packet number `284` with `SEQ=1` and `ACK=1`. This is what a three-way handshake process looks like. This can be seen before any real data transfer that happens that follows the TCP approach.

After the successful completion of channel creation, the client sends a `GET` request to access the contents of the web-root directory. The server acknowledged this in the packet number `287` and sent the requested content to the client's machine with the `200 OK` status message, which is acknowledged by the client in the next packet. As seen in the list pane again, the client was requesting a new resource, which the server wasn't able to find and thus sent a **404 Not Found** status message, which was acknowledged by the client in the the packet `295`.

After all the data transfer takes place, when the client has nothing left to request, or when the server has nothing left to send, the client sends `FIN/ACK` packets to properly terminate the connection. The server acknowledges this and sends its

own `FIN/ACK` packets, which are acknowledged by the client as well in the packet number `302`. This way of termination is often referred to as the teardown process. Take a look at the following screenshot that illustrates this process:



This was a small and sweet conversation that we captured and through which you learned about the process flow. I think I've one more interesting way to illustrate the process flow using graphs that we've already seen in the previous chapters. Refer to the preceding screenshot.

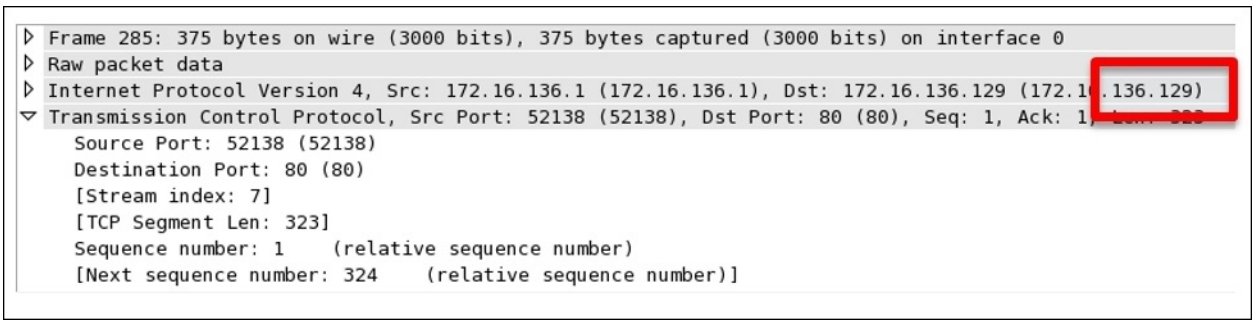
From this flow graph, it becomes more clear and concise to view the requests and responses shared between the two communicating hosts. The most interesting part that I like in the preceding screenshot is the comment section that lists out the `SEQ` and `ACK` numbers, which are sent and received by the hosts.

You must be wondering how these are generated and incremented. Let me tell you the trick behind this amazing world of numbers that is used while transferring data. The host that initiates a new connection uses **Initial Sequence**

Numbers (**ISN)** that are generated by the host's operating system. It can be any random number that has no significance with respect to the data. The sequence number we see in the packet one is zero is actually a relative referencing technique used by Wireshark to ease the numbering system for the sake of users. First of all, you should know that the numbers are used to keep track of how much data is being transferred between the two hosts.

Starting from the packet 1, where `SEQ=0` (the relative sequence number in real is `704809601`), which is received by the server and in return replies with its own `SEQ=0` and `ACK=1` for the client's `SEQ=0`. At the end of this three-way handshake, the client replies with `SEQ=1` and `ACK=1` without any further increments as no data is being transferred during the process.

Then, by the fourth packet, the client sends a `GET` request with `SEQ=1` and `ACK=1` where the data payload length equals `323` (refer to the following figure), which the server receives and acknowledges with `SEQ=1` and `ACK=324`. Did you see what just happened? The server replied by adding a total data payload length into `ACK` to denote that the data was successfully received. Hence, it sends the requested resource to the client with data payload length equals `451`, which in return gets acknowledged by the client with `ACK=452` and `SEQ=324`. In the same way, the transmission goes on until the tear down takes place using `FIN`/`ACK` packets at the end.

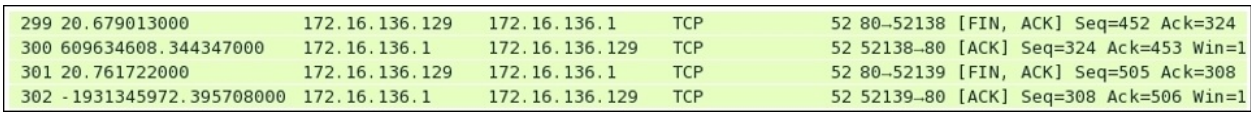


Graceful termination

We saw, in detail, the process of TCP three-way handshake using the captured packets and the flow graph that gave us insight about the process. Similarly, we should be comfortable about the teardown process, which indicates proper

termination of a session between two hosts.

Considering the same scenario that we discussed here, let me show you the packets that were generated to terminate the connection in a proper standardized format. Refer to the following screenshot for this:



After the successful delivery of all the required packets, the server initiated the teardown process (as there was nothing left to send or the client was just sitting idle and doing nothing). In the beginning, the server sent its own `FIN` and `ACK` packets to the client with `SEQ=452` (the client acknowledged the same with `ACK`) and `ACK=324` (this is the client `SEQ` number when the data transfer was completed). These were acknowledged by the client in the next packet. Following the same approach, the client issued its own `FIN` and `ACK` packets (using `SEQ` and `ACK` numbers used in the second round of communication, where the client requested something that wasn't available. Refer to the preceding flow graph to know more) to end the connection from its own side (as the connection was bi-directional), which was received and acknowledged by the server. As soon as the client received `ACK` from the server, the connection between the two hosts was closed completely, and the sockets and other resources involved during the communication were freed up.

RST (reset) packets

Often times, there will be situations when the server daemon is not available, it is not able to process your request due to overload, you are restricted to interact with the server, or the port you are trying to connect to is not open for connections (not associated with any service). There can be a lot of reasons why you will see a `RST` packet. Let me replicate the scenario and capture the traffic between the client and server I have, which will surely make it easy for you to understand this. An `RST` packet basically denotes that the connection you were trying to initiate got closed abruptly.

In this scenario, the server daemon is not running and the client is trying to

communicate; as a result, it receives `RST` packets in return for every `SYN` request sent. I tried visiting the web server just once, but you will notice more than one `SYN` and `RST` packets because every browser performs a different number of attempts over a non-responding or a closed socket at a particular interval of time. Hence, in our case, I am using the Apple Safari browser, which made at least three attempts to connect back in a max time of 3-4 minutes. I tried requesting Google Chrome as well, which made approximately 7 attempts to connect back in merely 10 minutes (the browser will continue to make a request at a particular interval of time). Refer to the following screenshot that illustrates the packets captured in the process:

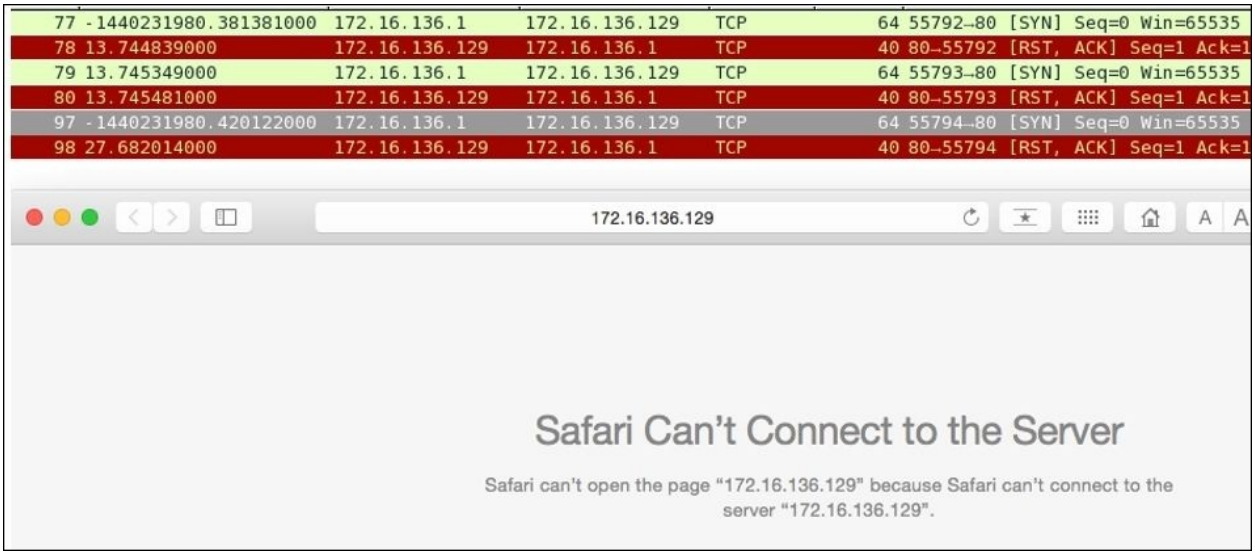


Figure 5.2: RST packets captured

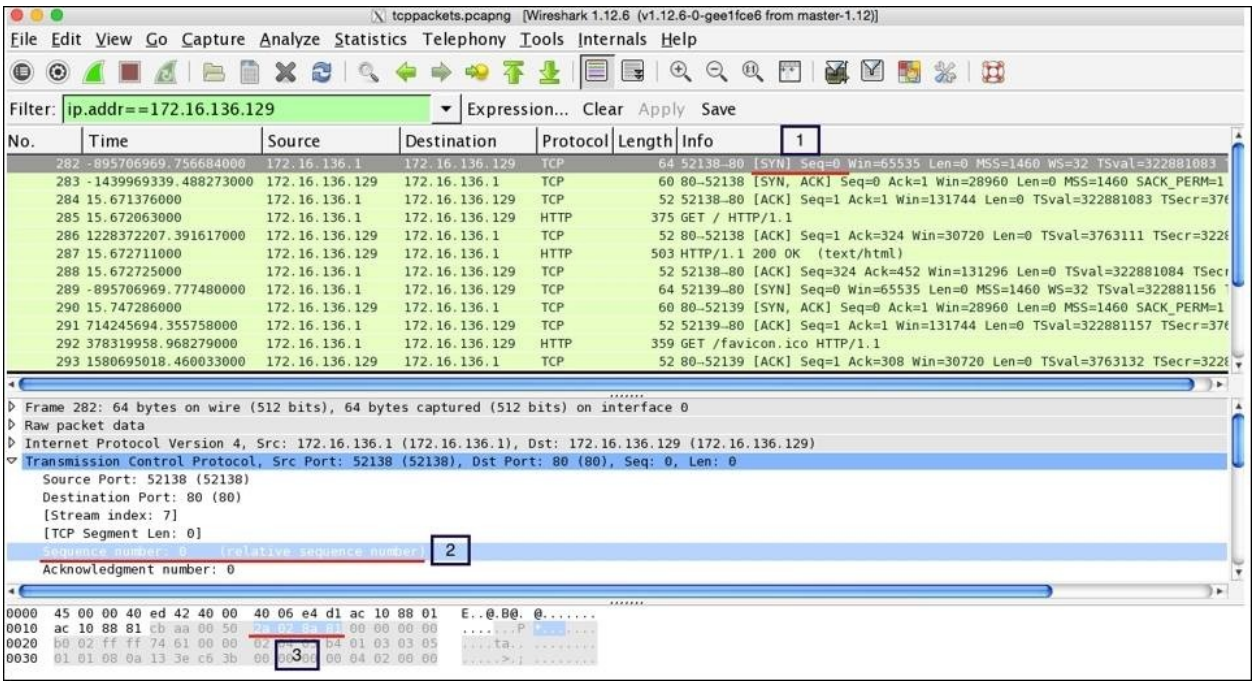
Relative verses Absolute numbers

Wireshark purposefully translates real `SEQ`/`ACK` flag numbers to a simpler format, which makes it significantly easier for us to keep track of data sent across the wire. For instance, I've a web server at `172.16.136.129` and a client at `172.16.136.1`. Using a web browser, I will try to visit the server that will generate a couple of packets, which will be captured by Wireshark. Refer to the following screenshot illustrating the same packets generated for the session.

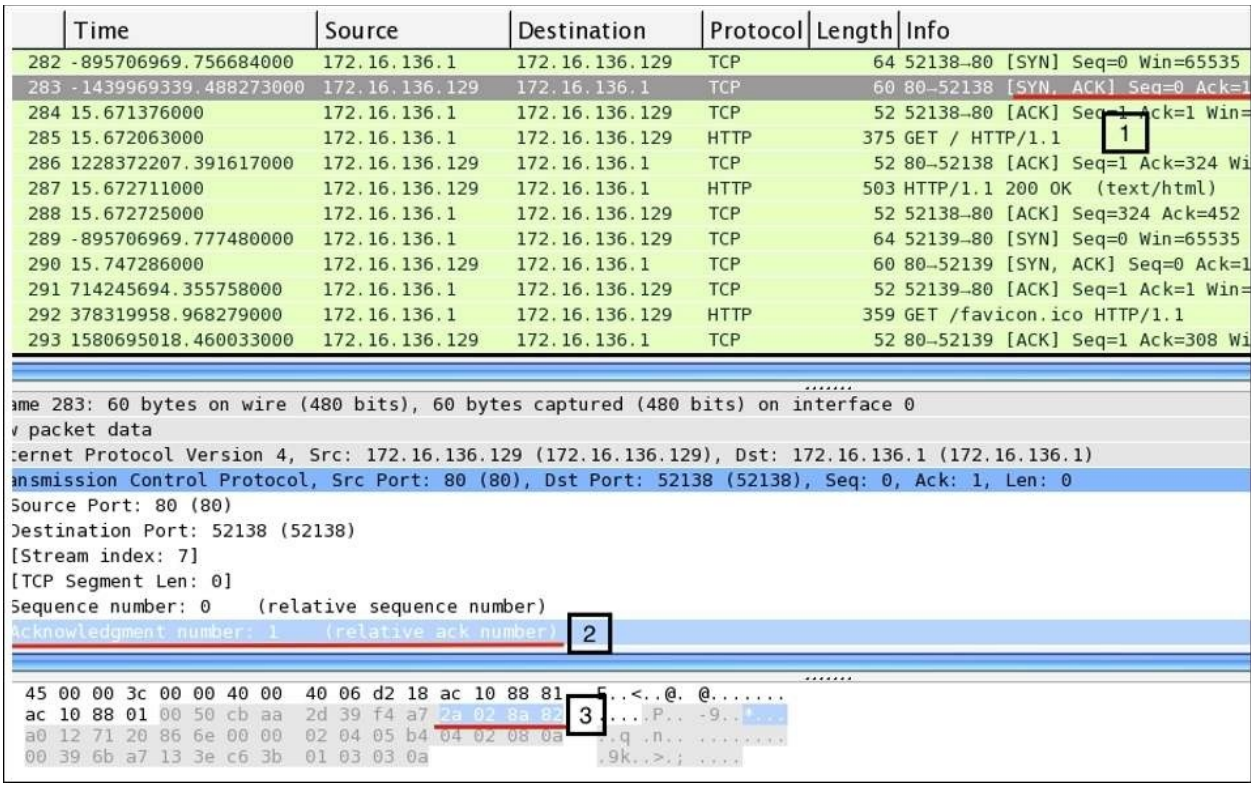
I have selected the first packet generated for the session in the list pane and its corresponding details in the packet. The details pane and bytes pane can be seen highlighted as follows:

- **1**: In the list pane, it can be observed that the `SEQ` number assigned for the `SYN` packet to begin communication is `zero`.
- **2**: In the details pane, we can see that the number `0` is a relative sequence number, which is not the real `SEQ` number and has been changed for our perusal by Wireshark.
- **3**: In the bytes pane, we can see that the corresponding hex value for `SEQ=0` is `0x2a028a81`, which is equivalent to `704809601` in decimal.

So, the real `SEQ` number is `704809601`, which was converted to `0` to make our analysis easy.



According to our analysis, the `ACK` value that we must receive should be `704809602` (incremented `SEQ` value with `1`). Let's verify the same using the next packet and its corresponding related information using the details and bytes pane. Refer to the following screenshot for illustration:



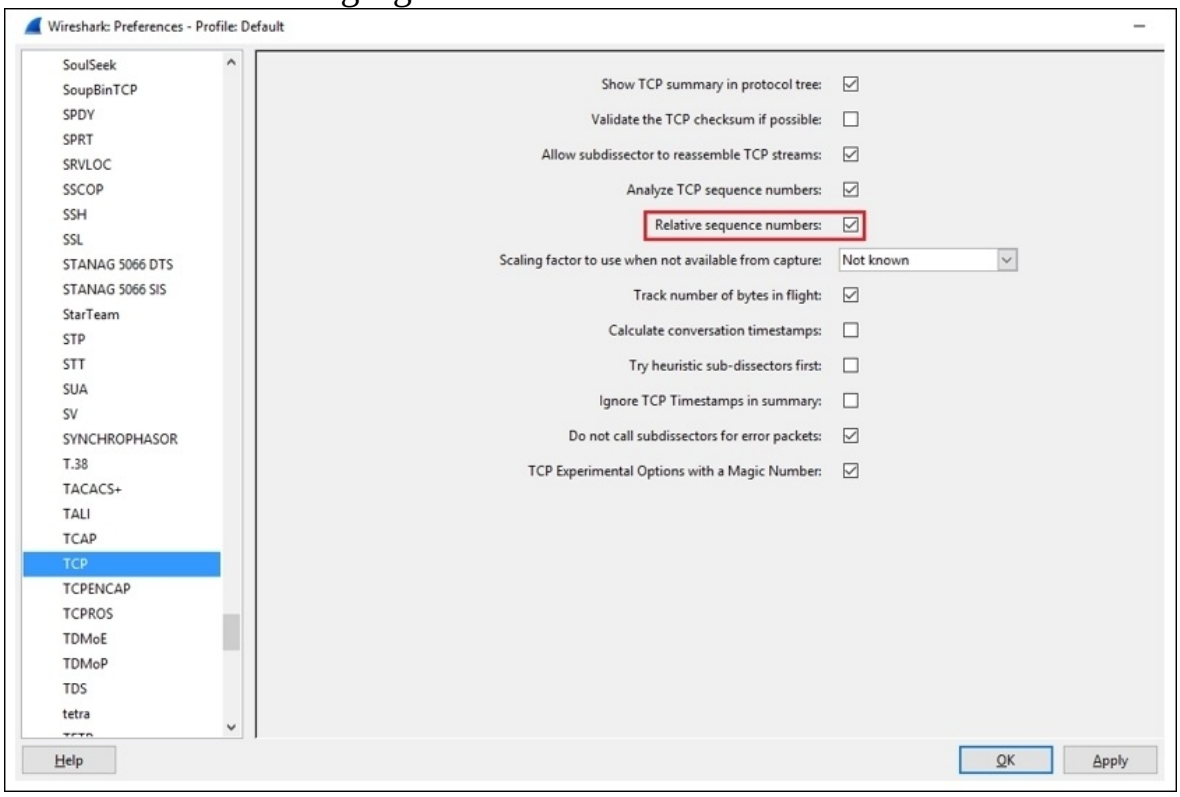
Refer to the following list to understand what each pointers highlights:

- The second packet I selected is the `SYN, ACK` packet that the client received from the server. It contains the `SEQ=0` and `ACK=1` (relative numbers) servers.
- The related information for the packet `2` in the communication is shown in the details pane and the bytes pane. If you observe, in the details pane, the `ACK` server sent for the client's request is `1`.
- The hex value for the `ACK` received is `0x2a028a82`, which is equivalent to `704809602` in decimal. This is the same value that we should be expecting.

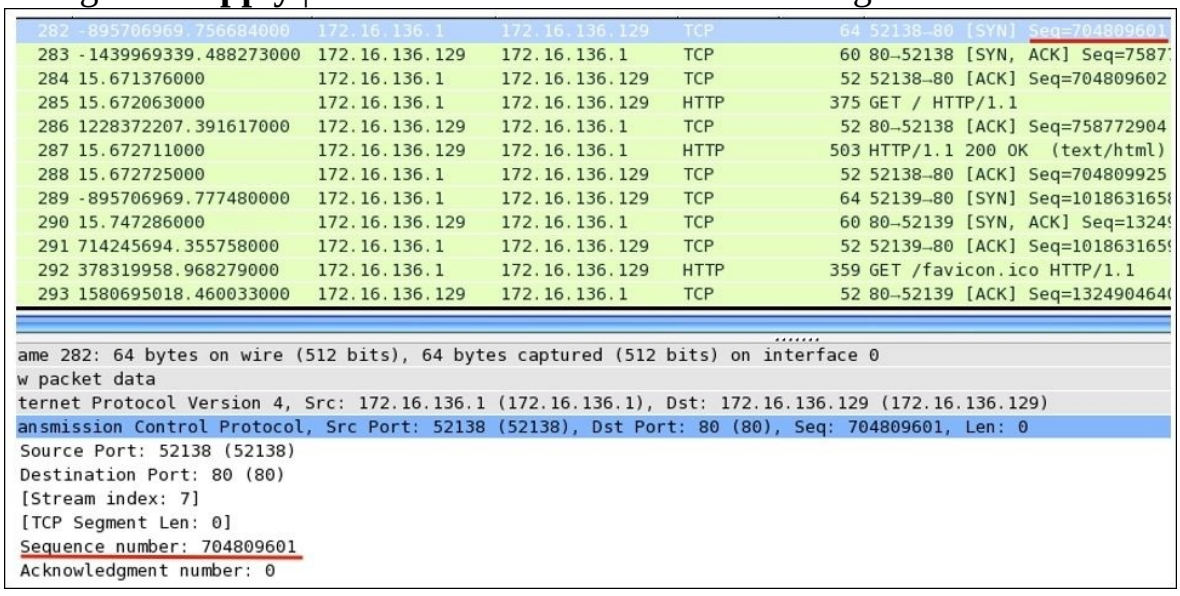
Now, it would be easy for you to check the absolute numbers translating them from their given hex values. There is one more interesting way by which we can customize the numbering system, where we can view the real absolute numbers directly in the list pane and the details pane. Follow these steps to activate and deactivate it:

1. Navigate to **Edit** | **Preferences** in the menu bar.
2. Expand the **Protocol** tree and look for TCP.

3. Remove the checkmark from the **Relative sequence numbers** option, as shown in the following figure:



4. Navigate to **Apply** | **Ok**. That's it. Refer to the following screenshot:

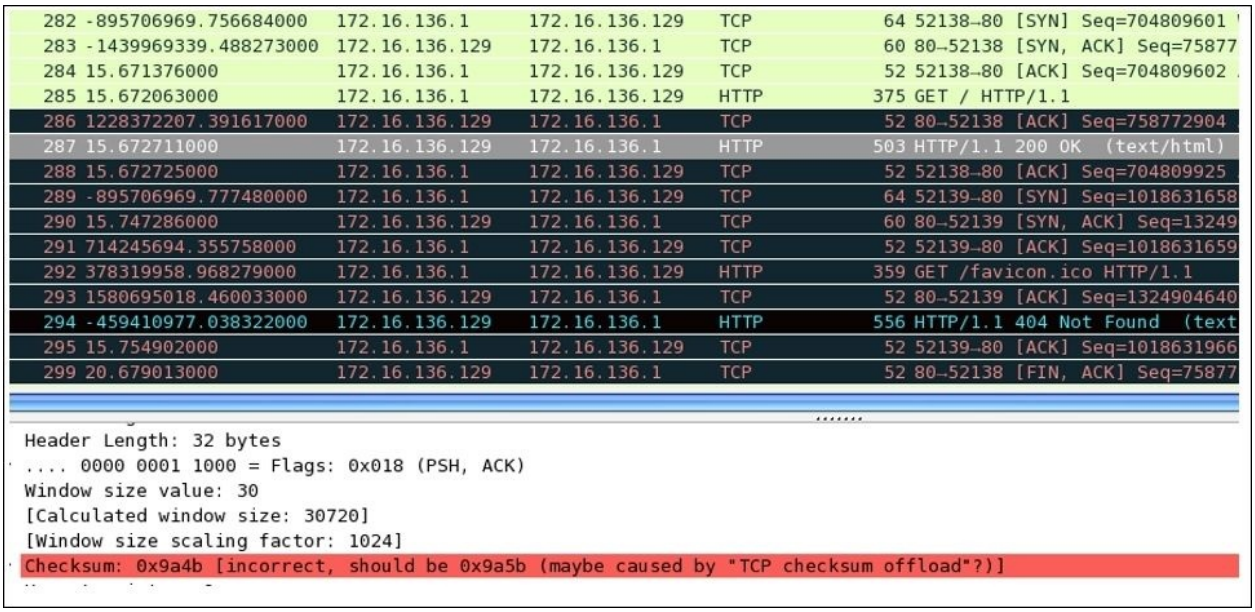


As we analyzed, the first packet in the TCP handshake process has an `SEQ` number `704809601` as an decimal equivalent. Now, after deactivating the **Relative sequence numbers** options, we can observe the same in the list and details panes.

There are a few more options that are enabled by default in the TCP **Protocol Preferences** window, which makes the analyses more systematic and advanced. For example, validating the checksum whenever possible and A=analyzing the TCP sequence numbers.

Checksums are generally used during the transmission to ensure the integrity of the data being sent and received. As discussed, there is an extra field in the TCP header. What actually happens is when the sender prepares the packet that needs to be transmitted, the checksum of the packet that contains data is calculated and sent along with the packet. Now, the receiving side will receive the packet and recalculate the checksum using the same algorithm used by the sender. If the checksum value that came along with the packet is identical to the one that the receiver calculated, then the packet is accepted; otherwise, the packet that contains the error (checksum not matched) is discarded and the sender side is not even informed about the error that has taken place. The sender is supposed to know about this by himself. The validation of the checksum is not 100% guaranteed, and even this reduces the performance as TCP packets reassembly won't take place now.

Checksum offloading is a feature that only new network drivers support, where the packets that are ready to be transmitted are passed on to the network hardware that are captured by Wireshark with an empty checksum field that generates the `checksum offloading` error. The reason is that, even before the actual packet transfer happens, Wireshark captures the packet (the packets will contain the valid checksum once the actual transfer happens). This might lead to several confusion. So, the best approach would be to switch off the offloading feature from your interface if available, or to disable the **Validate checksum** feature for TCP protocol preferences. Refer to the following figure that illustrates this:



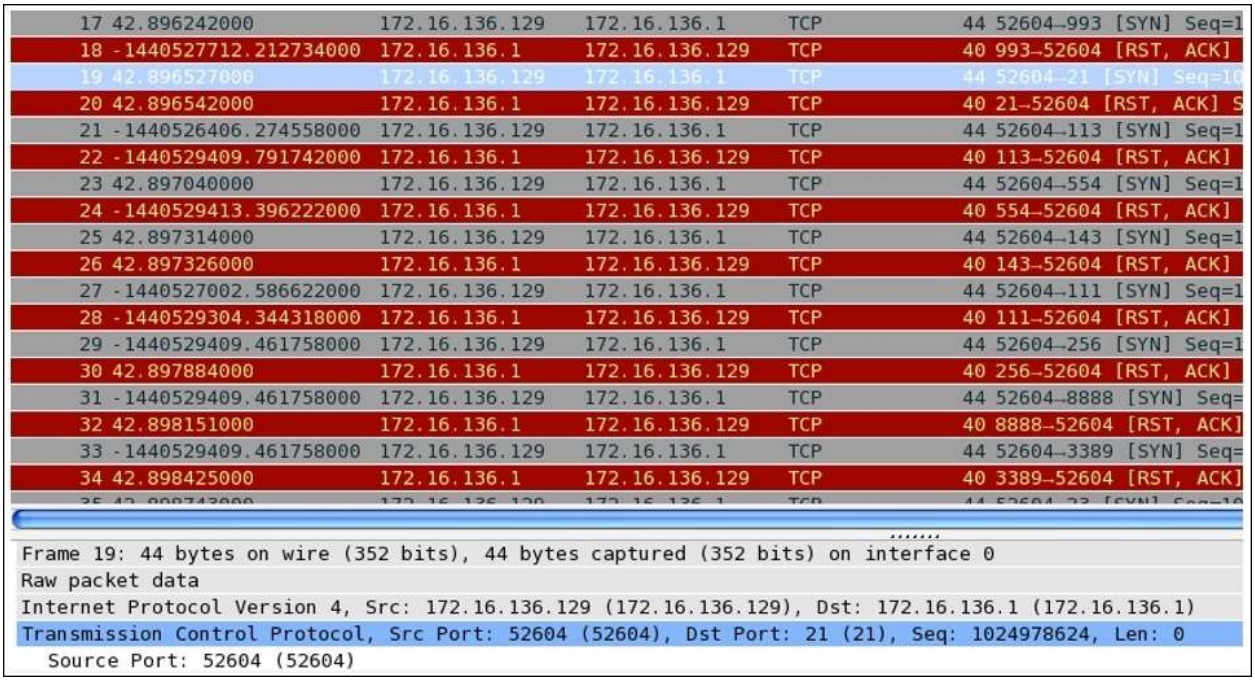
The packets with invalid checksums are displayed with a black background and red foreground color. Look at the error highlighted in red color in the details pane; this states that the checksum is incorrect, and this might be because the checksum-offloading feature is activated. The packets with an invalid checksum cannot be reassembled, and it doesn't look nice (a lot of invalid errors on the screen), so the best option is to deactivate this feature if not required.

Another option that you should know about is the **Analyzing TCP sequence numbers** feature, which keeps track of the `SEQ` and `ACK` numbers and keeps you aware of the various types of errors that can take place during transmission, for example, lost frames, duplicate `ACK`, retransmissions, window scaling, and several others. Turning this feature off will also affect the **Expert Info** dialog, where any of the warnings related to transmission errors and other useful information won't be populated.

Unusual TCP traffic

One of the scenarios that commonly falls under this category is the lost connection or unsuccessful connection attempt scenario, which we have already analyzed in the RST packets section. You might observe several other examples, such as high latencies, due to long-distance communications or queuing up of the traffic. To make the analysis easy and to sort out such problems, use the time column by sorting it, and then, you will be able to figure out large time gaps between the packets at the top of the list pane.

Another example can be where a malicious user is trying to perform a port scan on your network and your firewall responds with `RST` packets to the user to avoid such attacks, or it might also be possible that the port that the malicious user is looking for is closed. A normal scan can generate a lot of traffic and which is quite noisy. This can be easily observed in the list pane of Wireshark. Refer to the following screenshot where I've tried scanning my machine using `nmap` from another device, and it seems quite visible and hence is easy to track:



Observe `Frame 19`, where the port scan initiated by the malicious user sent a `SYN`

packet in order to check whether the port is open or closed. As a result, in our case, port `21` (`FTP`) was closed; hence our machine sent a `RST` packet, which will be used by the port scanner on the other side to display statistics. If the port was open, the malicious user will be notified with `SYN` and `ACK` (refer to the following screenshot), which signify that our machine is open to a connection over the port `21`, and this might become an entry point to the user's malicious attacks.

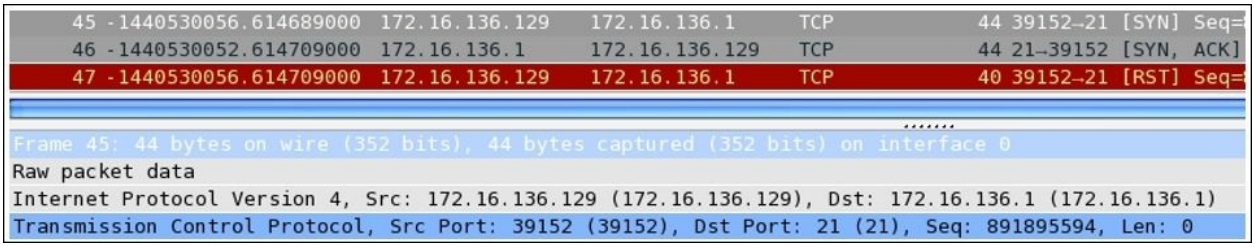


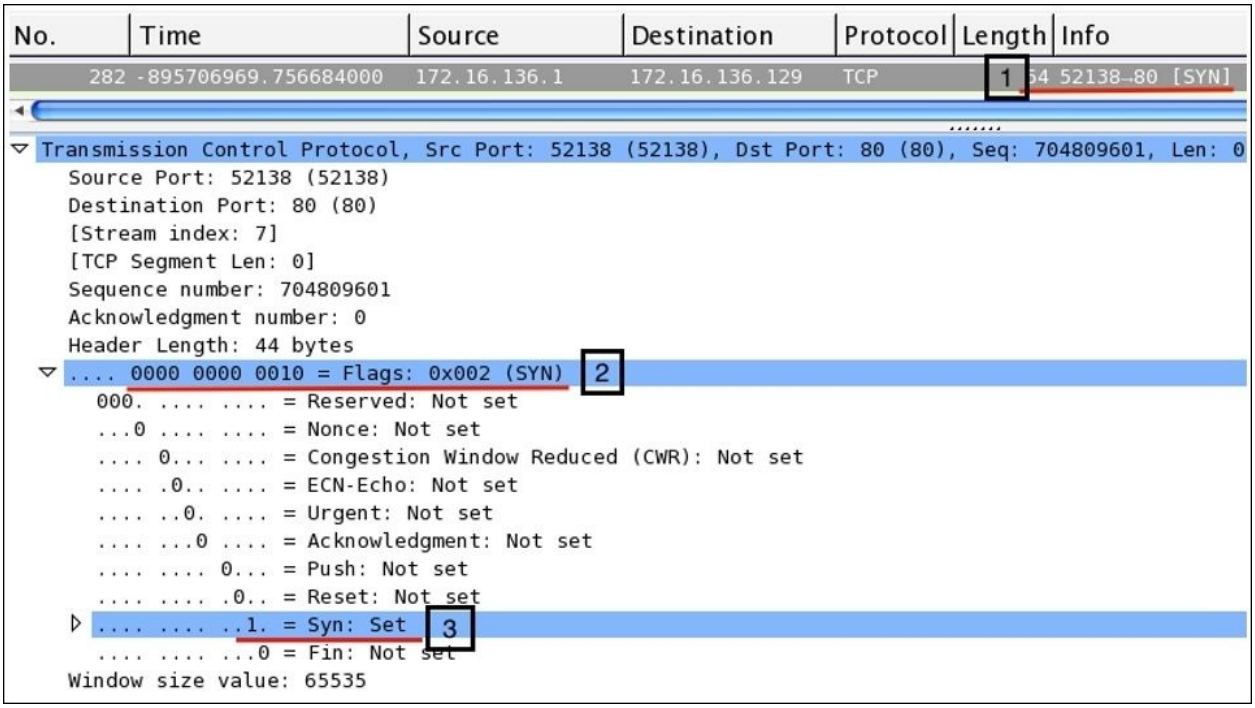
Figure 5.3: Port 21 open, an entry point for malicious attacks

Take a look at `Frame 45`, where the client sent a `SYN` request to the server at `172.16.136.1`, and by this time, the port was open so our server sent `SYN` and `ACK` packets (`Frame 46`), acknowledging the connection initiation attempt with a positive confirmation that the server is open to connection over port `21`.

There can be various scenarios other than this half-open scan (the scan shown in the preceding screenshot is called half open because the client who initiated the connection attempt, would never complete the connection by sending `ACK`, which the server will be expecting). If your basics regarding the packet behavior, connection initiation, completion process, TCP headers, flags in packets, and `SEQ-ACK` numbers are clear, then it would be quite easy for you to point out any unusual form of traffic that is flowing around. There is no such automated tool that can point out these abnormalities until you customize your environment about how to react or alarm you to such traffic anomalies. These are some traffic patterns that you can expect to happen on a regular basis.

How to check for different analysis flags in Wireshark

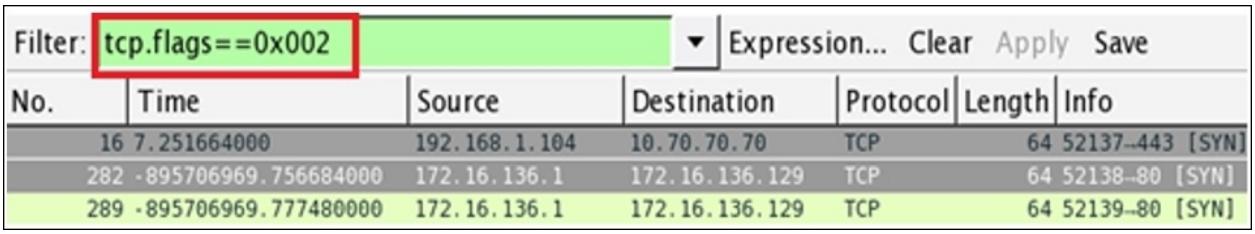
The analysis of the flags present in TCP packets is quite simple while using Wireshark, there is an individual section that is available in the details pane for every TCP packet. Let's take a TCP packet from our previous handshake process that we captured and see how flags are presented in the details pane. Then, we will try to create a display filter corresponding to the same. Refer to the following screenshot that illustrates this:



Now, we will see what each pointer signifies:

- Here, the `SYN` packet sent from the client to the server to initiate the three-way handshake can be seen in the list pane.
- Here, the flags related to the same packet are set and the hex equivalent of `000000000010` is set to `0x002`.
- For the corresponding TCP packet, the `SYN` flag bit is set to 1; the same can be seen in the details pane. The rest of them are still `0`.

Now, if you wish to create a display filter to see only the SYN packets that you have in the trace file, then apply the filter shown here. As a result, you will see only SYN packets present in your trace file. The following figure illustrates the same:



Let's try to create one more filter to view the SYN and ACK packets only in the list pane. Follow these steps to create the filter:

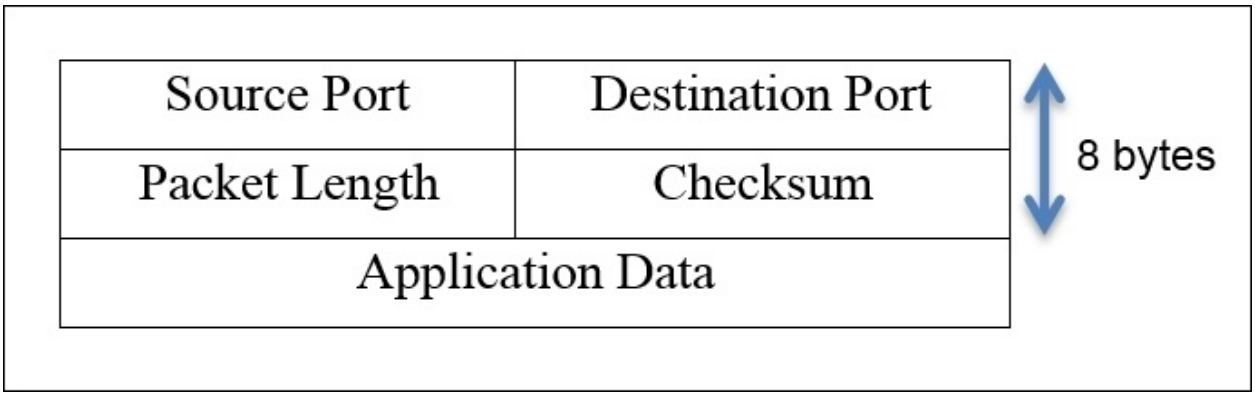
1. Open your trace file.
2. Choose any TCP SYN, or ACK packet.
3. Note the corresponding SYN and ACK hex equivalent values for the flags set.
4. Create your filter using the hex equivalent that you have. Your filter must look something like what is shown here.

The User Datagram Protocol

As defined in RFC 768, a UDP is a connection-less protocol, which is great for transmitting real-time data between hosts and is often termed as an unreliable form of communication. The reason for this is that UDP doesn't care about the delivery of packets, and any lost packets are not recovered because the sender is never informed about the dropped or discarded packets during transmission. However, many protocols such as DHCP, DNS, TFTP, SIP, and so on rely only on this. The protocols that use a UDP as a transport mechanism have to rely upon other techniques to ensure data delivery and error-checking capabilities. And these protocols are inbuilt with such features, which can provide some level of reliability during the transmission. A point that we should not to forget is that a UDP provides faster transmission of packets as it is not concerned about the initiation of the connection or graceful termination as seen in the TCP. That's why a UDP is also referred to as a transaction-oriented protocol and not a message-oriented protocol like a TCP.

A UDP header

The size of a usual UDP header is 8 bytes; the data that is added with the header can be theoretically 65,535 (practically 65,507) bytes long. A UDP header is quite small when compared to a TCP header; it has just four common fields: **Source Port**, **Destination Port**, **Packet Length**, and **Checksum**. Refer to the UDP header shown here:

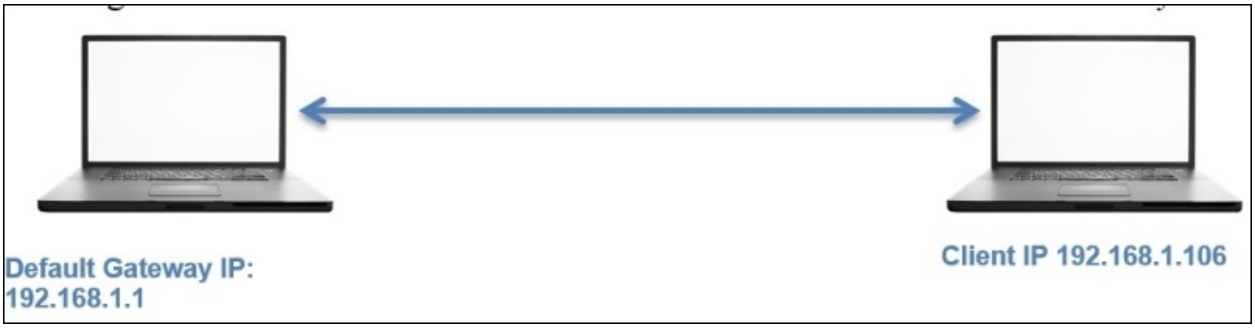


- **Source port**: This is the port number used by the sending side to receive any replies if needed. Most of the time, in a TCP and UDP, the port number chosen to be the part of the socket is ephemeral. On the other side of the communication, the port number comes in the category of well-known port numbers.
- **Destination port**: This field of the header identifies the port number used by the server or receiving side, and all data will be transmitted to this port. This port number is assigned to a particular service by IANA, and definitely, it is permanently assigned to the same service specifically. For example, port `53` is for DNS and cannot be assigned to any other service (not advisable).
- **Packet length**: This field specifies the length of the packet, starting from the header to the end of the data; the minimum length you will observe will be 8 bytes every time, that is, the length of the UDP header.
- **Checksum**: As discussed earlier, checksum is performed over data, that is, the packet of the packet to ensure data integrity that is what is sent from the sender side is the same what receiver got and to verify this there are couple of checksum algorithms which comes to the rescue. Sometimes, while working with a UDP, you will see that the checksum value is `0` in the packet we received. This means that the checksum is not required to be validated.

How it works

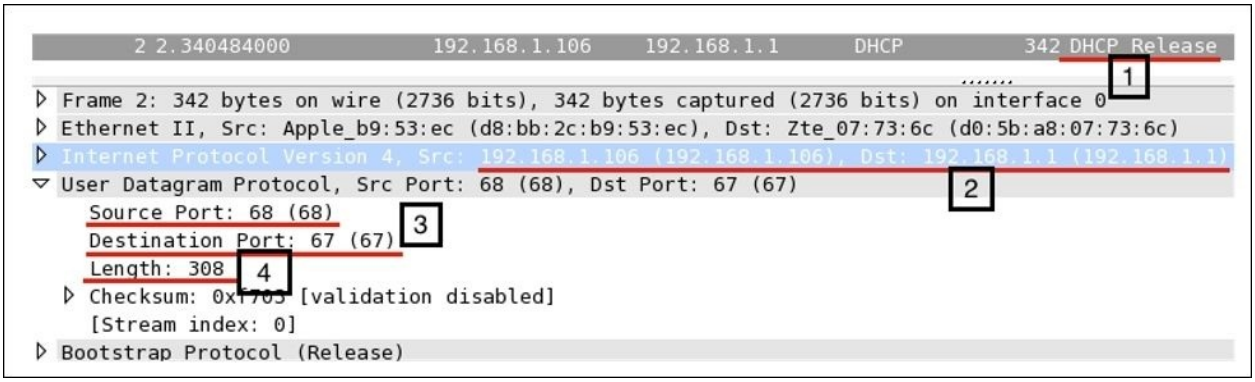
To understand the way a UDP works, let's go ahead and analyze some of the protocols that use a UDP as a delivery protocol. First, I would like to discuss DHCP, and then we will see DNS traffic as well. We actually saw UDP traffic before as well while we were going through VOIP and SIP analysis.

For analysis purpose, I have a default gateway configured at `192.168.1.1` and a client at `192.168.1.106`. Using the client, I will try to generate DHCP and DNS traffic, which will be captured in Wireshark, and then, I will try to dissect each protocol's communication process as well as the different components utilized during the whole session. Refer to the following network architecture that I have:



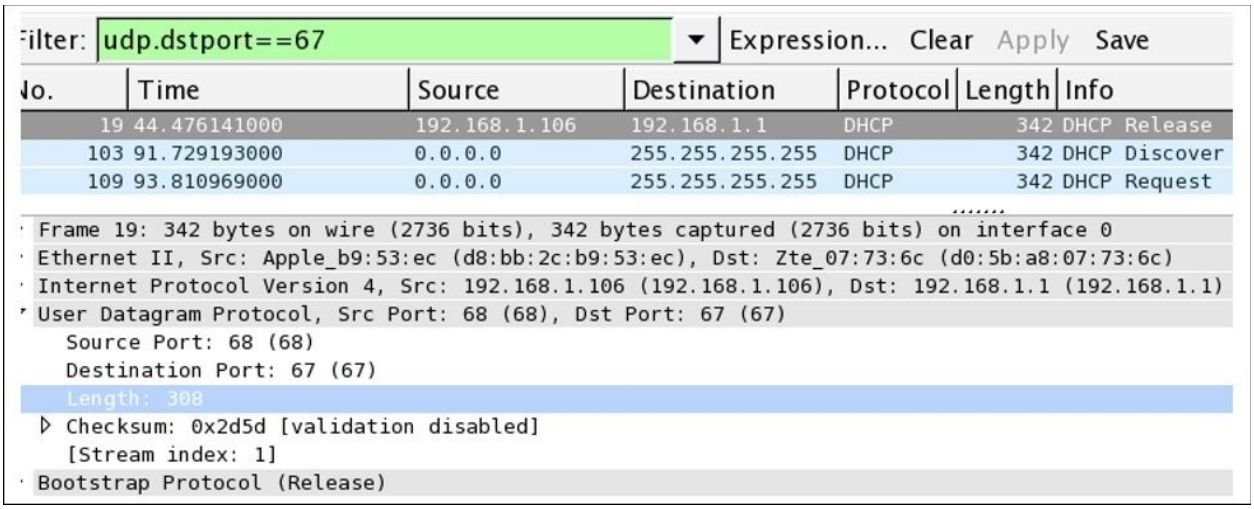
The DHCP

The most common and important protocol that assigns IP addresses to devices and makes them network compatible is **Dynamic Host Configuration Protocol** (**DHCP**). Now, from the client, I will try to release the IP address that the client already holds, which will generate a DHCP packet, and the same will be captured by our sniffer. Look at the following figure to understand this:



In the list pane, we can see a DHCP release packet that was generated implicitly by the client in order to release the current IP address (I used the `dhclient -v -r` command on the Linux terminal to release the IP address, but be careful while using this command as it may disconnect your machine from the network, hence making it incompatible for network communication). The client from the IP address `192.168.1.106` to the server at `192.168.1.1` initiates the request. The port numbers used by the client and server in case of DHCP are permanent, these won't be changed in your case either unless they are manually configured.

The DHCP server port number is `67` and the DHCP client port number is `68` by default; you can see the same in the preceding figure (highlighted as **3**). There is a fourth field that I have highlighted, the packet length field, which specifies the length of the packet starting from the first byte until the end of data in the packet. However, out of 308 bytes, 8 bytes show the length of the UDP header and the remaining 300 bytes represent the application data that is appended. Interestingly, if a machine is power cycled, it will request the DHCP server to allocate an IP address. This, as a result, will generate a couple of packets related to the DHCP request, release, and offer and various others that will also use the UDP as a transport mechanism.



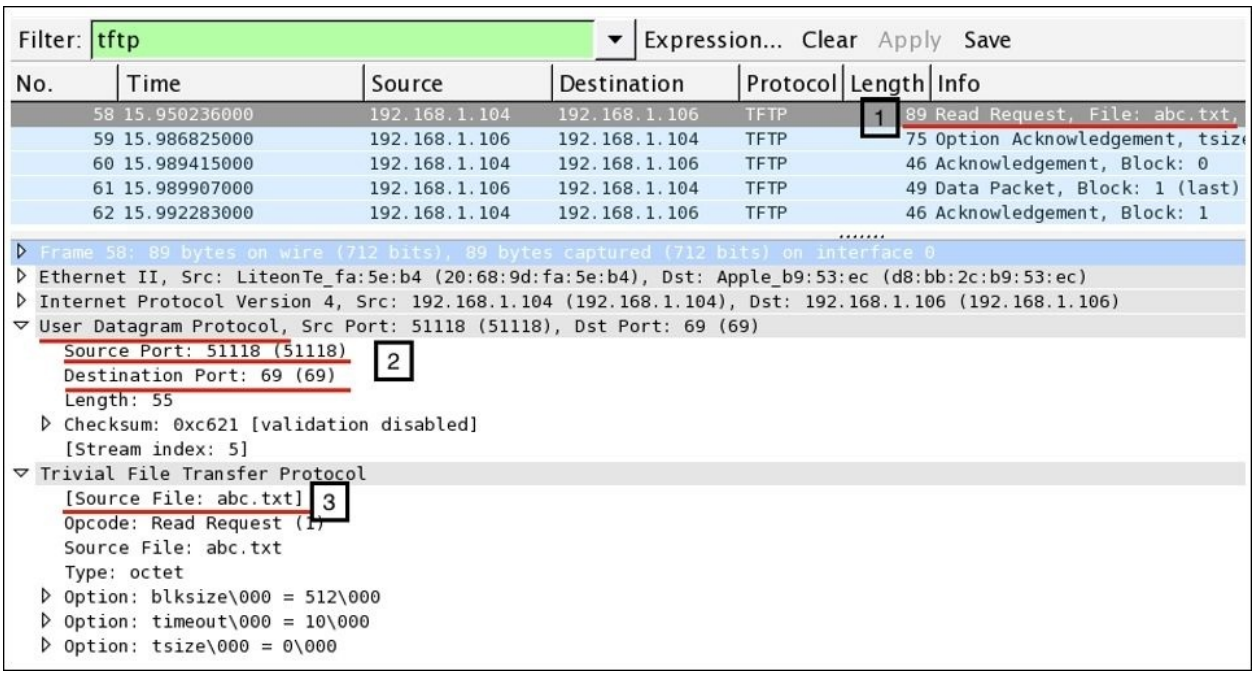
I filtered the packets listed to show only DHCP packets using the `udp.port==67` filter; as a result, only DHCP packets will be listed in the list pane.

The TFTP

The **Trivial File Transfer Protocol** (**TFTP**) is a lightweight version of the FTP that is used to transfer between hosts. Unlike the FTP protocol, TFTP does not ask users for any credentials. A TFTP uses a UDP as a transport mechanism. Most commonly, a TFTP is used in LAN environments, and when dealing with manageable devices such as switches and routers, network administrators do use TFTP servers to take a back up of configuration files and to update the firmware running in those devices. A TFTP is also used by security professionals to transfer files from their system to yours in order to escalate the privileges (gaining more rights on a compromised system).

I have a TFTP server running at `192.168.1.106` and a client running at `192.168.1.104`. There is a text file `abc.txt` that I've created on the server, and the client will try to download the same. And our sniffer in place will capture the traffic that is generated.

The traffic generated due to the transaction that takes place between two hosts is successfully captured and the packets corresponding to it are shown in the following figure:



Now, let's see what each pointer signifies:

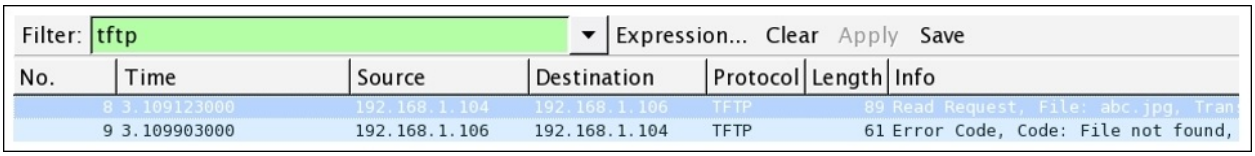
- This shows that the transfer of the packet is initiated as soon as the client requests the `abc.txt` file. The request frame can be seen in the list pane.
- As discussed, a TFTP uses a UDP for a transport mechanism. The related details for the request are shown in the details pane, which states that the

request was initiated from a ephemeral port number from the client destined to port `69` on the server (`69` is a well-known port to the TFTP protocol).
- The request was specific to the `abc.txt` file that is also shown in the details pane in the TFTP protocol section.

You must be wondering about the acknowledgement packets that are shared between the two hosts. As we discussed, a UDP is an unreliable form of communication, so why are we seeing ACKs in a UDP? The reason is that the TFTP server I am using has some kind of inbuilt reliability feature. Even on the client side, over the standard console, after initiating the request, I received quite interactive messages from the server, such as the file of size 3 bytes has been transferred successfully, and various other details were listed along with the message. The interesting thing to know here is that port `69` was only involved in the first packet, and the rest of the packets were sent and received by the acknowledging feature that the server is embedded with. So, the statement that some protocols use a UDP as a transport protocol and have their own inbuilt feature to ensure delivery is true, as we have just witnessed.

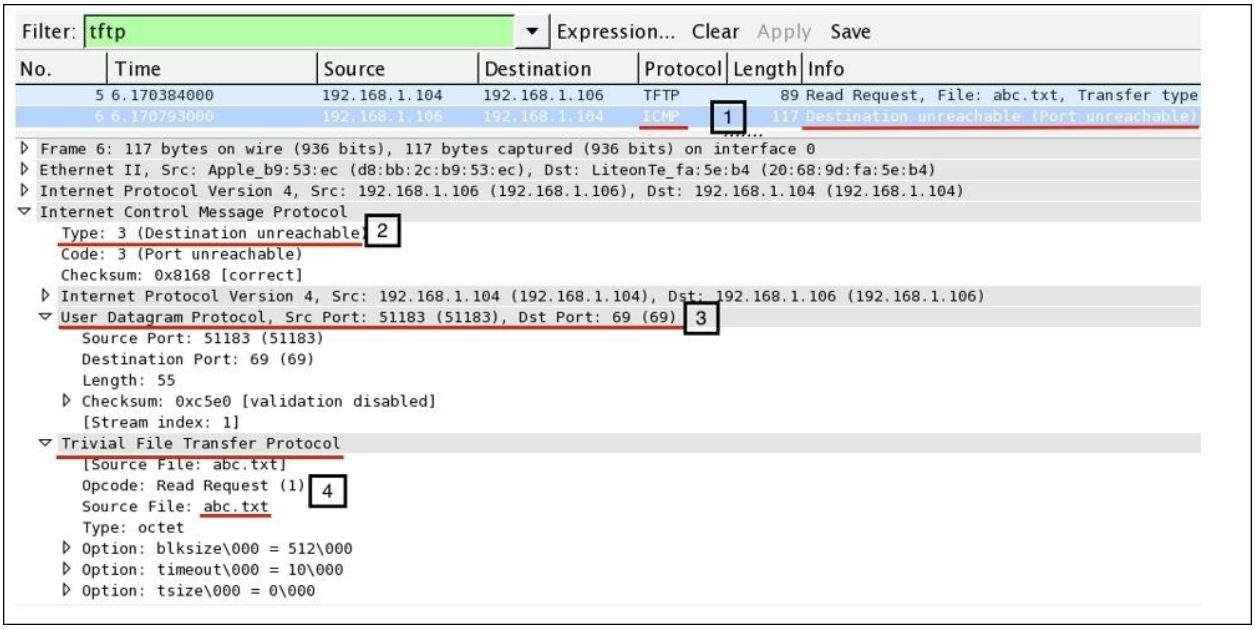
Unusual UDP traffic

Suppose that the resource we are looking for is not available on the server. How will traffic look like then? Refer to the following screenshot to understand this:



As seen in the preceding screenshot, the client requested an invalid resource that the server wasn't able to locate and hence returned with an error code and the summary message `File not found`. The same message was shown over the standard console to the client.

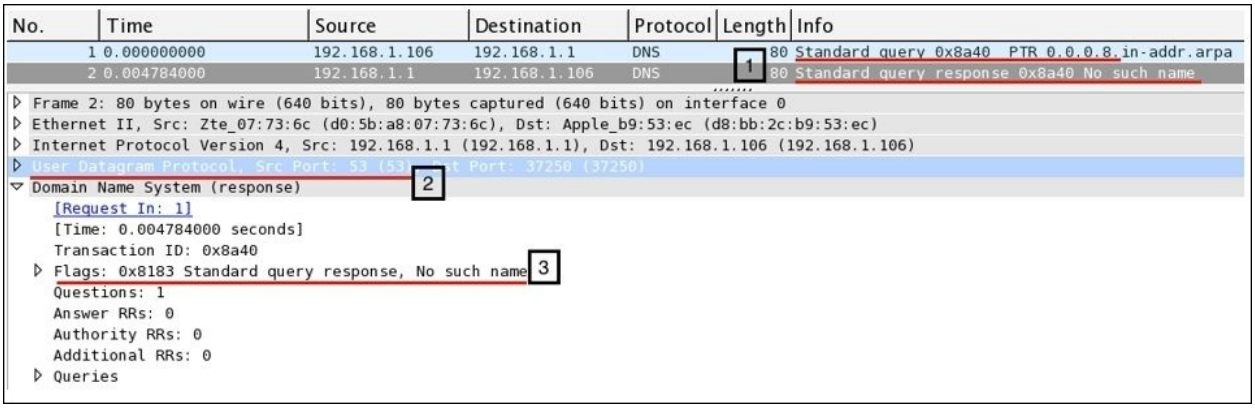
Sometimes, it is also possible that the server daemon may not run and the client may request a certain resource. In such cases, the client would receive the `ICMP destination unreachable` error with the error code `3`. Refer to the following figure for the same:



Now, we will see what each pointer signifies:

- The server returned with an `ICMP destination unreachable` message when the TFTP server daemon was not functional
- The client received an error code of type `3`
- The details regarding the request were mentioned in the reply under the UDP protocol section, which stated that the request was sent to port `69`, which was currently nonfunctional
- The requested resource was shown under the TFTP protocol section

Unusual DNS requests are also very often seen when a client initiates a request to look for name servers associated with an address. It would look similar to the one shown in the following figure:



Now, we will see what each pointer signifies:

- **1**: As seen in the list pane, the client at `192.168.1.106` initiated a request to look for the address `8.0.0.0` and received a response in `Frame 2`
- **2**: The request was sent to the default gateway that holds the DNS cache
- **3**: The gateway responded with a `No such name` error

There can be multiple scenarios where you will see unusual traffic related to a UDP. The most important thing to look for is TFTP traffic, which might be generated because of a the TFTP client in your network. It may be malicious traffic that you would like to make a note of.

Summary

TCP is a reliable form of communication that has features like a three-way handshake and a tear down process ensures the connection is reliable and interactive.

A TCP header is 20 bytes long and consists of various fields such as source and destination port, `SEQ` and `ACK` numbers, offset, window size, flag bits, checksum, and options. The presence of various flags and header fields let the sender and receiver be sure about the delivery as well as the integrity of the data being sent.

The `SEQ` and `ACK` numbers are used by TCP-based communications to keep track of how much data is being sent across between the hosts taking part.

A UDP is a connection-less protocol that is a nonreliable means of communication over IP, where the lost and discarded packets are never recovered. A UDP does provide us with faster transmission and easier creation of sessions. A UDP header is 8 bytes long, which has very few fields such as source and destination port, packet length, and checksum. At the end application, the data is appended.

Common protocols such as DHCP, TFTP, DNS, and RTP mostly use a UDP as a transport mechanism, and these services are some of the major services that we deal with in our everyday life. To make the connection reliable, some of these protocols support their own version of acknowledging features that comes inbuilt.

In the next chapter, you will learn the basics of wireless traffic, how to decrypt wireless traffic, and the anomalies that may follow.

Practice questions

Q.1 List at least five differences between TCP and UDP protocols.

Q.2 Capture a three-way handshake and tear down packets using your own FTP server.

Q.3 Explain the purpose of window scaling and checksum offloading and state their corresponding significance in terms of TCP communications.

Q.4 In what way can TCP-based communication can recover from a packet loss or unexpected termination? Imitate any scenarios that can generate such traffic.

Q.5 Create a display filter to show only `TCP FIN` and `ACK` packets sent to your machine from your default gateway in the list pane.

Q.6 What is the difference between the absolute and relative numbering system used by Wireshark in order to keep track of packets?

Q.7 What is the purpose of the options field at the end of the TCP header and what kind of arguments does it contain?

Q.8 There is one more way through which you can create filters to view a packet with a specific flags set. Without providing the HEX equivalent, figure out what it is and how you can filter a packets set with a PSH flag set using the same technique.

Q.9 Find out why the length of data can only be 65507 bytes while working with a UDP.

Q.10 What kind of packets you will see in a list pane if the server daemon for a TFTP is not running?

Q.11 Try performing a zone transfer on your locally configured DNS and capture the traffic for analysis. What interesting facts did you notice about the packets? Explain them in brief.

Chapter 6. Analyzing Traffic in Thin Air

In this chapter, you will learn how to analyze wireless traffic and pinpoint any problems. You will also learn how to analyze wireless traffic using Wireshark. The following are the topics we will cover in this chapter:

- Understanding IEEE 802.11 traffic
- Analyzing normal and unusual behavior
- Lab up—wireless communication
- Decrypting encrypted wireless traffic
- Lab up—decrypting WEP and WPA traffic
- Practice questions

We start from the basics such as how WLAN traffic gets generated and various essential elements responsible for handling the wireless transmission between hosts. Then, moving ahead, we will analyze the usual and unusual forms of packets that can be seen in Wireshark. Side by side, we will identify anomalies and regular traffic patterns. We will also discuss how you can decrypt wireless (WEP) traffic using Wireshark, which can definitely give an advantage while auditing WLAN environment.

What we are going to witness is not much different from the wired networking that we saw earlier; here, we will be quite concerned with the medium through which packets are flying around us. The two layers at the bottom of the OSI model are important as they represent the data link and the physical layer. The data link layer is divided into two parts: **Logical Link Control** (**LLC**) and **Media Access Control** (**MAC**).

Understanding IEEE 802.11

At the **Institute of Electrical and Electronics Engineer** (**IEEE**), there are several committees working together on several projects, and one of these is 802, which is responsible for developing LAN standards. A free white paper can be downloaded from the IEEE website based on 802 standards. Specifically, 802.11 contains WLAN standards. If you want to analyze what normal traffic looks like, you should be aware of the standards and the present working technologies within 802.11.

There are a couple of 802.11 standards, but the few important ones that we should know about are 802.11b, 802.11a, 802.11g, and 802.11n, which are explained in the following list:

- **802.11**: This only supports a network bandwidth of 1-2 Mbps. This is the reason why many 802.11-compatible devices have become obsolete. Hence, it became necessary to develop other 802.11 standards.
- **802.11b**: This specification uses a signaling frequency of 2.4 Ghz that is similar to the 802.11 standard. A maximum of 11 Mbit transmission rate can be achieved over a 2.4 Ghz band using b specification. As most of the home appliances (microwave, cordless phones, and so on) work over a 2.4 Ghz spectrum, it causes quite dense interference and congestion during WLAN packets transmission. To avoid the interference, the access points can be installed at a reasonable distance. The 802.11b band is divided into 14 overlapping channels, where every channel has 22 Mhz widths. In one instance, there can be a maximum of three non-overlapping channels operating at the same time. This space separation is necessary and required in order to let the channels operate individually. One device can be part of one channel at a time; the same follows when you listen to the packets. Practically, it is possible now to sniff more than one channel at a time, which is facilitated through various tools that are now available; one of them is *Kismet*, which can sniff up to 10 channels at regular short intervals.
- **802.11a**: This is based on **Orthogonal Frequency Division Multiplexing** (**OFDM**) that was released in 1999 and supports a maximum transmission rate up to 54 Mbps, which also gives us an advantage over 802.11b congested bands. This specification was developed as a second standard to 802.11 standards. It is commonly used in business environments, but

because of its high cost, the *b* specification is not best suited for home environments. Though it supports higher speeds around 5 Ghz spectrums than 802.11b, the range of devices falls short if it is configured with *a* specification. The capability of bypassing the obstructions that comes in between is not better than 802.11b. There is no channel overlap that happens in 802.11a. A higher regulated frequency helps in preventing the interferences caused by devices that work on 2.4 Ghz spectrums.

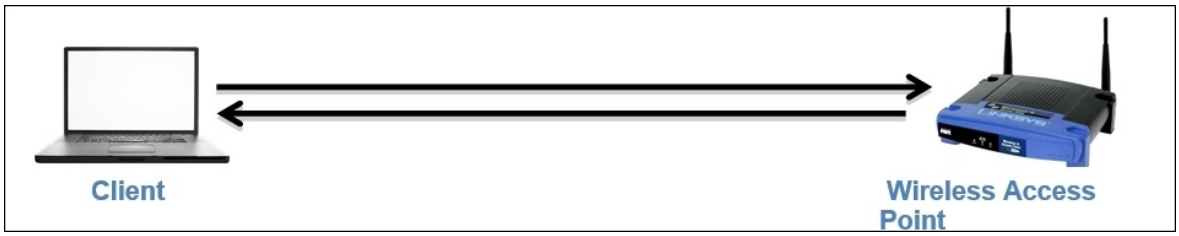
- **802.11g**: Somewhere around the middle of 2002, this specification came into existence, and this tried combining the best features of 802.11a and 802.11b. The signaling frequency used here is 2.4 Ghz, and the bandwidth it supports is upto 54 Mbps. Due to the 2.4 Ghz frequency in use, the range parameter that suffered a decline was improvised. The 802.11g also supports backward compatibility, which means that all 802.11g access points will support network adapters using 802.11b and vice versa. A strong point in this specification is: it won't get easily obstructed.
- **802.11n**: To improve further, the wireless N was introduced. The key area where the improvement was carried on is the range and the transfer rates. The base technology that is implemented to make all this possible is **Multiple-Input Multiple-output** (**MIMO**) communication. There are multiple antennas fitted into the access point that are used to send, receive, and bounce off the signals. This enables a channel frequency of 40 Mhz. The final version of this specification, which was released in 2007, stated a transfer rate up to 600 Mbps. It can be configured with 2.4 or 5 Ghz (if the access point is compatible with both); it can use both frequencies at the same time, thus enabling backward compatibility with network adapters. A maximum of four antennas can be used with the MIMO technology. Once all of this starts working together, users can experience fastest speed and maximum signaling range, and it's not much affected by another device working on the same frequency band. If this network type gets inferred, then it will other specifications such as 802.11b/g.

Various modes in wireless communications

WLANs uses the **Carrier Sense Multiple Access and Collision Avoidance** protocol (**CSMA/CA**) to manage the stations sending data, where every host that wants to send data is supposed to listen to the channel first, that is, if it is free, then the host can go ahead and send the packet; if not, then the host has to wait for its turn. This is because the same medium is being shared by every host, thus avoiding collisions that might happen if two hosts start transmitting at the same time, as a result making the performance of the network go slow and more prone to errors. The 802.11 architecture is composed of several components such as a **station** (**STA**), a wireless **access point** (**AP**), **basic service set** (**BSS**), **extended service set** (**ESS**), **independent basic service set** (**IBSS**), and **distribution system** (**DS**).

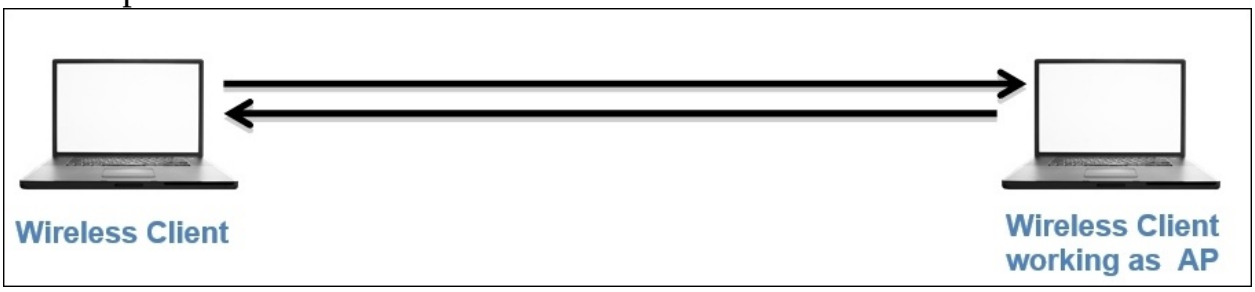
There are four common modes of association between the STA and the AP, which are as follows:

- **Infrastructure/managed mode:** A wireless network environment where two devices wish to connect an STA and an AP to share data and network resources is termed as the infrastructure mode. An AP is defined with a **Service Set Identifier** (**SSID**), which is actually just a name given to the access point for identification purpose (for security reasons, sometimes, broadcasting an SSID can be disabled, which will prevent your wireless network from being discovered by unintended users). For example, once you start scanning for available Wi-Fi networks around you to connect to, you'll be shown multiple network names, from which you are supposed to choose a network that you know about. All these names of networks are called SSID. Another useful term to know is **Base Service Set Identifier** (**BSSID**), that is, the access point's MAC address. By default, every access point is supposed to broadcast the SSID and transmit a beacon frame 10 times in a second to let devices know that they are ready to accept connections. Refer to the following diagram that illustrates this example:



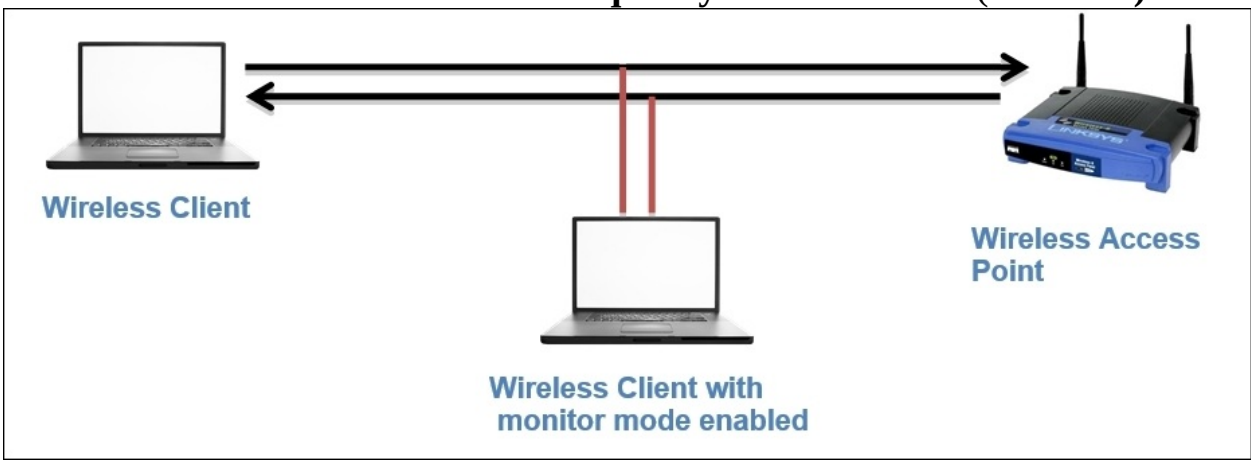
- **Ad Hoc mode**: In this kind of network, a peer-to-peer network is formed where two clients are connected to each other. The packets sent and received by the wireless clients are not relayed to the access point. The clients taking part in this communication now handle the process of sending beacons and processing authentication that a WAP handles in normal scenarios.

- **Master mode**: When the NIC card in your machine lets you become an AP, this is what the master mode is all about. Higher-end devices have a capability to act like access points, and this is possible when NIC cards start working together with a special driver.



- **Monitor mode**: For the purpose of this chapter, this mode is very important. This mode is used to listen to the packets that are flying around; when the monitor mode is activated, your device will stop transmitting and receiving any packets and it will just sit silently and sniff live traffic. If you want to capture

packets from the wireless network concerning 802.11 protocols, then your NIC and the driver that is being used must support the monitor mode. It is quite easy to activate the monitor mode on an OS, such as Linux and MAC; however, with Windows, it becomes quite troublesome to activate the monitor mode. This mode is often termed as the **Radio Frequency Monitor Mode** (**RFMON)**.



After learning the basics of different forms of wireless networking infrastructures that you might note in a production environment very casually, it would definitely become a bit easier for you to choose between the various modes available as per your requirements.

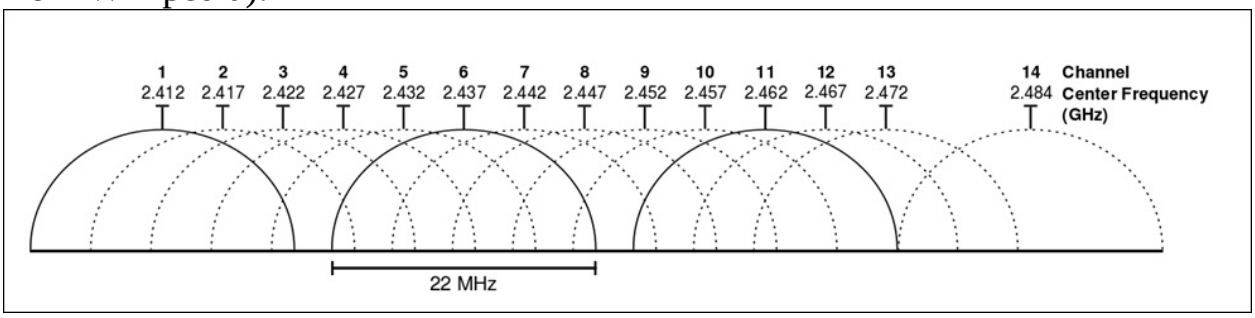
Wireless interference and strength

To better understand the normal traffic pattern, we should be aware of the various usual factors that govern the performance of a wireless network. For example, data packets, associations, and disassociations, signal strength with/without interferences. Our objective while analyzing preceding parameters is to form a baseline that can prove worthy when comparing the traffic patterns with unusual ones. The factor that affects the network performance the most is a different form of interference, which is caused due to various factors such as physical obstructions such as thick walls, roofs; and electronic appliances, such as microwave, cordless phones, and so on.

While dealing with wireless networks, the integrity of data becomes more important because the packets are simply traveling in the air, and anyone with some basic hardware and knowledge of how wireless networks work can sniff

and capture these packets easily. Wireless networks don't have any rescue options to protect the integrity, so using them, you cannot be 100% assured regarding the security of data.

Let's say, for example, you are listening to a particular channel in the spectrum. Normally, you can sniff only one channel at a time, but if the channels start overlapping each other, than it is quite possible that you will see other channel packets in the list pane. As per the normal functioning of a wireless spectrum, the networks that operate close to each other are supposed to choose non-overlapping channels such as `1,6,11,14` to avoid any issues. Refer to the following figure that best illustrates channel overlapping (I used from the same from Wikipedia):



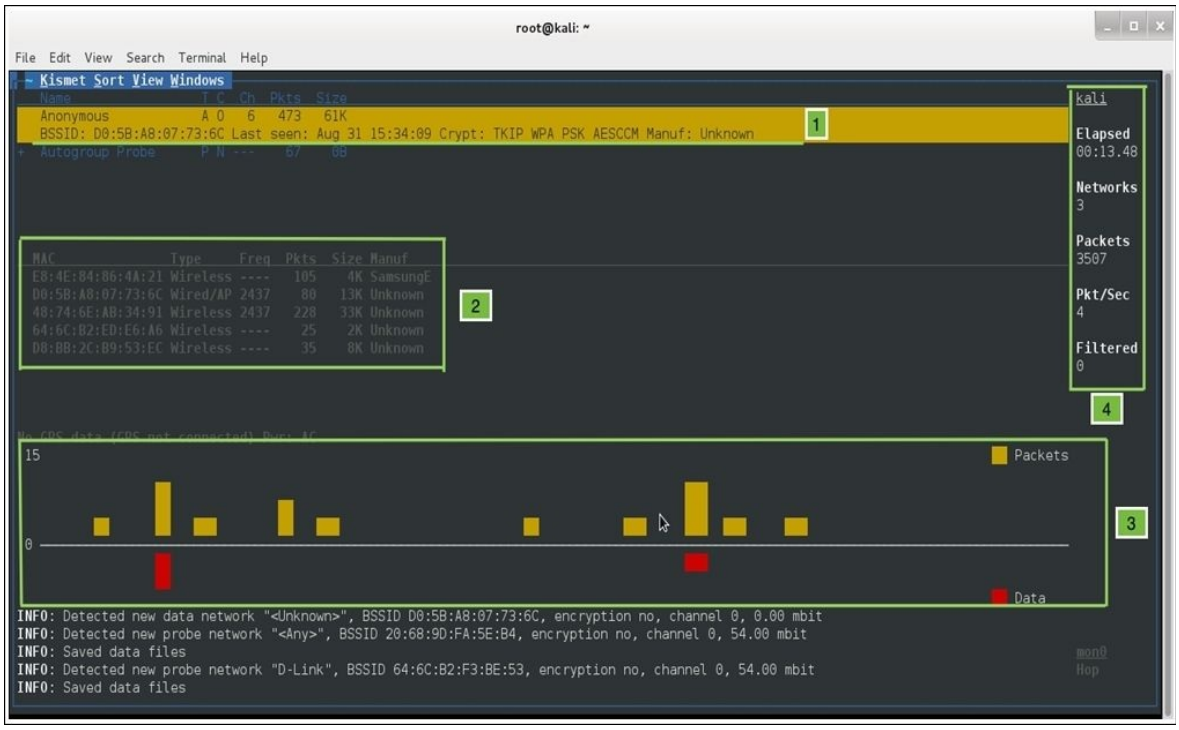
The strength of the wireless network is totally dependent on **Radio Frequency** (**RF**) signals that carry the traffic. Once the wireless signal starts traveling, the strength is supposed to lessen eventually, as it travels farther because of the obstructions that come in between. The device that works over the same RF energy is also responsible for reducing the wireless signal strength. If you are also dealing with such issues, then just using Wireshark to listen on an interface in the monitor mode won't solve the purpose. You need a spectrum analyzer, such as Wi-Spy+Channelyzer, that is paired with a USB (refer to http://metageek.com) adapter and gives you an extra eye over the RF energy form; otherwise, you won't be able to see them. Most of the time, the device emitting high RF energy can be the cause of poor network performance.

To inspect the environment for RF energy, you need to walk down the office on your own with your laptop running a spectrum analyzer, which would be able to detect the RF anomalies that can affect your wireless network performance. The

placement of these analyzers does play an important role in solving the problem. If a host in your office is not able to connect then the best option is to place your analyzer as close to the host as possible in order to perceive the situation from the host's perspective. If various hosts in your office experience a similar problem, then the best option would be to place the analyzer near the access point they are trying to connect to. Depending on the scenario you are dealing with, you can dynamically decide and even manually scan through the office premises to get to know whether there is any RF energy interfering.

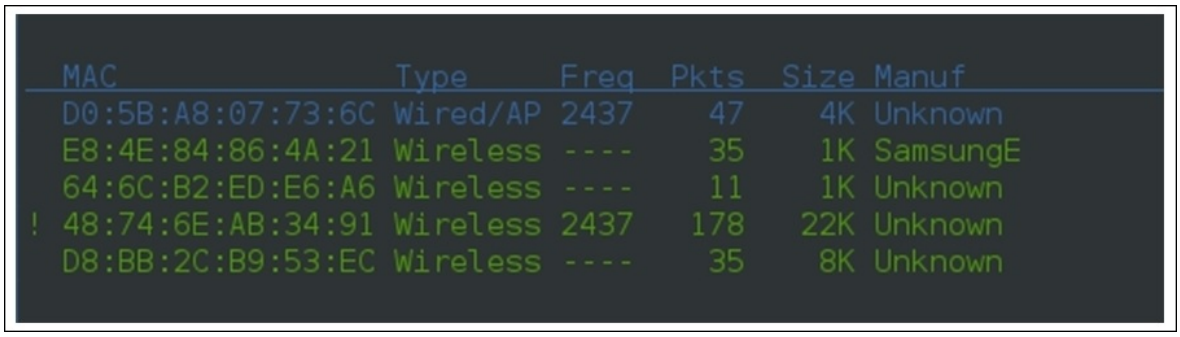
I don't have any special hardware to show you RF energy, but I will use an inbuilt tool from the Kali Linux OS, which will help us fetch various granular details regarding different WLANs available around my premises and all the devices that are connected to Wi-Fi (if paired with a hardware used for spectrum analysis, this can prove really useful). The name of the tool is Kismet, and it is quite efficient in representing details in graphical and various available statistical formats, thus enabling us to know more about the neighborhood (use it for ethical purposes). Follow these steps to use the Kismet tool on Kali Linux:

1. First I enable the monitor mode using the `airmon-ng start wlan0` command (`wlan0` is my wireless interface).
2. Open the terminal and type `Kismet`. You will be asked to set various customization options—do not change any default settings.
3. Once you're asked for the source (interface name) for the Kismet server to capture the packets, specify your interface running on the monitor mode (in my case, this is `mon0`. You can check your interface using the `iwconfig` command).
4. Now, let the tool run on its own for a few minutes; gradually, you will start noticing that a graph is getting plotted for the live traffic captured. You will see various wireless networks around you and most of the associated devices connected with it.
5. In the network section, you will see specific details for the wireless network, such as BSSID, SSID, encryption algorithm used, and so on.
6. The clients' section will show various devices associated with the network. Refer to the following figure of the tool that lists my network and various clients connected to it:



Now, let's see what does each pointer in the preceding screenshot signifies:

- In this part, just below the menu bar, the number of networks that my Wi-Fi adapter is able to scan is shown. The first row shows my home network **Anonymous** and its BSSID, when the network was last seen, the algorithm used, and the manufacturer of the device.
- In this second section, Kismet lists out various devices that are currently associated with the **Anonymous** network, their type (is it an access point or a wireless client), the frequency that the devices are using for transmission, the total number of packets a particular device has transmitted, the size of all packets, and the manufacturer of the device (interestingly, Kismet was able to identify one device manufacturer that is currently associated with my network, as shown in the first row). Refer to the following screenshot that shows the device section separately:



	MAC	Type	Freq	Pkts	Size	Manuf
	D0:5B:A8:07:73:6C	Wired/AP	2437	47	4K	Unknown
	E8:4E:84:86:4A:21	Wireless	----	35	1K	SamsungE
	64:6C:B2:ED:E6:A6	Wireless	----	11	1K	Unknown
!	48:74:6E:AB:34:91	Wireless	2437	178	22K	Unknown
	D8:BB:2C:B9:53:EC	Wireless	----	35	8K	Unknown

- In the third section, there is a graph that shows the current rate at which the packets are traveling around and the total amount of data packets that are shown with red bars.
- In the fourth section, we can see a lot of details that are listed, such as the hostname (Kali), total number of networks my NIC is able to see, for how long Kismet is running, the total number of packets captured, and an average rate of packets seen per second. Using such simple tools without any special configuration, we were able to collect a good amount of specific details.

In the bottom-right corner of the window, the interface used to capture details is shown: `mon0` (a monitor mode activated interface). Through this tool, we are not able to capture any RF energy that can distort the traffic shape, which lessens our network performance. But the same tool, when paired with Wi-Spy or Ubertooth hardware, will show the RF energy spectrum. If you are one of those professionals who needs to deal with Wi-Fi troubleshooting in day-to-day working, then you should use this—if not now, then someday you will.

The RF energy emitted from the devices won't be the problem every time; sometimes, you would be required to look at the packet level like checking authentication and association packets, that is, you can match your normal traffic pattern with the anomaly you might be facing.

The IEEE 802.11 packet structure

The medium used by the packets to travel from one host to another is changed for now, but the basic protocols that work on the preceding layers are still the same. As we already discussed, layer 2 (data link) is of great importance here. Understanding packets traveling in detail is obviously a good thing; we will discuss various types of frames, header structures, and information an 802.11 packet contains.

There are basically three types of frames that you will see while analyzing wireless packets. All the packets listed are almost similar to the one we saw earlier; the only difference here is the extra information that is appended because of the 802.11 header. The following are the header types that you will see:

- **Management**: To form a connection between the hosts at the data link layer, these frames are used. These frames are used to join or leave a network, associations/disassociation/reassociation and to broadcast beacon packets and a few administrative tasks. Management frames are responsible for a lot of activities that take place while the connection between the hosts is established.
    - **The beacon frame**: The AP sends beacon frames every 10th of a second to let the STA know that the AP is available for connection.
    - **The authentication frame**: This type of frame is sent by the STA to the AP containing its identity. If the AP follows an open system authentication, then STA would send just one authentication frame that AP acknowledges to understand whether the connection is accepted or rejected. If the AP follows shared key authentication, then the STA sends a request to the AP to get connected. Now, AP sends a challenge text to the STA. After this, STA completes the challenge and encrypts the challenge text requested using the same algorithm that the AP is using, and then it sends it to the AP. AP receives and decrypts the text using it's own key value, and no matter what the result is, it determines the status of the connection request.
    - **The association request frame**: This frame is sent from the STA to the AP to provide details of the allocation of resources and for syncing purpose.
    - **The associate response frame**: This frame is sent in response to the

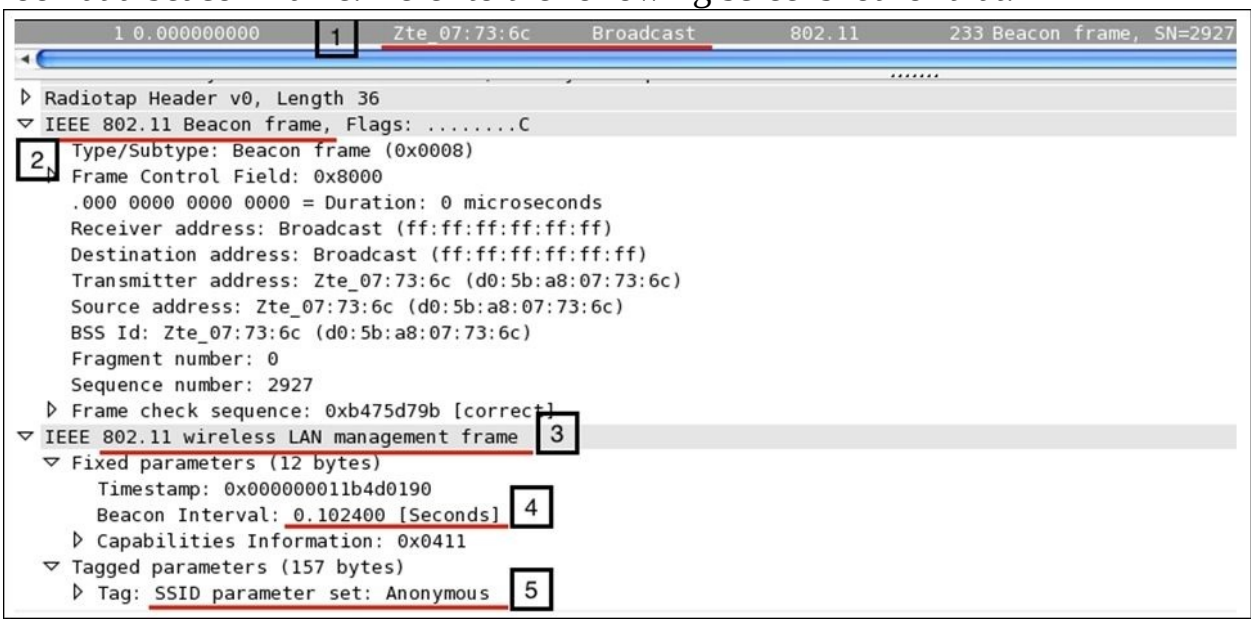
AP for the STA request that is sent.
    - **The deauthentication frame**: This is sent by the STA to terminate the connection with the AP/STA.
    - **The disassociation frame**: This frame is a graceful way of terminating the connection so that the AP can free up the resources allocated for the STA.
    - **The probe request frame**: This frame is sent by the STA to another STA/AP to request for its details; this is basically used to find nearby APs.
    - **The probe response frame**: This frame is sent in response to the request that AP/STS might have received from another device in the network.
    - **The reassociation (request/response) frame**: This frame is sent to the new AP when an STA's association with the current AP gets dropped. In response, the AP acknowledges the acceptance/rejection for the reassociation request.

  Monitoring the time gap between each beacon frame sent from the hosts can be useful when dealing with high latencies. Due to these beacon packets broadcasted from the AP, the devices know that they are available to connect to.
- **Control**: This is to ensure that the delivery of the packets between the hosts manages the level of congestion in your channel and uses packets such as clear-to-send and request-to-send. In short, we can say that these frames are used for maintenance tasks. These control packets ensure the integrity of the packets that are transmitted. Likewise, the management frame several kinds control frame has just three kinds:
    - **Request-to-send (RTS)**: This frame is sent by the STA to request for gaining the control of the medium for a particular duration.
    - **Clear-to-send (CTS)**: This frame is sent by the AP from where it received the RTS to specify when the medium will be allocated to the STA for transmission. This frame is often used for protection from older stations that want to gain access to the medium again.
    - **Acknowledgement (ACK)**: This frame is sent by the receiving STA to tell the sending station that the data packet was received successfully. If the sending station does not receive this packet, then after a definite period of time, the sending station will resend the data packet to the

same recipient to ensure the delivery of the packet.

- **Data**: These frames contain the data that is actually sent between the hosts. These are the only frames that get transmitted between the wireless and the wired domain.

The 802.11 packets are similar to the wired network packets that we saw; the terminologies do differ a little bit, but the basic concept is identical. Let's take a look at a beacon frame. Refer to the following screenshot for that:

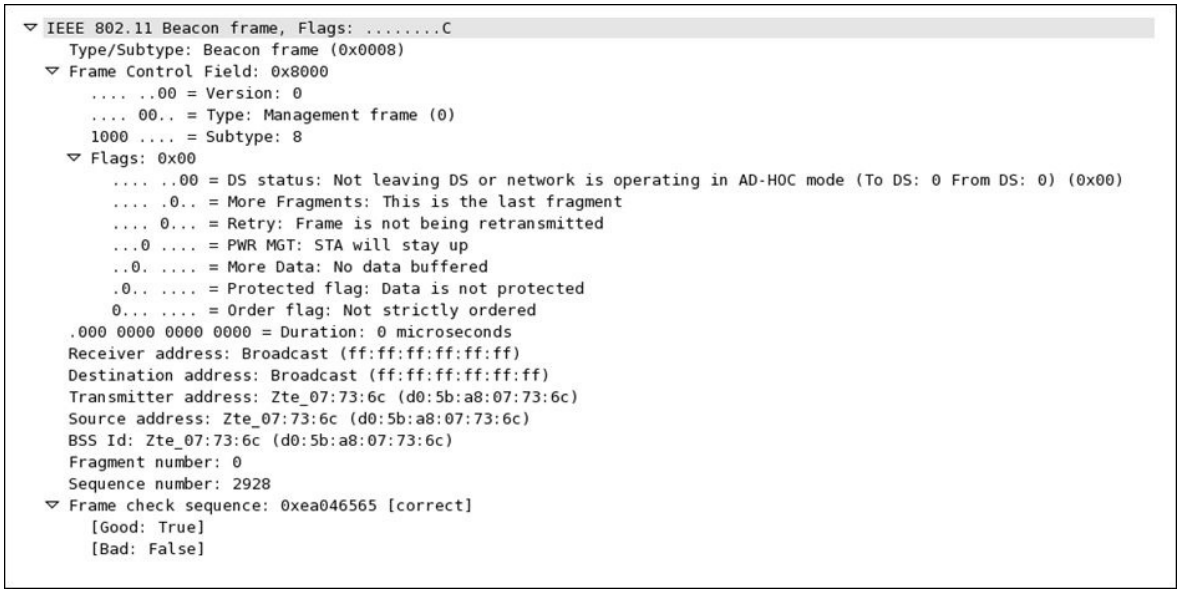


Now, let's see what all the pointers in the preceding figure signify:

- **1**: The packet describes it all; the beacon frame is sent to the broadcast address from the Wi-Fi-enabled device or any device that is currently listening can connect to it using the right credentials.
- **2** and **3**: Here, the type of the frame is management and the subtype is beacon.
- **4**: As we discussed earlier, beacon frames are transmitted every 10 seconds. You can verify the same from the packet itself, to be precise; the next beacon frame was sent after an average time of 0.102385000 seconds (this is just the time gap I calculated between the two packets seen in the list pane).
- **5**: The SSID broadcast is enabled, and hence, the packet is shown with the

broadcasted SSID **Anonymous**, which will be visible when you try to scan nearby Wi-Fi hotspots that you wish to connect to (you need to use the monitor mode to capture this packet). Various other details are included in the beacon frame that is part of the header and is quite necessary to know about. Refer to the following frame structure that shows how a layer 2 datagram looks like in theory and in Wireshark:

Frame Control	Duration/ID	Address1	Address2	Address3	Sequence Control	Address4



Let's take a look at the fields present in the frame in detail: This is the first section in the frame header that lists out quite a good amount of info in it.

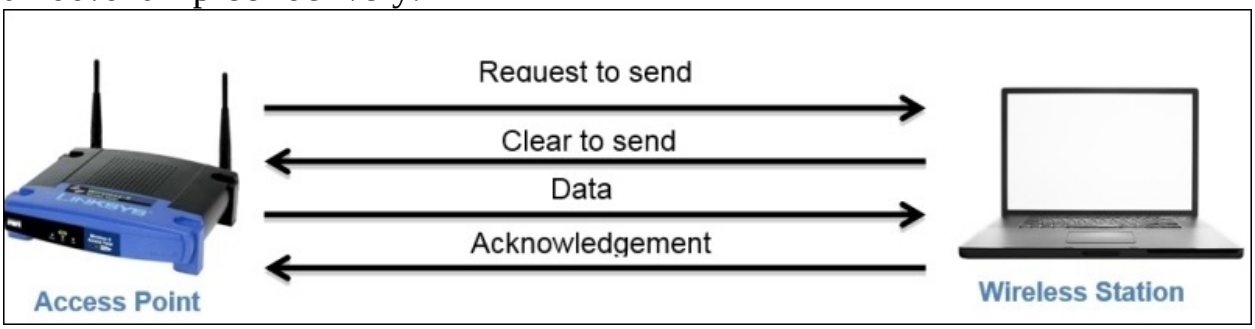
- **Frame Control**
- `Protocol Version`: This represents a 2-bit value that is used to verify the version of the protocol in use; the current version is 0 at the time of writing.
- `Type`: This identifies the type of the frame; in our case, we are dealing with a management frame (beacon).
- `Subtype`: This represents the subtype of the header; for us, it is a beacon frame for which we are seeing a numerical code `8`.
- `DS Status`: This represents whether a data frame is heading to a **distribution system** (**DS**) or working in which mode. If the bit is set to `1`, then this must be a data frame; if this is set to `0`, then this frame is probably

a management/control frame.

- `More Fragments`: If this bit is set to `1`, this means that the frame has been distributed into couple of parts and is being sent one by one.
- `Retry`: This bit is set to `1` when there is a requirement upon retransmission of the frame.
- `PWR Management`: If this is set to `1`, it represents the current power management state of the STA whether it is `active:0` or in the `power-save:1` mode.
- `More Data`: This bit is set to `1` if the AP is trying to tell the STA in the power-save mode that it has more frames to send. In case of control frames, this will always be `0`.
- `Order`: If this bit is set to `1`, this means that the frame is forcefully lined up and would be sent in a sequence. Usually, this bit is not set because it might cost transmission performance.
- `Duration ID`: This denotes the time the sender might require for frame exchange; this is usually seen in an **request-to-send** (**RTS**) frame, which requests to occupy the medium for a certain amount of time.
- `Address 1/2/3`: This is the physical address of the communicating device (receiver, transmitter, and destination address).
- `Sequence Control`: This is composed of two subfields: a 12-bit sequence number and a fragment number of 4 bit. A sequence number field is used to identify the sequence of the frames that arrive and for their proper reassembly (this ranges between 0-4,095). The fragment number field is used to denote the number of fragments for each frame (this ranges between 0-15).
- `Address 4`: This represents the sender's physical address and would only be present in a wireless distribution mode.
- `Data/Payload`: This field is not part of the header, but at the end, it will be appended when data is being sent across. The size of this field can be up to 2,324 bytes.
- `FCS`: The frame check sequence field is used to perform a data integrity test; you must have heard about the **cyclic redundancy check** (**CRC**), which helps in calculating a value related to the data we received. If the FCS value is identical to the one we calculated, then the packet is received without errors.

RTS/CTS

These are one of those essential components of WLAN data transfers that avoid collisions from happening and ensure the integrity of the data that is sent. The following illustration determines the four-step process that takes place to follow a 100% fail-proof delivery:



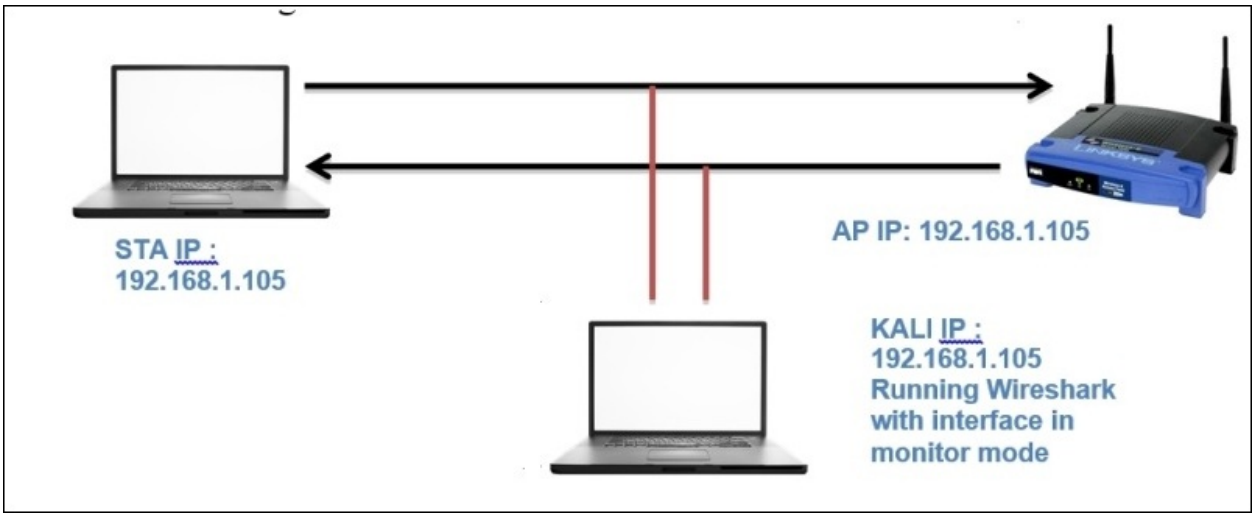
First, the AP sends a request to the STA to gain medium access; once the STA approves the AP's request, the AP starts sending data. As soon as the data transfer is completed, the STA sends an ACK packet to acknowledge error-free delivery. If the ACK is not sent, then then the AP will start retransmission after some time.

Usual and unusual WEP – open/shared key communication

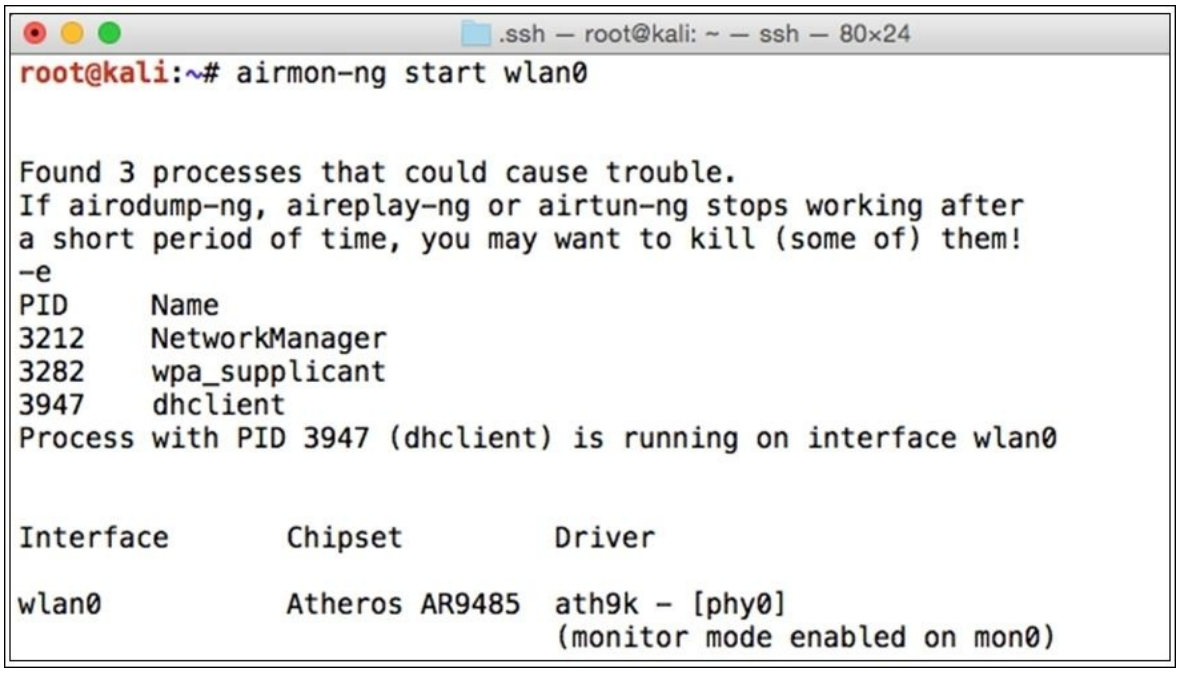
Here, we will discuss two types of **Wired Equivalent Privacy** (**WEP**) authentication procedures: open and shared keys. As a matter of fact, discussing WEP is really unnecessary, but we should be aware of how it works because you never know when you might be asked to troubleshoot an old router whose firmware is still not upgraded and just supports WEP as an authentication mechanism.

WEP-open is way better than WEP-shared because even when the password that you provide turns out to be wrong, you will get connected to the network; here, it reduces the chance of getting the router brute forced. If you are using WEP-shared communication, then an experienced hacker won't take more than 2 minutes to crack your strongest key, and because of the small pool of keys that WEP supports, your password won't last long.

So, to begin with, we need the infrastructure to capture packets that are required for WEP-open. A key point to note here is that the infrastructure I am using consists of three different machines: the access point on the `192.168.1.1` IP, the station on the `192.168.1.105` IP, and Kali Linux running Wireshark on the `192.168.1.104` IP. Refer to the following illustration to understand this:

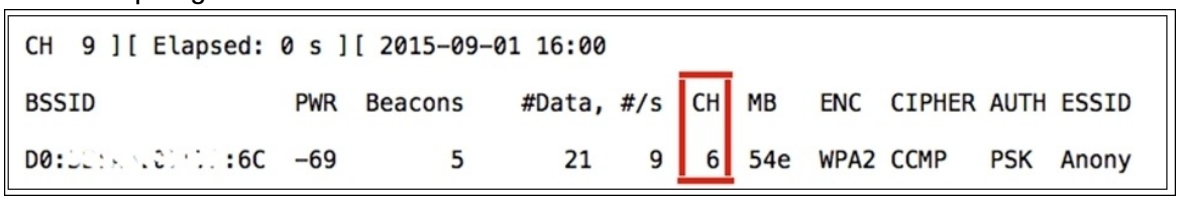


1. First, let's activate the monitor mode over my interface:



In the bottom-right corner of the preceding screenshot, you can see the message that the monitor mode is enabled over the `mon0` interface. This is the same interface that we will use to capture `802.11` packets from our AP and STA.

2. Next, to confirm the channel over which my channel is working, I used the `airodump-ng mon0` command.



3. Now, once we have figured out that the channel is `6`, we can go ahead and make our interface listen specifically to this channel, thus avoiding any noise from other channels. To do so, I used the `iwconfig mon0 channel 6` command.

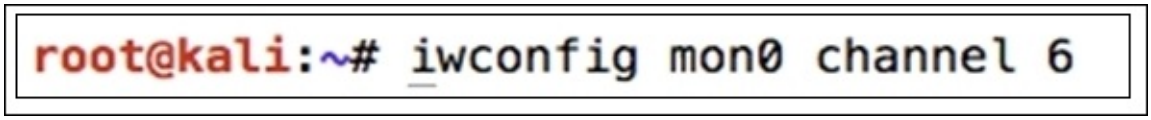


Figure 1: Configuring mon0 interface to channel 6

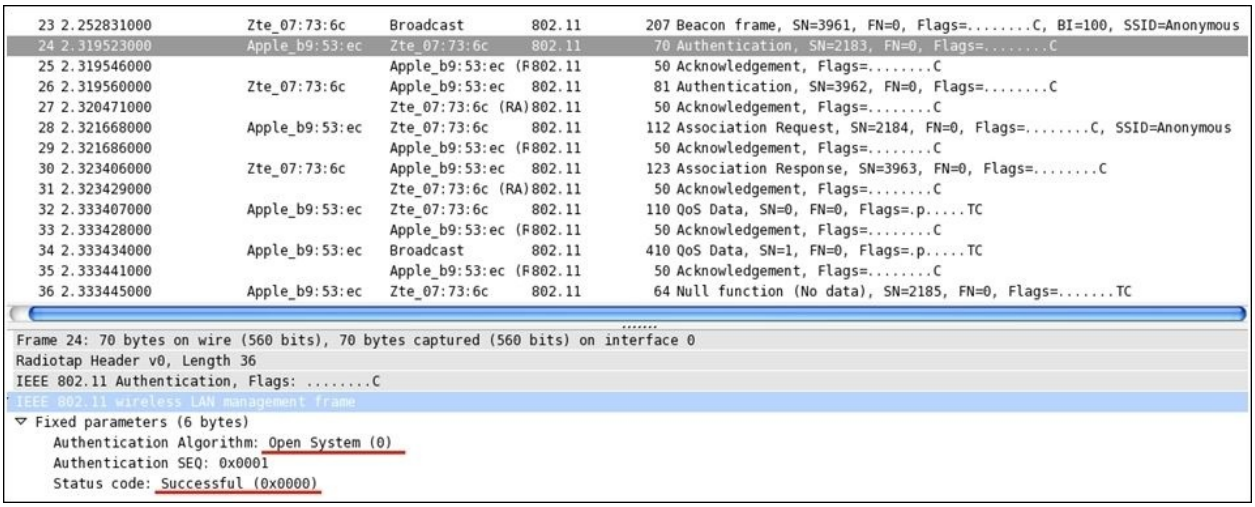
4. Once you have completed all these steps, go ahead and launch Wireshark. If the output of the commands you issued gives any error, then please rectify it before you proceed.

WEP-open key

Once the interface starts working fine and you are able to see the beacon frames broadcasted from your access point and probe request or response to and from your station, then you can simply launch a WEP-open authentication session. When asked for a password, just give any random password which will let you get connected to the network, but it might be possible that you won't be able to access the Internet connection shared by the AP with other STAs. Refer to the following screenshot depicting a WEP-open authentication session.

To capture the normal traffic pattern, I will use a Linux distribution (Kali) running on an independent machine that has a feature to activate the monitor mode (without the monitor mode, you can not capture 802.11 packets.) First, activate the monitor mode on our WLAN adapter using a basic set of commands, and we will also configure the same adapter to listen to a specific channel.

After launching Wireshark, make sure that you choose the `mon0` interface only; then, you will be able to capture relevant traffic (keep the promiscuous mode on as well).



As clearly visible in the details pane of the first authentication frame selected in the list pane, the authentication system is `Open-System (numeric code 0)` and the connection attempt is successful as well. Following this, we can see an

association request/response and then some QOS and Null function data frames.

An **association request/response** is sent and received by the STA/AP to associate a dropped connection, which the client was already a part of before, and to allocate the resources STA might require for communication over the channel.

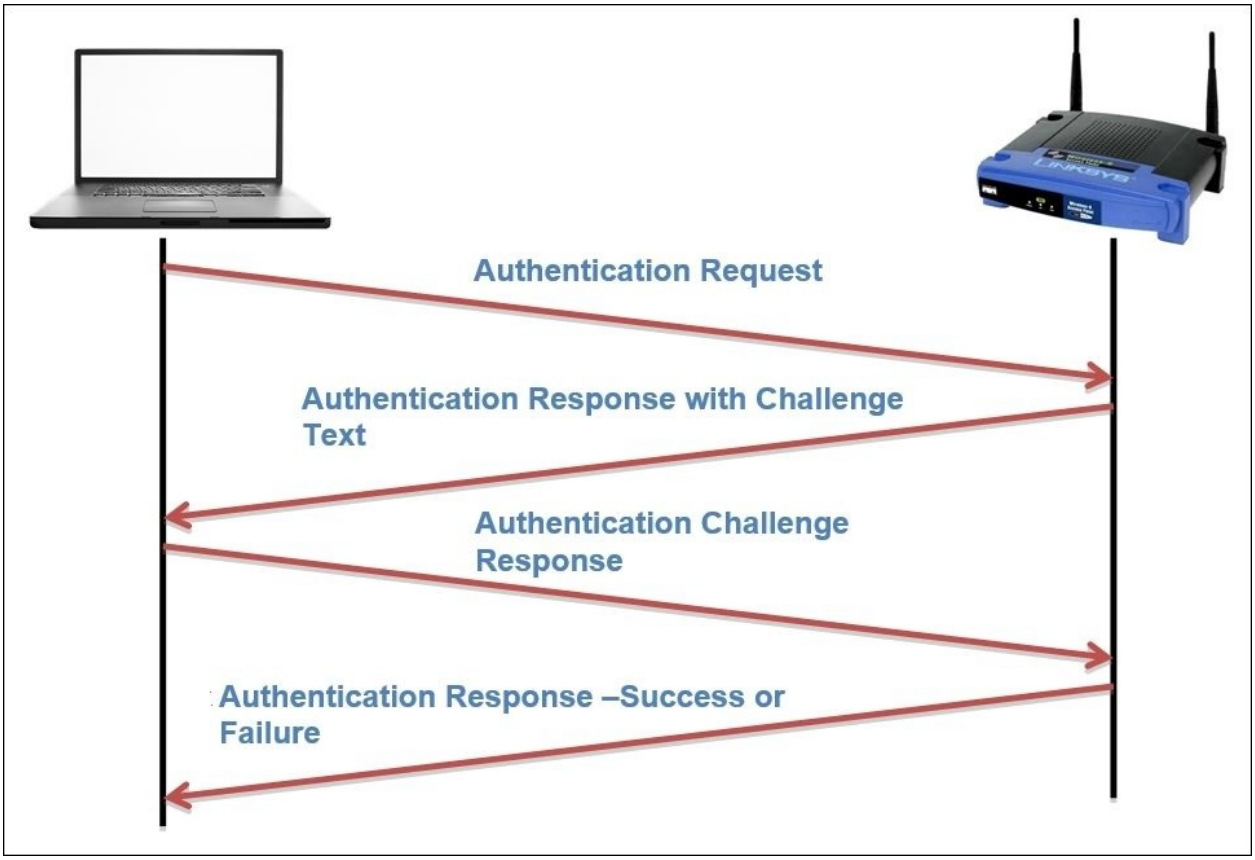
A **QOS data** packet is a subtype of the control frame types, which depicts the quality of service and the over all performance.

Null Function packets are used to inform AP that the STA is going in the power-save mode. This packet does not carry any data, just some flag information.

And for every kind of information being shared between hosts, there are **ACK** packets that are sent across to determine the delivery of every packet in the communication.

The shared key

Before we start configuring, I want you to understand the process of WEP-shared key authentication, that is, the steps involved in the whole session. Refer to the following illustration to understand this:

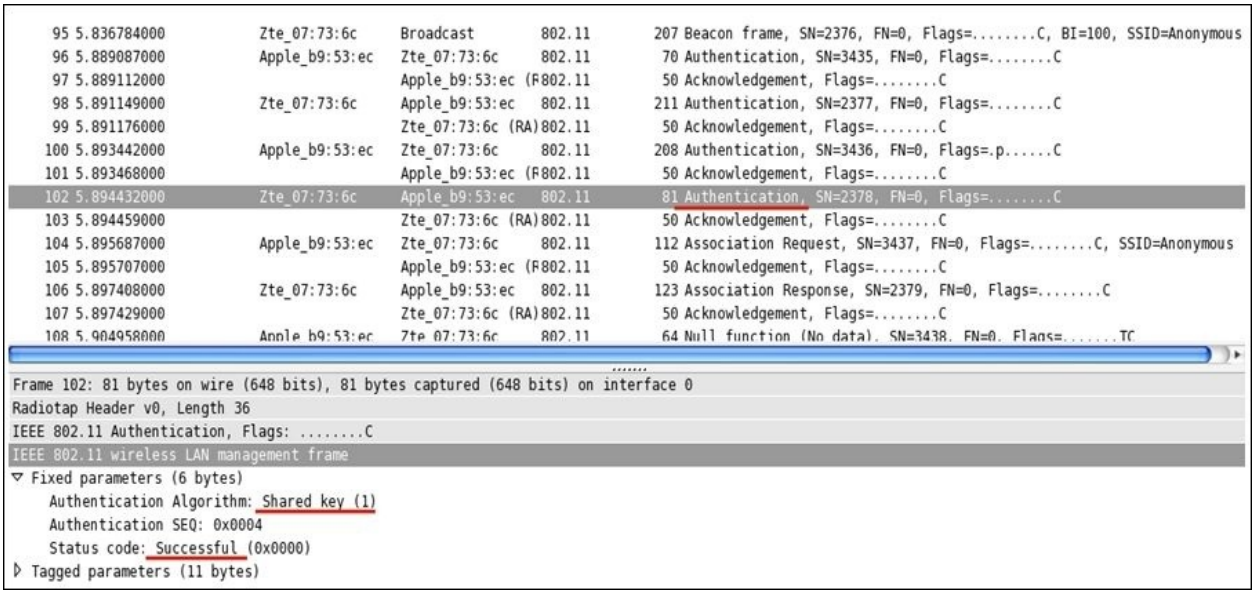


In short, the STA tries to connect to the AP by sending an authentication request, which the AP acknowledges by sending a text challenge that the STA is supposed to complete and before sending an encrypt using the key algorithm AP knows about. Once STA has completed the challenge process over his end, STA sends the challenge response which is being evaluated by the AP and determines the success or failure of the connection and the same is acknowledged to the STA in another authentication frame.

So, for a normal WEP authentication session, you will observe at least four authentication frames. If the authentication is successful, then the authentication

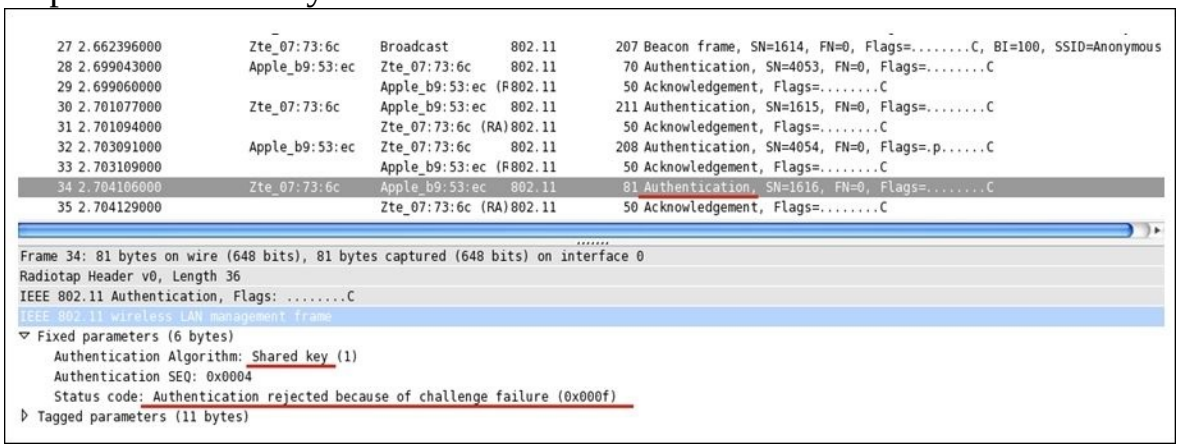
frames will be followed by an association request/response along with some data transfer. And if the authentication is not successful, then after four authentication frames, the session between the STA and the AP will end. Follow the next steps to capture WEP management, control and data frames from your WLAN.

As discussed, you will note that the same pattern of packets is captured. Refer to the following screenshot depicting a successful WEP authentication session that was captured by Wireshark:



- For the fourth authentication frame, I have expanded the details section to confirm whether the connection attempt was successful or not. And from the preceding screenshot, we can verify that it was successful. The authentication type used for the communication can also be seen here.
- As we know, now if the connection attempt between the STA and AP fails, the whole session will be terminated after the fourth authentication frame and we will see a failure message. To verify the same, I tried duplicating the scenario while Wireshark was listening through an interface in the monitor mode on an individual system.
- Refer to the following figure that illustrates a failed WEP connection attempt. In the list pane, we can see the same authentication frame pattern (just four authentication frames), but the last frame that the STA received from the AP acknowledges the connection status. As is clearly visible in the

details pane, the connection attempt failed due to an incorrect challenge response text sent by the STA.

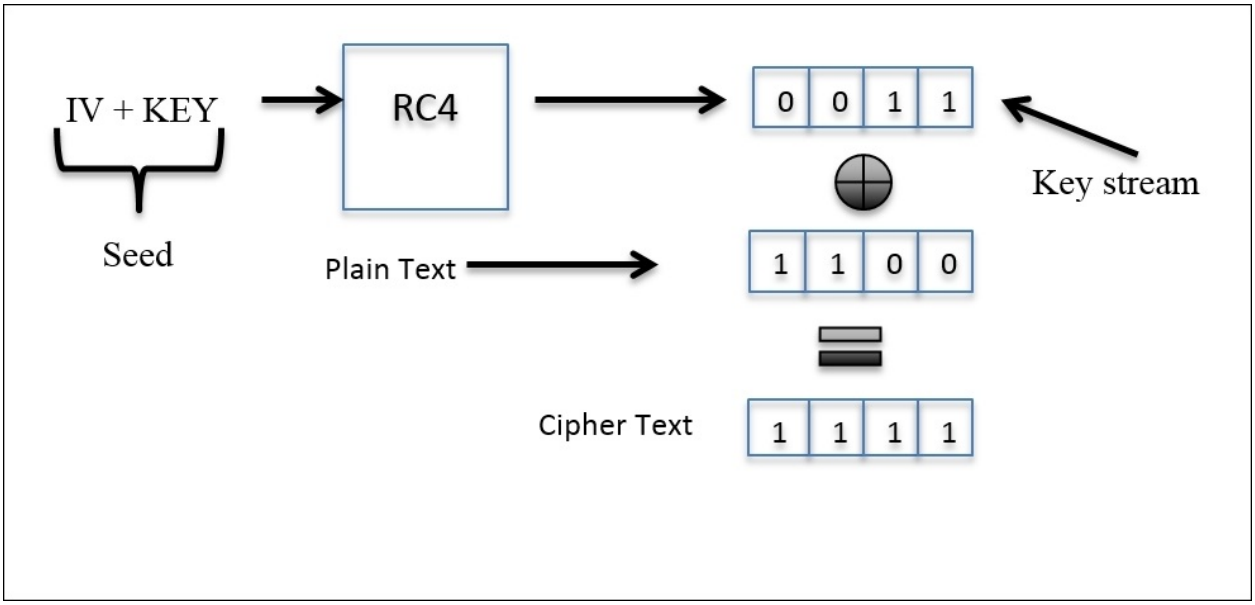


We witnessed two types of authentication procedures that WEP supports, but what is really important to know is that WEP is now obsolete, so I would never recommend to any of you to use this as an authentication protocol. If you have any old devices that only support WEP, then kindly upgrade to the latest hardware.

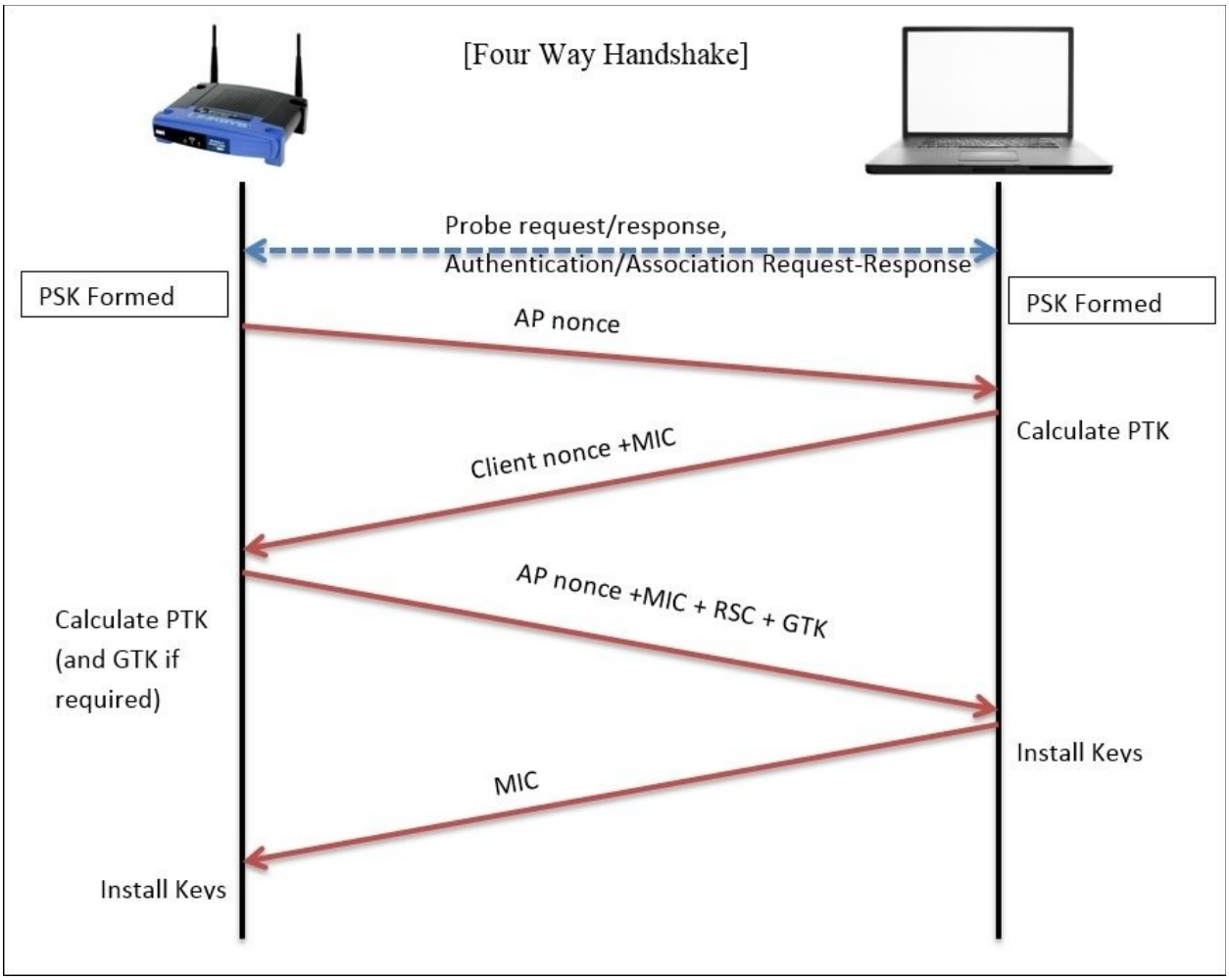
WPA-Personal

We talked about a crappy authentication algorithm that has been used since the birth of wireless networking, but when we have a better option, why not use it. I am talking about the **Wi-Fi Protected Access** (**WPA**) security algorithm that is stronger than WEP when we add the corrective measures required. In 2003 when WPA was launched by Wi-Fi Alliance as a measure to make WLAN communication stronger than the previous protocol, WEP. Nowadays, almost every WNIC supports WPA authentication mechanism, thus enabling you to take advantage of using a better security protocol. The **Temporal Key Integrity Protocol** (**TKIP**) lets the existing legacy hardware upgrade easily to implement WPA. The key size used by WEP was 40/104 bits, whereas WPA uses a key size of 256 bits, and the interesting thing to know is that every packet transmitted between the AP and STA is encrypted using the 256-bit key, which makes the situation quite tight for malicious users. One more advance was done in WPA that let the devices communicate with more assurance about the integrity of the message.

In WEP, the traditional CRC was implemented, but here, the popular Michael 64-bit **Message Integrity Check** (**MIC**) was introduced to address the issue. WPA also uses the RC4 algorithm to build a session based on dynamic encryption keys (you would never end up using the same key pair between two hosts). If compared to WEP, it has a larger IV size of 48 bits. Refer to the following illustration of how the cipher text is formed that is transmitted over the medium:



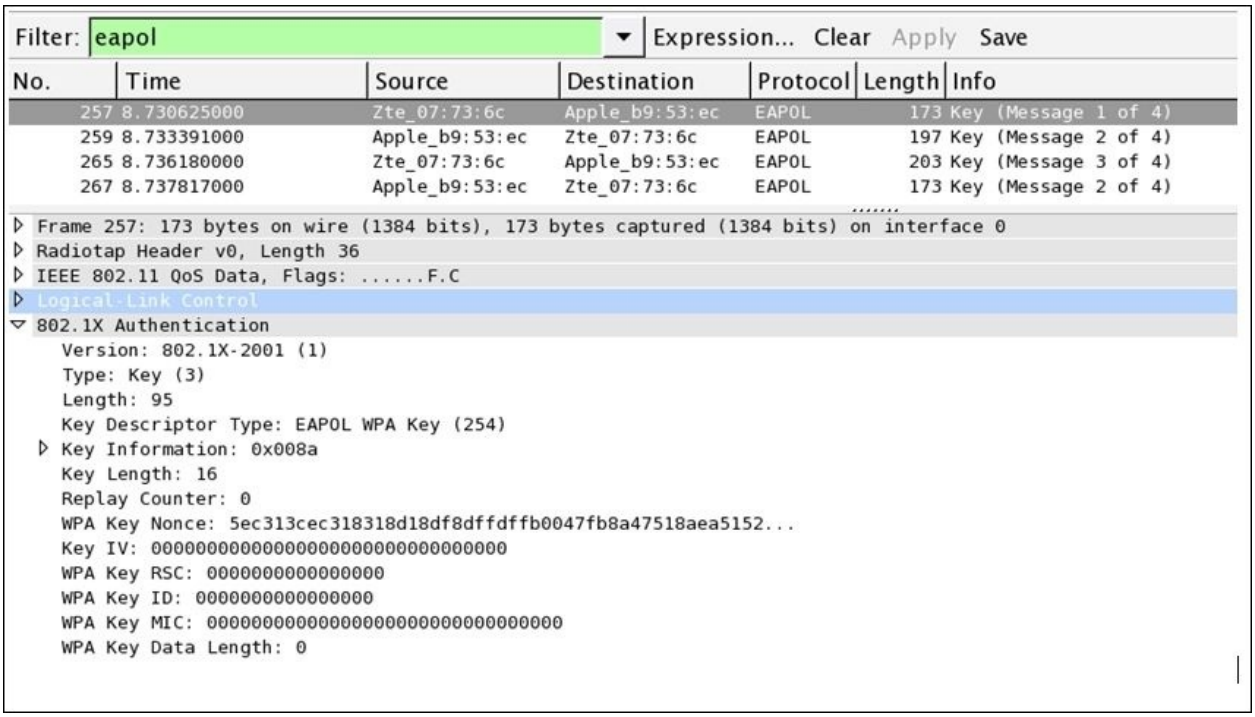
The preceding illustration depicts how the whole process starts by appending the IV and the dynamically generated 256-bit key. Then, is passed on to the RC4 algorithm, which encrypts the packets with keys, and then the resulting encrypted key stream is appended with the data and voila! We have the cipher text. Now, I will introduce you to the normal authentication session between an AP and an STA. Refer to the following figure for the same:



In the case of the Enterprise WPA configuration, first, the *Master Key Exchange* takes place. I will later give you a brief about it. As of now, we have an AP that sends its nonce (random value) to the STA (initiation of connection) that will use the AP's nonce value and its own nonce to calculate the **Pairwise Transient Key** (**PTK**) along with the **Pre Shared Key** (**PSK**), which was established during the initial connection process. The resulting value will be sent to the AP. Then, the AP will calculate the PTK over its end and append the MIC with the **receive sequence counter** (**RSC**) that helps in identifying the replayed messages. The resulting value will be passed on to the STA. Now, the STA will first verify the MIC in the message to ensure the integrity and install the keys. Then, a response will be sent to the AP regarding the status. If the status shows success, the AP then installs the same keys (dynamic keys) that will be used in further

communication between the hosts.

After configuring WPA-Personal on my AP, I had sent an authentication request from my client and the corresponding communication was captured by Wireshark, which is shown in the following screenshot:



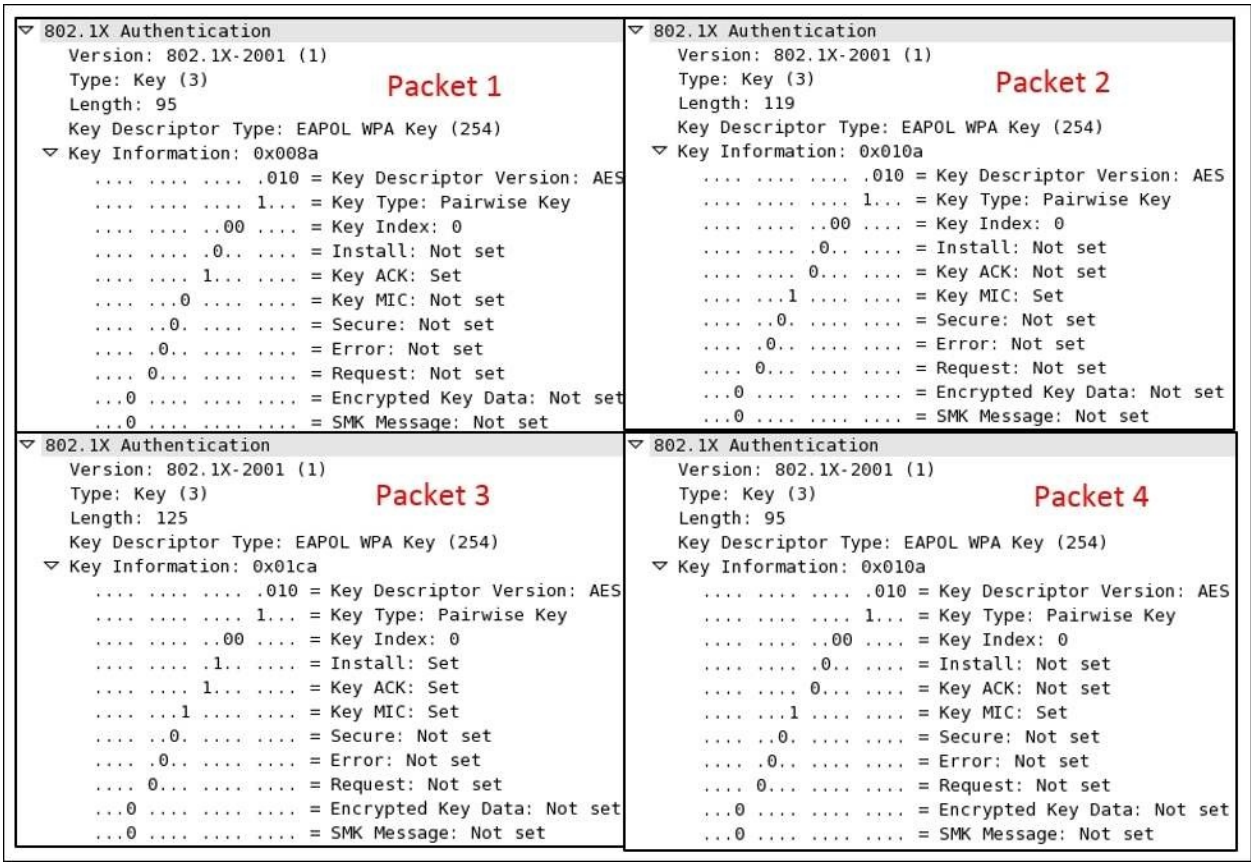
Note

You need the same infrastructure that we used while capturing WEP communication that is an interface in the monitor mode that is listening on a separate machine.

This is what a normal WPA successful handshake (authentication) process looks like, that is, four EAPOL packets. To analyze the session specifically between the AP and STA, I applied a display filter to see only EAPOL packets (authentication frames). Before the authentication frames, AP's beacon frame, and STA's probe, we looked at authentication and association request/response packets that led to the authentication session, following which PSK was used to generate the dynamic keys. Because of a software package error that I installed on my machine, the fourth packet says `Message 2 of 4`, whereas it should be

`Message 4 of 4.`

Getting into more detail, I would like to show you the flags marked in all of these four authentication packets that will definitely clear your thoughts regarding the WPA handshake process. Refer to the following screenshot that illustrates this:



Here is the description of the preceding authentication packets:

- **Packet 1**: The pairwise master key (pre-shared key) and the ACK bit are set (probably because of the association request/response exchanged earlier), which was sent by the AP to STA to initiate the connection along with the nonce value that was chosen randomly.
- **Packet 2**: The pairwise master key and the MIC flag is set, which STA sent to the AP to for acknowledging the request received, along with its own nonce value appended to the AP's nonce and the MIC for integrity check.

- **Packet 3**: The pairwise master key, `install`, key ACK, and MIC flags are set, which the AP tries to send to the STA. The STA will fulfill the challenge text values received and will confirm to the AP along with the encrypted challenge text which AP is going to be crosschecked.
- **Packet 4**: Here, the pairwise master key and the MIC flag are set, which the STA sends to the AP to make the connection complete. Now, the AP is mutually ready to perform data transfer with the STA.

I hope these flags help you understand the four-way handshake process in an easy and realistic manner.

Next, we are going to see what happens when the AP receives an incorrect challenge text from the STA, what the packets look like in the list pane, and whether there would there be any difference in the pattern of packets that are captured.

The STA will try to connect to the AP and the AP will request the challenge text. The STA this time is not aware of the secret keys used by other clients in the network, so ending with an incorrect pass key which won't be accepted by the AP, or please check acknowledged by the STA. The STA will try again to send the challenge text and the same process goes on. After this, you will notice a couple of similar packets in the list pane. Refer to the following figure for the same:

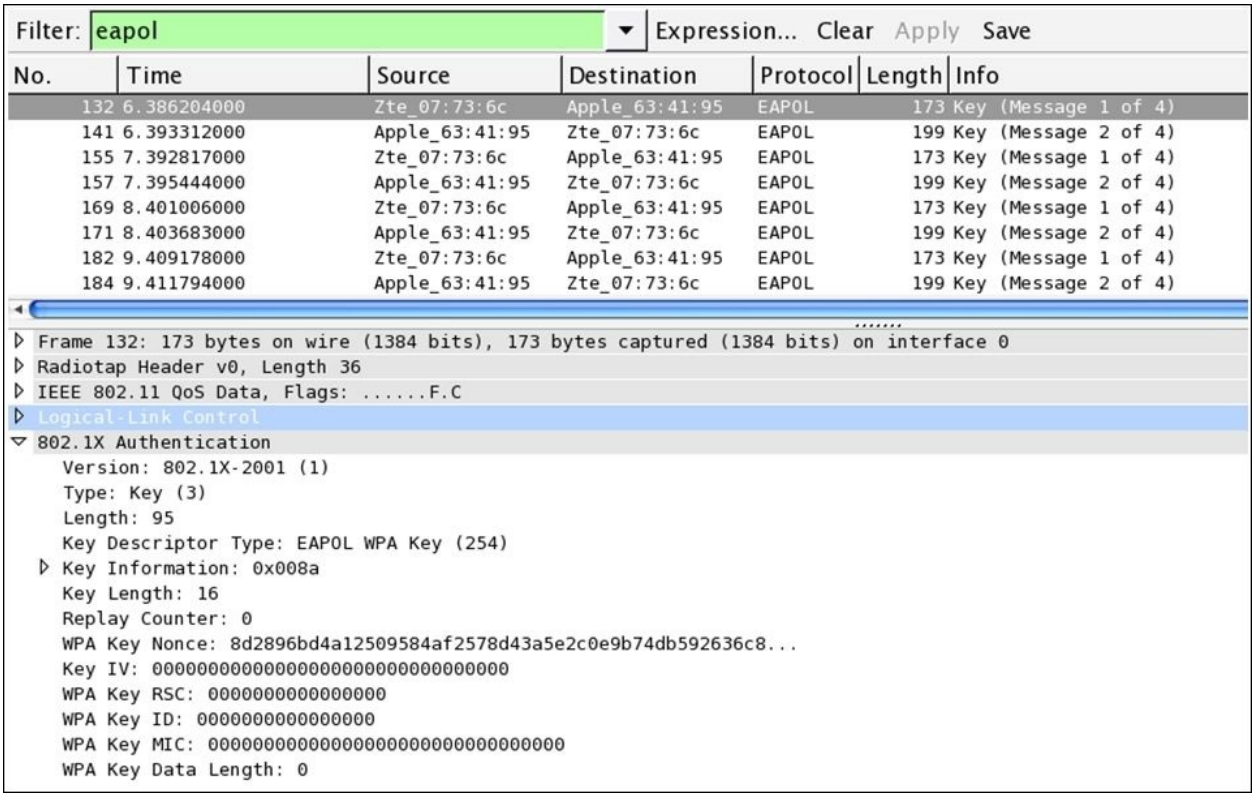
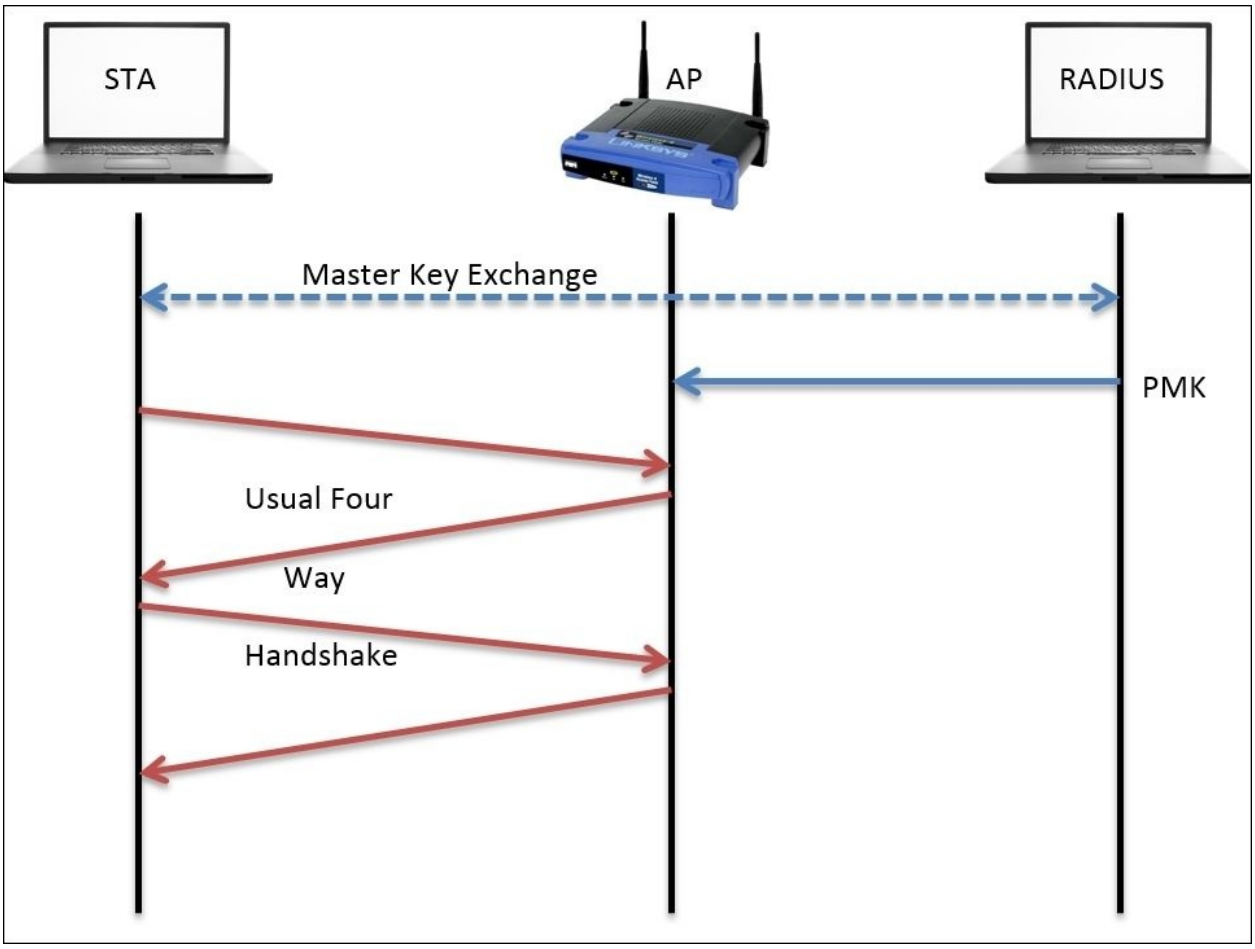


Figure 2: WPA Failed authentication

As you can see in the preceding screenshot, **EAPOL Message 1 and 2** can only be seen because when the STA provides the challenge text response, the AP rejects it and again the process starts from beginning. The same thing will continue for a couple of times, but a packets pattern of such kind denotes unsuccessful connection attempts (may be a brute force attack). The packets listed can be associated with each other using the replay counter listed that we saw earlier in the key nonce in details section.

WPA-Enterprise

I promised we would be discussing the enterprise mode in brief, so here it is. In the corporate infrastructure, the key and passwords are not kept with the AP, and even the AP is not responsible for authentication with the **STA**. There is an extra entity, the **RADIUS** server, that takes care of authentication here. Before the four-way handshake takes place, the **RADIUS** server and the access point are supposed to go through a **Master Key Exchange**, which gives an assurance to both the communicating devices that the other part is legitimate. Let's have a look at the following figure:



Afterwards, the pairwise master key is created and passed on to the AP, which will lead on and complete the four-way handshake process and complete the

authentication session.

I've scrolled down the packet list and look what I found for you: **Disassociation** and **Deauthentication** packets in action captured by our sniffer. So, before we wrap up, you should take a look at them.

The wireless stations/access points use disassociation packets in order to notify the access point that the client is now going offline and the resources that have been allocated by the AP to wireless clients can now be released. Refer to the following figure that illustrates the same:

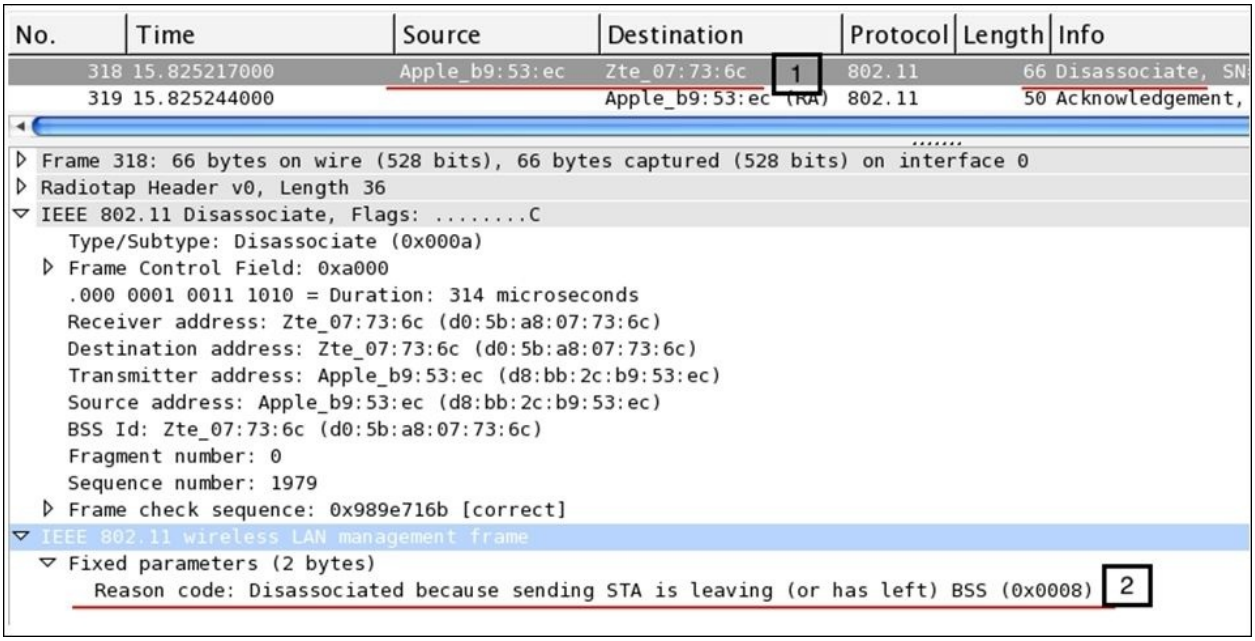


Figure 3: The disassociation packet

As you can observe, at first, the STA sends a `disassociation` frame and receives `ACK` (`318`,`319`) for the same. Now, for better understanding of the packets, we can take a look at the details pane (select the disassociation packet first), where the `Reason Code` parameter states that the STA is leaving or has already left. This gives us a feature through which we can view and understand packet behavior efficiently.

The wireless stations or the access points use the `deauthentication` frames to

notify the other side of the communication that the other device is leaving. There can be several reasons for it. Refer to the following figure to understand this:

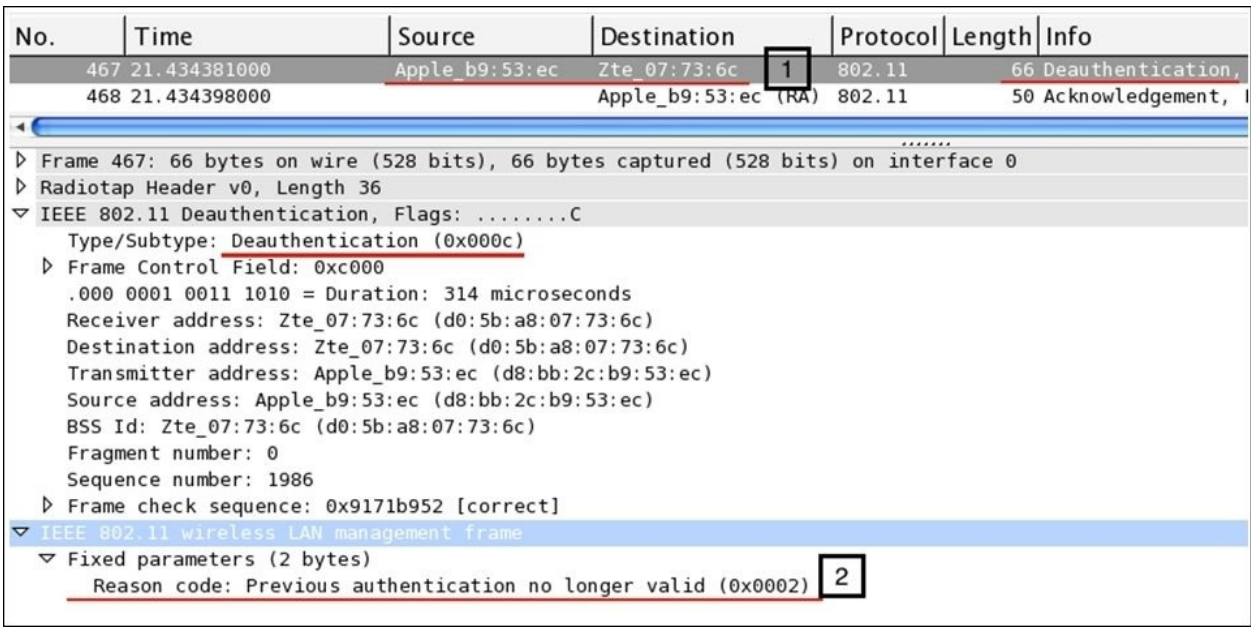


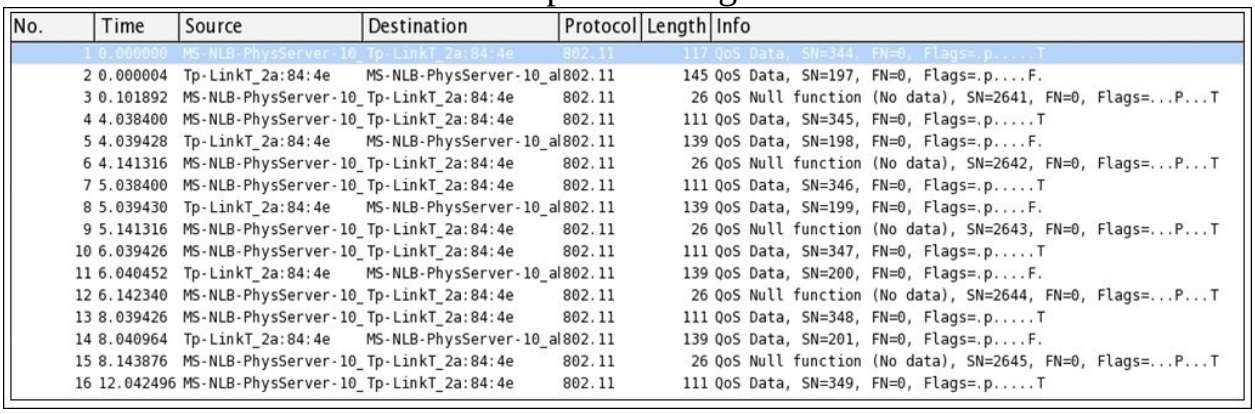
Figure 4: The deauthentication packet

First, the STA sends a `deauthentication` frame to the access point, which gets acknowledged in the next packets (`467`,`468`). After expanding the details section for the deauthentication packet, we can easily note that the `Type/Subtype` field is verifying the same. And at the bottom, we get to understand why the deauthentication packet was generated. In our case, it is `Previous authentication no longer valid`, which the STA tried to notify the AP about, and if they wish to communicate again in the future, then the process of authentication has to start over, from the probe and association frame, following the four-way handshake.

Decrypting WEP and WPA traffic

The technique to decrypt WEP and WPA traffic is available with the use of Wireshark. As we know, WEP is the weakest security encryption protocol and it has been exploited for a long time. Once you have the key for the wireless network, it becomes a matter of a few clicks to decrypt the traffic.

To demonstrate the same, I have sanitized the wireless traffic between my access point and a client that is connected to it. Refer to the following screenshot where the normal IEEE802.11 traffic is captured using Wireshark:

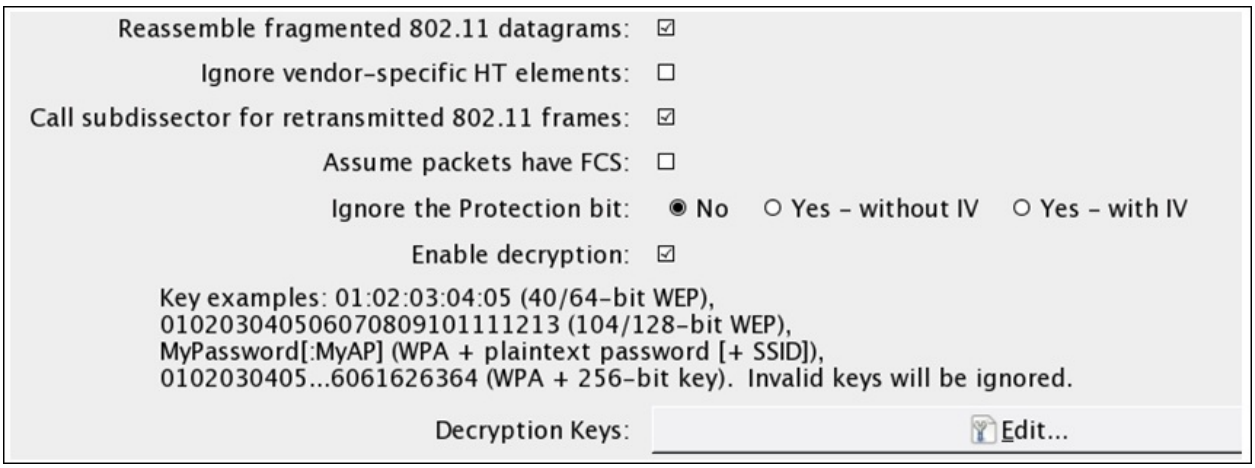


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	117	QoS Data, SN=344, FN=0, Flags=.p....T
2	0.000004	Tp-LinkT_2a:84:4e	MS-NLB-PhysServer-10_a	802.11	145	QoS Data, SN=197, FN=0, Flags=.p....F.
3	0.101892	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2641, FN=0, Flags=...P...T
4	4.038400	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	111	QoS Data, SN=345, FN=0, Flags=.p.....T
5	4.039428	Tp-LinkT_2a:84:4e	MS-NLB-PhysServer-10_a	802.11	139	QoS Data, SN=198, FN=0, Flags=.p....F.
6	4.141316	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2642, FN=0, Flags=...P...T
7	5.038400	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	111	QoS Data, SN=346, FN=0, Flags=.p.....T
8	5.039430	Tp-LinkT_2a:84:4e	MS-NLB-PhysServer-10_a	802.11	139	QoS Data, SN=199, FN=0, Flags=.p....F.
9	5.141316	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2643, FN=0, Flags=...P...T
10	6.039426	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	111	QoS Data, SN=347, FN=0, Flags=.p.....T
11	6.040452	Tp-LinkT_2a:84:4e	MS-NLB-PhysServer-10_a	802.11	139	QoS Data, SN=200, FN=0, Flags=.p....F.
12	6.142340	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2644, FN=0, Flags=...P...T
13	8.039426	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	111	QoS Data, SN=348, FN=0, Flags=.p.....T
14	8.040964	Tp-LinkT_2a:84:4e	MS-NLB-PhysServer-10_a	802.11	139	QoS Data, SN=201, FN=0, Flags=.p....F.
15	8.143876	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2645, FN=0, Flags=...P...T
16	12.042496	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	111	QoS Data, SN=349, FN=0, Flags=.p.....T

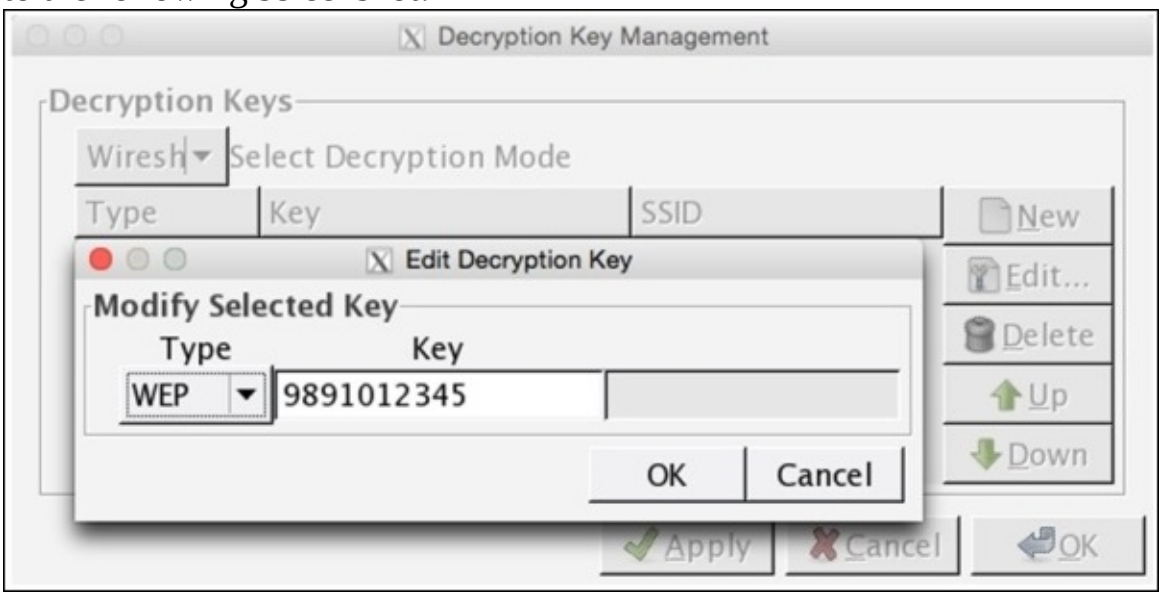
Figure 5: WLAN traffic before decryption

I hope that by now you must be aware of the kind of packets that we see in the list pane, but still, it does not make much sense in terms of network-activity-related traffic. This is why you need to learn the technique to make the entire traffic more readable. Before you proceed, you need to make some changes in the preferences section of the IEEE 802.11 protocol.

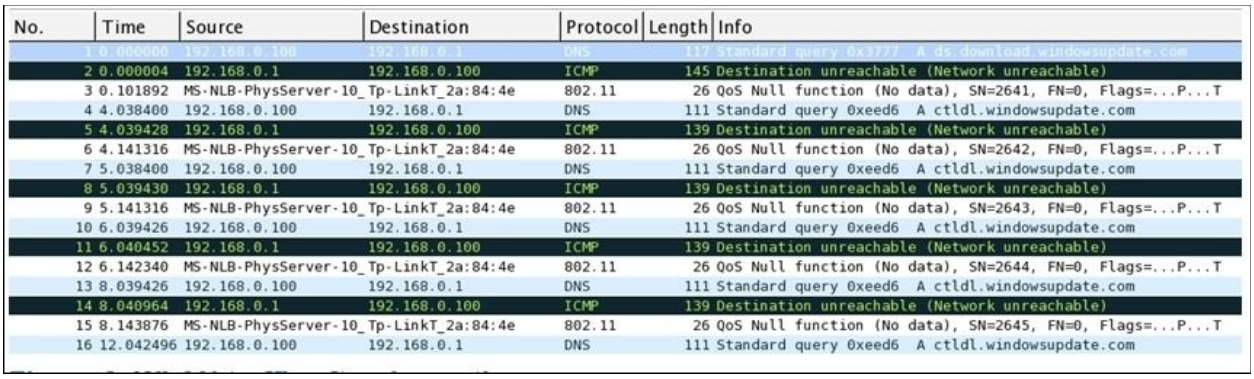
Go to **Edit** | **Preferences**, expand **protocol** section and select **IEEE 802.11** and make the changes. Refer to the following screenshot and make the changes that are highlighted:



Once you have set the configuration as shown in the preceding screenshot, click on the **Edit** button next to **Decryption Keys** (to add the **WEP/WPA** key). Refer to the following screenshot:



Click on **New** and you will be presented with the same dialog where you can add the **WEP/WPA** key in order to decrypt the preceding communication that we saw. After all the changes have been made, click on **OK** under **Apply**. Now, you will be shown the decrypted traffic similar to the one shown here:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.100	192.168.0.1	DNS	117	Standard query 0x3777 A ds.download.windowsupdate.com
2	0.000004	192.168.0.1	192.168.0.100	ICMP	145	Destination unreachable (Network unreachable)
3	0.101892	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2641, FN=0, Flags=...P...T
4	4.038400	192.168.0.100	192.168.0.1	DNS	111	Standard query 0xeed6 A ctldl.windowsupdate.com
5	4.039428	192.168.0.1	192.168.0.100	ICMP	139	Destination unreachable (Network unreachable)
6	4.141316	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2642, FN=0, Flags=...P...T
7	5.038400	192.168.0.100	192.168.0.1	DNS	111	Standard query 0xeed6 A ctldl.windowsupdate.com
8	5.039430	192.168.0.1	192.168.0.100	ICMP	139	Destination unreachable (Network unreachable)
9	5.141316	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2643, FN=0, Flags=...P...T
10	6.039426	192.168.0.100	192.168.0.1	DNS	111	Standard query 0xeed6 A ctldl.windowsupdate.com
11	6.040452	192.168.0.1	192.168.0.100	ICMP	139	Destination unreachable (Network unreachable)
12	6.142340	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2644, FN=0, Flags=...P...T
13	8.039426	192.168.0.100	192.168.0.1	DNS	111	Standard query 0xeed6 A ctldl.windowsupdate.com
14	8.040964	192.168.0.1	192.168.0.100	ICMP	139	Destination unreachable (Network unreachable)
15	8.143876	MS-NLB-PhysServer-10_	Tp-LinkT_2a:84:4e	802.11	26	QoS Null function (No data), SN=2645, FN=0, Flags=...P...T
16	12.042496	192.168.0.100	192.168.0.1	DNS	111	Standard query 0xeed6 A ctldl.windowsupdate.com

Figure 6: WLAN traffic after decryption

The same list pane that we saw in the beginning of this section for this capture file is shown in a decrypted format now. Here, we are able to see the **ICMP** and **DNS** packets (normal network traffic); this is the normal traffic I was talking about. To manage the keys, there is a more effective way where you are not required to open the **Decryption keys** dialog from the **Preferences** section under **IEEE 802.11**. Just navigate to **View** | **Wireless toolbar**; this will add a new toolbar just below the display filter area.

Once added, you can easily mage the WEP/WPA keys. The dropdown showing **Wireshark** is really helpful and will enable you to toggle encryption on/off. If you choose **None** from the list, the decryption will be disabled and your traffic will be back to normal from just 802.11 wireless traffic. If you choose **Wireshark**, as in the preceding screenshot, then the decryption will be applied.

Summary

What we discussed here is not going to facilitate you with every scenario that can be seen in wireless communication, but definitely, it will give you a jump start.

The IEEE 802.11 standard works over radio frequencies for communication purpose. The protocol that works behind WLANS is CSMA/CD, which facilitates a collision-free environment that is required for a wireless infrastructure. Under 802.11, there are multiple standards that have been developed, and this provides a robust solution for different infrastructure-based requirements.

Sometimes, you need to look at the RF energy level too, which can really play a big role in performance upgrade. Due to various devices that work over the same spectrum of 2.4 Ghz, it is possible that your WLAN signals may get distorted. What you need in such cases is a spectrum analyzer, which lets you analyze and monitor the RF energy flowing around you. To do so, you need special hardware that can be purchased from an online tech store, and you need to pair the same hardware with software that lets you use the same, for example, Metageek's Wi-SPY hardware paired with Channelyzer.

Kismet is a graphical tool available in Kali Linux that lets you collect various advanced details about the wireless networks that are available around you and the devices connected to those networks. Kismet comes with various customization options that can be really helpful while you look for specific information. Kismet also facilitates users with several graphical features to plot live traffic over a graph for a particular duration.

In a conventional WLAN environment, there is an AP and an STA that communicate with each other. Before the actual data transfer takes place, both the devices are supposed to negotiate the session over a key (password and encryption algorithm), which will be used by both the devices that are communicating to maintain the integrity of the data that is sent.

There are commonly three types of frames that you will see while working with Wireshark: management, control, and data frames. These are the packets that you

can see in the details pane once a packet is selected. Management frames control the establishment of the connection, control frames control the transfer of management, and data frames simply consist of the actual data that is sent.

Authentication protocols such as WEP and WPA take care of how an AP and STA negotiate to start communicating.

EAP is used to let the exchange of master keys take place. As defined in RFC 3748, EAP is an authentication framework that supports multiple kinds of authentication methods, and to execute EAP, you do not require an IP because it runs over data-link layer.

EAP with LAN becomes EAPOL, which is used in 802.11 infrastructures (RADIUS/AAA) for the exchange of master keys. As per the normal pattern, an AP broadcasts beacon frames that STAs listen for. If not, then the STAs will send a probe request to get connected by themselves. Then, the AP and STA conduct an authentication session and negotiate until both the hosts are convinced with each other. Once this is done, the AP would send a success message to the STA.

Using Wireshark, it is possible to decrypt WEP communications by simply adding wireless network keys with the protocol in use and modifying the preferences for the IEEE 802.11 protocol.

The monitor mode used to capture the relevant packets can be configured easily over a Linux-based system, and it is essential for Wireshark 802.11 analysis.

RTS/CTS are used in contrast to CSMA/CA in 802.11, which keeps the medium collision free and easy to work with.

Using the hash function, **Password-based key derivation function** (**PBKDF2**), the 256-bit preshared key is evaluated using the passphrase.

Practice questions

Q.1 After reading the IEEE 802.11 section in this chapter, make an extensive note regarding this protocol and whatever you have understood—take help from the respective RFC if you want to.

Q.2 Install any Linux-based system live on an individual machine and try to enable the monitor mode using the commands mentioned in this chapter.

Q.3 Capture the packets with the monitor mode off and the promiscuous mode on first, and then capture with the monitor mode on and the Promiscuous mode on. Analyze the difference.

Q.4 Install the Aircrack tool on your Windows machine and try capturing the 802.11 traffic around you.

Q.5 What is the difference between the various standards available in 802.11 (b/a/g/n/i.)?

Q.6 Suppose you have a router, and over to one end of the router you have a switch connected, which further connects to multiple wired clients. Over the other end of the router, you have a wireless access point connect, which serves as a medium to let various wireless devices connect to the corporate network. Now, send a packet from the wireless domain to the wired domain and analyze the packets while they transit between the domains. What difference would it make in the 802.11 header?

Q.7 What can be happen when your wireless NIC does not support the monitor mode or the promiscuous mode? Explain the importance of each.

Q.8 To view the availability of the probe requests that your device has sent to the access point, which display filter would you use?

Q.9 Configure your AP with the WEP-Open authentication and then try to connect to it using the AP while capturing the traffic, and do the same with WEP-Shared and analyze the difference in the pattern of the packets that appear.

Q.10 Which one is better: WEP-Open or WEP-Shared key and why?

Q.11 Use a capture filter to capture traffic only from your host, access point, and the broadcast address. Does this help you to decrease the noise?

Q.12 Configure your wireless interface in the monitor mode to a specific channel and capture the WLAN traffic then.

Q.13 What is the difference between the WPA-Shared key and WPA-Enterprise authentication protocols? Elaborate the same.

Q.14 Duplicate the scenario where you have a WEP-Shared key configured access point capture, with quite a good amount of traffic for the same, and try to decrypt the traffic you have using the WEP key.

Q.15 Why is WEP-Open better than the WEP-Shared key authentication mechanism?

Q.16 Can you figure out a way that you can forcefully disassociate a wireless client from it's own currently connected network?

Q.17 For deauthentication packets, how many types you do think exist? Modify the coloring rule for the same to view the packets uniquely. In what way are they different from the disassociation packets?

Q.18 While analyzing the WPA handshake, do you observe any open-system-based authentication before the actual handshake? If it is there, then analyze the traffic and explain what is it for?

Q.19 Configure your access point with the WEP protocol encryption capture normal 802.11 wireless frames. Then, using the same approach that we discussed, try to decrypt your traffic using the key for your network.

Q.20 Is it possible to decrypt the traffic using the ASCII format key or you can you also mention the key in HEX format? If yes, in which case can writing the key in HEX format prove worthy?

Chapter 7. Network Security Analysis

This chapter will teach you how to use Wireshark to analyze network security issues, such as analyzing malware traffic and foot printing attempts. You will learn how to use Wireshark for network security analysis. This chapter will cover the following topics:

- Analyzing port scanning, foot printing, and attack activities
- Lab up—port scanning with Nmap
- Analyzing brute force attacks
- Lab up—analyzing brute force attacks
- Inspecting malicious traffic
- Lab up—inspecting malicious traffic
- Solving real-world CTF challenges
- Practice questions

Up to this chapter, I have tried to make you aware of how one should use Wireshark to analyze the packets flowing around. We have just focused on how to use this sniffing tool for basic analysis purposes. However, what I am about to tell you is that in most of the places, Wireshark is used for security-analysis purpose, ranging from basic footprinting attacks to advanced Trojan-based attacks.

Using a couple of scenarios in my virtual lab, I will try to duplicate the most common one, along with capturing the live traffic between the attacker and the victim. Later on, we will dissect the trace file to get an idea of how malicious traffic looks like. We will use this knowledge base to create IDS/IPS or firewall signatures in an attempt to protect our internal critical infrastructure by analyzing the traffic shown in Wireshark.

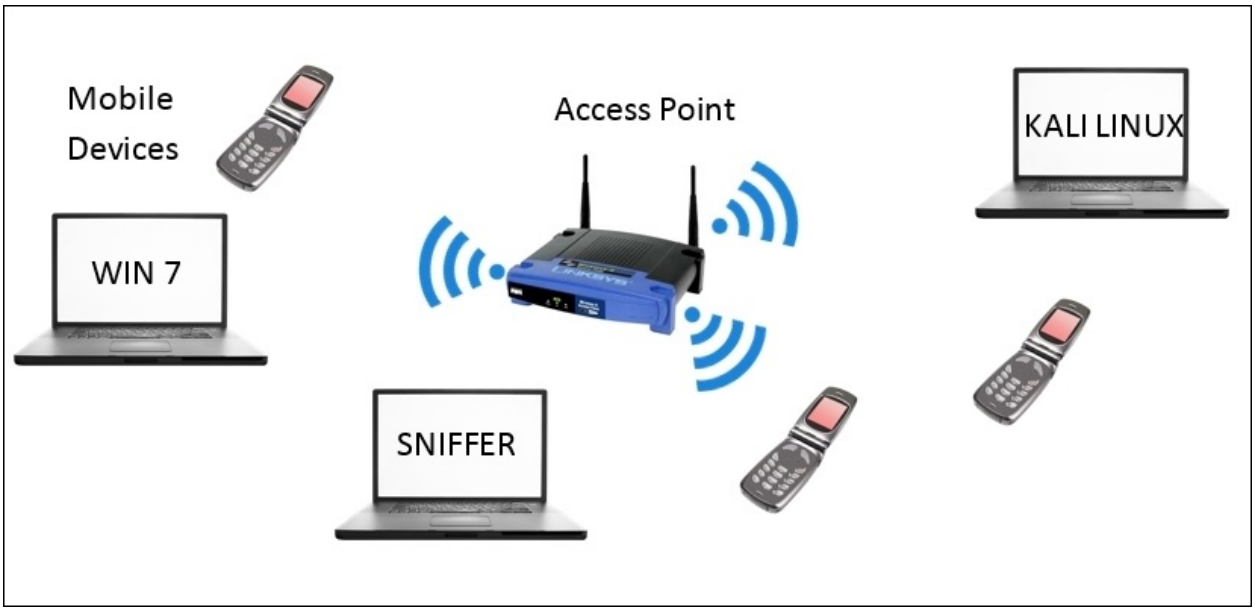
To achieve all this, you need to change your perspective a little bit. In other words, you need to act and think like a security professional who is in charge of the corporate network and constantly working to tighten the perimeter that will make the attack process more complex for bad users. We can start all of this by analyzing the packets captured for our daily usual traffic and also duplicate certain scenarios.

Information gathering

The primary step in the exploitation process is to collect as much information as you can. In today's world, gathering specific and relevant information about a person or an organization is not so difficult (using search engines), and this is where everything begins. A lot of security professionals will start launching attacks directly on the targets, which is not appropriate in the beginning. Let's say, for example, there is an ABC Corp. Ltd. located in the next block, and an XYZ attacker is planning to exploit it in terms of physical security (to get entry to the server rooms or any high-valued target available inside). To do so, the first thing the attacker should know is the working hours and the non-working hours. Then, they should know about the working days in the targeted company. The attacker should also know about the physical layout of the building the company is located in, and they should have some basic knowledge about the security policy. With all this information, the attacker should be able to identify the weak points inside the premises that might be an easy target and can give access to what they are looking for. Did you notice what just happened in the preceding scenario? We assumed that the attacker is collecting useful information and then planning and figuring out the easy targets to attack, because following this approach will improve the chances of success. Footprinting and reconnaissance are synonyms for the term *information gathering*. The chances of success would be higher if you are following the planned approach.

Let's use the same approach in targeting an organization using networks. The first step would be to identify the public IP address of the organization, the subnet it belongs to, and the range of IP addresses allocated to the organization. This basic information can be passively (without directly interacting with the company's network) collected through the use of DNS lookup services available online. We can try to check whether zone transfer is available, which can give some juicy and granular details regarding the organization's infrastructure we are targeting. After you have collected the basic information and have mapped the basic layout, you are ready to perform a **port scan**. I would prefer that you do a **ping sweep** first, which will tell you about the live machines over the network, and from where you will get to know more about the network (while performing a ping sweep, you can modify the TTL value to figure out the internal LAN architecture).

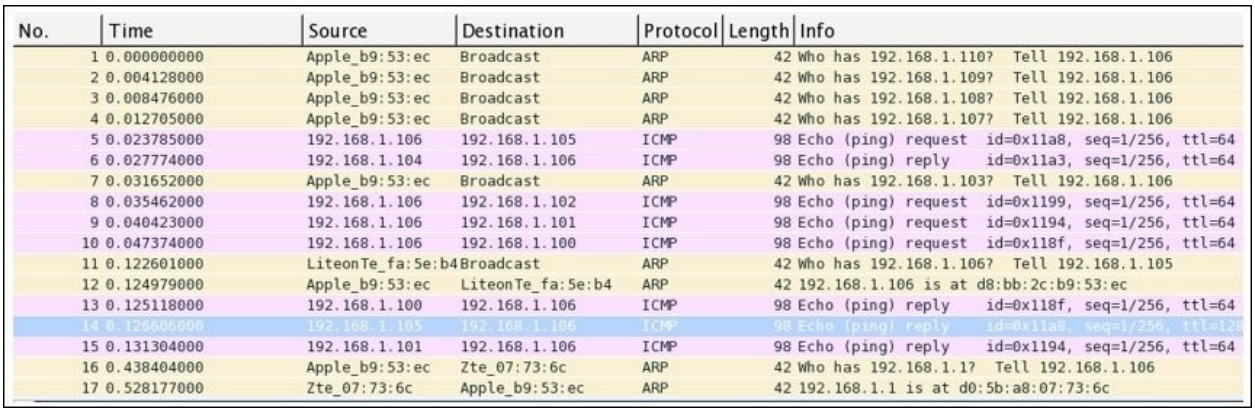
Before we go ahead and try duplicating the most common scenarios, I want you to visualize the local virtual computer infrastructure I have created for practice purpose. Refer to the following figure:



Hopefully, now you have a rough idea about my internal network that I'll be working with. The access point located at `192.168.1.1` assigns the IP address to all these devices using DHCP (the DHCP range starts from `192.168.1.100` and continues up to `192.168.1.110`; it means I can have a maximum of 10 DHCP clients at one instance). For this chapter, the IP address for our attacking machine is static assigned to `192.168.1.106`.

PING sweep

Let's begin with our first scenario where an attacker would try to perform a ping sweep attack over the subnet, and the traffic generated is captured by our sniffer listening through its interface in the promiscuous mode Refer to the following figure that displays the traffic pattern that was generated after running a bash script the script pings each IP starting from 100 to 110):



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	Apple_b9:53:ec	Broadcast	ARP	42	Who has 192.168.1.110? Tell 192.168.1.106
2	0.004128000	Apple_b9:53:ec	Broadcast	ARP	42	Who has 192.168.1.109? Tell 192.168.1.106
3	0.008476000	Apple_b9:53:ec	Broadcast	ARP	42	Who has 192.168.1.108? Tell 192.168.1.106
4	0.012705000	Apple_b9:53:ec	Broadcast	ARP	42	Who has 192.168.1.107? Tell 192.168.1.106
5	0.023785000	192.168.1.106	192.168.1.105	ICMP	98	Echo (ping) request id=0x11a8, seq=1/256, ttl=64
6	0.027774000	192.168.1.104	192.168.1.106	ICMP	98	Echo (ping) reply id=0x11a3, seq=1/256, ttl=64
7	0.031652000	Apple_b9:53:ec	Broadcast	ARP	42	Who has 192.168.1.103? Tell 192.168.1.106
8	0.035462000	192.168.1.106	192.168.1.102	ICMP	98	Echo (ping) request id=0x1199, seq=1/256, ttl=64
9	0.040423000	192.168.1.106	192.168.1.101	ICMP	98	Echo (ping) request id=0x1194, seq=1/256, ttl=64
10	0.047374000	192.168.1.106	192.168.1.100	ICMP	98	Echo (ping) request id=0x118f, seq=1/256, ttl=64
11	0.122601000	LiteonTe_fa:5e:b4	Broadcast	ARP	42	Who has 192.168.1.106? Tell 192.168.1.105
12	0.124979000	Apple_b9:53:ec	LiteonTe_fa:5e:b4	ARP	42	192.168.1.106 is at d8:bb:2c:b9:53:ec
13	0.125118000	192.168.1.100	192.168.1.106	ICMP	98	Echo (ping) reply id=0x118f, seq=1/256, ttl=64
14	0.126606000	192.168.1.105	192.168.1.106	ICMP	98	Echo (ping) reply id=0x11a8, seq=1/256, ttl=128
15	0.131304000	192.168.1.101	192.168.1.106	ICMP	98	Echo (ping) reply id=0x1194, seq=1/256, ttl=64
16	0.438404000	Apple_b9:53:ec	Zte_07:73:6c	ARP	42	Who has 192.168.1.1? Tell 192.168.1.106
17	0.528177000	Zte_07:73:6c	Apple_b9:53:ec	ARP	42	192.168.1.1 is at d0:5b:a8:07:73:6c

Figure 7.1: Ping sweep

Starting from packet 1–4, the Kali box started generating an ARP request because of the ICMP `ping` command issued, but none of those IP's are allocated. Hence, we did not receive any replies. In packet 5, Kali box sent a `ping` request to 105, and the reply for it was received in packet 14, which means the device is on. Then, in packet 7, an ARP request was sent to 103, but this IP might also be unallocated for the instance, so no reply again. In packets 8–10, Kali box sent an ICMP request packet to IP's 102, 101, and 100. The reply for the same can be seen in packets 13 and 15 from IP's 101 and 100. For 102, we did not receive any reply. It might be any device blocking our ping probes or some mobile device not responding to the ping probes. Finally, in packet number 17, we can see that the access point is informing the Kali Machine about its physical address. If you scroll down through your trace file, you would see various replies from online devices describing their physical addresses.

Half-open scan (SYN)

The next step in the process would be to scan any specific device that you would like to target. Let's suppose I want to target my Win7 machine running at IP `192.168.1.105`. My next step should be to check for available services running on that box. By services, I mean HTTP daemons, mail server daemons, FTP server, and so on. You might be wondering what a half-open scan is? Look at the process of a TCP three-way handshake we discussed, where the client initiates the connection by sending a `SYN` packet if the server is available. Then, the client receives the `SYN`, `ACK` packet, and in return, the client sends an `ACK` packet to the server for completing the handshake process.

Now, what would happen if the `ACK` packet sent in the last step of the TCP handshake is never sent to the server? The server will wait for a specific period before terminating the handshake process initiated by the client, and the connection to the specific TCP service would never be completed. That's why this type of scan is called half-open scan. This is a very common scanning technique used by the majority of users who are involved in malicious activities, being aware of such traffic pattern could help us in identifying future risks. I initiated the half-open scan from Kali box to target Win7 box. I am using Nmap, which is an open source tool available for every platform and can be downloaded for free from http://nmap.org (to use the tool, you can refer to various tutorials available online). The traffic generated because of the `SYN` scan is captured and shown in the following screenshot:

Figure 7.2: Half-open scan

There are three kinds of replies that you can see after the scanning is completed: Open, Closed, and Filtered. Now, the point to discuss is what these states mean and what relation do these states have with the packet shown in the preceding screenshot. Let's look at the states in more detail here:

- **Open**: If a service is open, then a `SYN`, `ACK` packet will be sent back to your machine for taking the TCP handshake process to the next step of completion. In packet `26`, Kali sent an `SYN` request to port 135 and received a `SYN`, `ACK` reply in packet `28`.
- **Closed**: If a service is not available to respond, then you would receive an RST packet that confirms that the service/daemon is currently not running. In packet `22`, a `SYN` request was sent destined to port `113`. In packet `25`, the RST packet for the same is received. It states that the service is not available at this moment.
- **Filtered**: Sometimes, a firewall might be configured between you and your target that might be intercepting your requests and would be dropping them without forwarding them to the target. In such scenarios, you might be seeing port states such as `open|filtered`, `closed|filtered`, or just `filtered`.
- Let's suppose you are trying to scan an HTTP webserver that is outside your VLAN and is restricted by the firewall from your machine. Then, the handshake process would never move to the second step, that is, you will never receive a reply of any kind. You will not receive any `SYN`, `ACK` or `RST` packet.

Using this scan type, you can identify the state of the services running. However, using this kind of scan type will generate a hefty amount of traffic too. The scan I initiated was completed in 1.76 seconds, and in such a short time, it generated 2024 packets between the two machines. Now, this proves disadvantageous. Any well-configured IDS/IPS can figure out such activity very easily, which will in turn trigger an alert to notify the security admins. Nmap has configurable switches that can help you out in these situations too.

OS fingerprinting

Being aware of the operating system running on the target takes the scanning process to the next step in the methodology. If the attacker knows about the OS you are running, the patch level of your OS, and the version of your OS, then it would be quite simple to structure the attack process and will increase the chances of success.

There are a couple of tools available in Kali that will let you identify the target's OS. It is not 100 percent accurate, and it is correct most of the times. Now, how do you think a simple tool is available to identify the remote machine's OS? I will tell you the secret. Every OS has a different way of implementing the TCP stack. So, a packet received from the remote machine will have certain fields in it such as TTL, fragment offset, and most importantly window size. By comparing the values in the packet with the database we have, it will tell you the OS. For example, if you try to ping a Windows machine, the TTL value returned would be 128, and if you ping a Linux machine, the TTL value would be 64 most of the time. Simple, isn't?

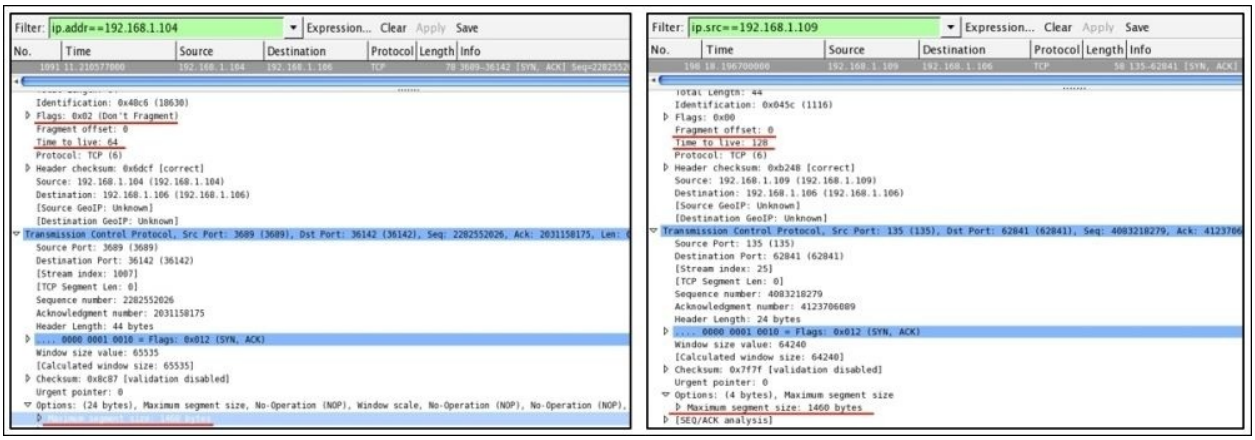
There are two types of fingerprinting: active and passive. They are described here:

- **Active fingerprinting**: When you are directly interacting with the system, the requests and responses are directly shared between you and the target. This kind of scan can be really dangerous and is not stealthy. The captured packets will give you values that can be matched with the signature we have to identify the OS running on the remote machine.
- **Passive fingerprinting**: When you are just listening for the packets originated or destined to the target, the values in the packets can be examined in order to identify the OS running. A disadvantage off passive type scan is that it is not as accurate as active fingerprinting. But the process would be stealthier than active scans.

Using the `nmap` scan, I will try to fingerprint a machine at IP `192.168.1.109` and `192.168.1.104` and see what kind of traffic is generated due to such requests. The type of scan we will witness is active scanning, and we will be directly interacting with the systems. We won't just rely on Nmap's output to confirm the OS. The packet that would be returned to our attacking machine is the base of all

necessary information, which I will try to dissect for your better understanding.

I will use the `nmap -O 192.168.1.109,192.168.1.104` command for active OS fingerprinting, where the `-O` switch is for checking the OS and its version. Refer to the following two screenshots to compare the outputs they present to us:



Using just the TTL field, we can verify that the first traffic we captured is from some Linux/Macintosh-based machine, as the TTL value is `64`. The second traffic screenshot belongs to a Windows machine as the TTL value is set to `128`.

Secondly, the maximum segment size highlighted at the bottom can also be a deciding factor for OS fingerprinting. In both cases, it is `1460`. The value is correct if you are talking about a Linux-based machine, but if it is a Windows machine, then you might observe that the value is `1440` most of the time.

For both Linux and Windows platforms, the Fragment Offset field should be 0 (not set). See how, simply by observing basic fields in the TCP header and IP header, we were able to fingerprint on our own. Now let's see what `nmap` has to say.

Refer to the following screenshots for illustration:

Figure 7.3: nmap output for 192.168.1.104

The `nmap` output for the machine IP `192.168.1.104` detects that the machine might be one of these OSes running (in the red box). I think what we figured out and it is quite close. OS detection by `nmap` is done by analyzing the requests and responses traffic that the target machine generates.

The `nmap` output for the machine at `192.168.1.109` says that it is a Windows server machine, may be `SP1` or `SP2`. This time, the result is more accurate than the previous one. We also presumed that it would be a Windows OS, and it is.

The traffic generated from both these scans would be quite similar to the `SYN` scan traffic where the TCP handshake request and `ICMP` request/replies can be seen. Once the attacker's machine running `nmap` receives the replies for the requests made, it will start analyzing and comparing the results with the database of the results it already has. Thus, in the end, after comparing the values, Nmap will present you with the most accurate results.

So, if you are seeing a lot of `RST` or `RST`, `ACK` packets sent from one of your internal LAN machines, then it is something that you should be worried about. Better create signatures for such traffic in your firewall so that they can alert you.

ARP poisoning

As we all know, the function of the ARP protocol is to translate an IP address to its corresponding MAC address. By doing so, the devices are able to communicate effectively in a LAN-based network. Any device that wishes to get connected with the other device on the same network requires the MAC address of the other hosts. Every OS maintains a list of communicating devices that can be populated in the terminal window using the `arp -a` command. The same command is used on every platform. We have also seen the ARP requests and reply packets that are used by the devices connected to the local network to gain the MAC addresses of other devices.

For instance, I have a local network too, which is being governed by the router (gateway) located at `192.168.1.1`, and there are 3 devices connected to it. The following table lists all the required information specific to the devices connected, which we will use later:

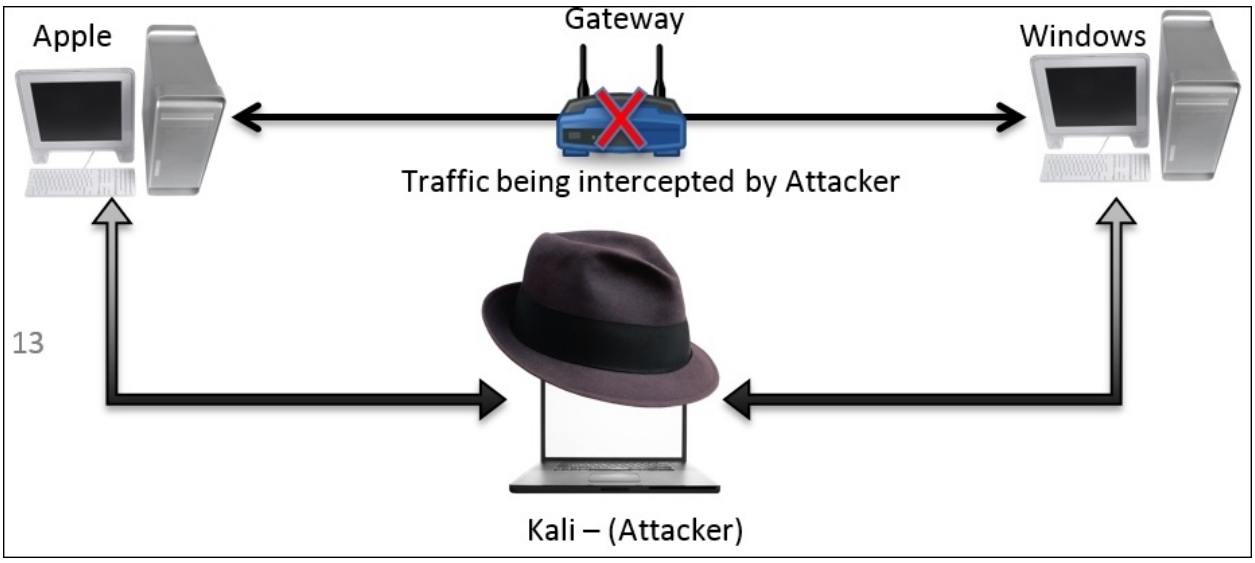
Device	**IP Address**	**MAC Address**
Router (default gateway)	192.168.1.1	D0:5B:A8:07:73:6C
Apple (victim)	192.168.1.103	D8:BB:2C:B9:53:EC
Windows server (victim)	192.168.1.109	00:0C:29:B3:CB:B6
Kali Linux (attacker)	192.168.1.106	00:0C:29:5D:A7:F7

This preceding information is listed in the ARP cache of every host connected to the local network. You must be thinking exactly how this is being populated in the local cache. Whenever any device intends to communicate with the other device, the requesting device sends a broadcast to the whole subnet. Then, the device to which the IP address belongs replies with it's MAC address using a unicast packet. For example, if the Apple machine wishes to communicate with the Windows machine located at `192.168.1.109`, Apple will send a broadcast asking for the Windows MAC address stating `Who has 192.168.1.109? Tell 192.168.1.103`. Then, as soon as the Windows machine gets to know about the

request, the ARP reply unicast packet stating `192.168.1.109 is at 00:0C:29:B3:CB:B6` will be broadcasted. This is how the process works.

The preceding packets transfer will only happen if the Apple machine has the Windows MAC address in it's local cache. After searching in the local cache, the request is sent to the default gateway. If the default gateway knows about it, an ARP reply packet is sent by the gateway itself. If not, then the request will be forwarded to the subnet from where the destination PC will reply with the physical address using a unicast packet. After this, the conversation can happen using TCP/IP.

ARP poisoning is used to poison the local cache of the victim that enables the attacker to sniff the data that is travelling between the two victims. The attacker intercepts the traffic and then forwards it to the other side. Refer to the following illustration:



We can poison the local ARP cache of both the victims and can achieve the same. There is one more thing you need to configure: IP forwarding on Kali so that your attacking machine would be able to transfer the traffic back and forth without any loss or without letting the victims get suspicious. Follow these steps to achieve ARP poisoning:

- First, configure IP forwarding using the `echo '1' >`

`/proc/sys/net/ipv4/ip_forward` command.

- Once this is configured, you can go ahead and send unsolicited ARP reply packets to both the victims for poisoning the cache. Before we poison it, let's take a look at how they look in normal form, for both the victim machines:

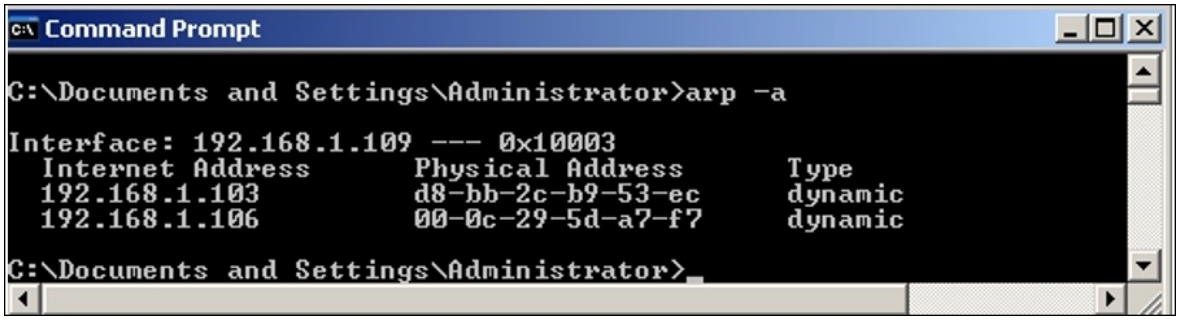


Figure 7.4: Windows server cache

To populate entries in `linux arp` cache use similar commands; refer to the following screenshot for reference.

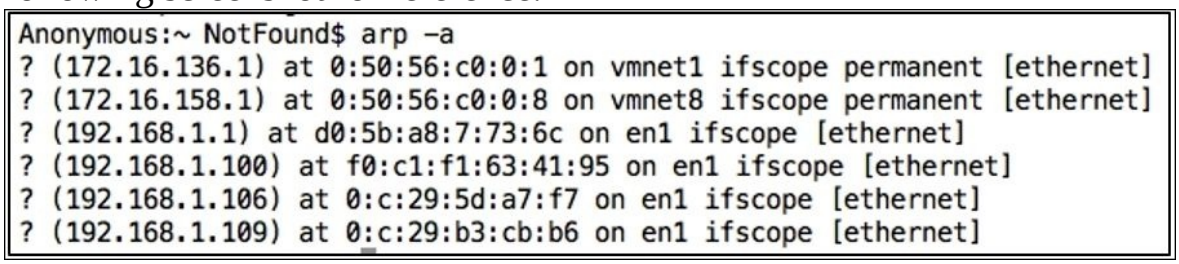
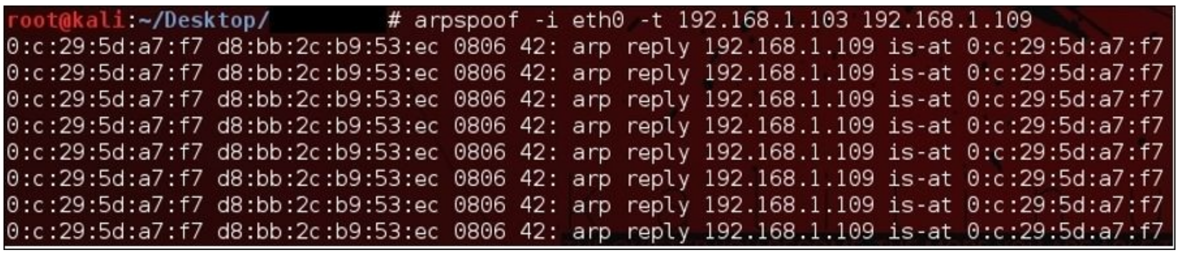


Figure 7.5: Apple cache

- Now, let's start sending unsolicited ARP reply packets to the Windows server machine that Apple machine is located at `00:0C:29:5D:A7:F7`. The same packet would be sent to the Apple machine that the Windows server machine is located at `00:0C:29:5D:A7:F7`. If you notice, the MAC address specified in the packets sent to the Windows and Apple machines belongs to Kali (the attacker). Refer to the following screenshot to check out the command I used for the spoofing fake MAC addresses:

```
root@kali:~/Desktop/[redacted]# arpspoof -i eth0 -t 192.168.1.109 192.168.1.103
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.103 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.103 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.103 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.103 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.103 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.103 is-at 0:c:29:5d:a7:f7
```

Figure 7.6: ARP reply packets sent to the Windows server on behalf of the Apple device

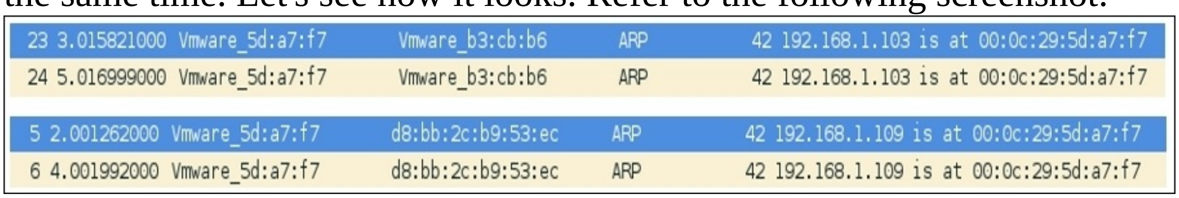


```
root@kali:~/Desktop/[redacted]# arpspoof -i eth0 -t 192.168.1.103 192.168.1.109
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
0:c:29:5d:a7:f7 d8:bb:2c:b9:53:ec 0806 42: arp reply 192.168.1.109 is-at 0:c:29:5d:a7:f7
```

Figure 7.7: ARP reply packets sent to Apple device on behalf of the Windows server

Using a one-liner command with few parameters, we were able to poison the victim's cache by sending numerous ARP reply packets.

- The traffic generated due to the preceding command was also captured at the same time. Let's see how it looks. Refer to the following screenshot:



```
23 3.015821000 Vmware_5d:a7:f7   Vmware_b3:cb:b6    ARP   42 192.168.1.103 is at 00:0c:29:5d:a7:f7
24 5.016999000 Vmware_5d:a7:f7   Vmware_b3:cb:b6    ARP   42 192.168.1.103 is at 00:0c:29:5d:a7:f7

5 2.001262000 Vmware_5d:a7:f7    d8:bb:2c:b9:53:ec  ARP   42 192.168.1.109 is at 00:0c:29:5d:a7:f7
6 4.001992000 Vmware_5d:a7:f7    d8:bb:2c:b9:53:ec  ARP   42 192.168.1.109 is at 00:0c:29:5d:a7:f7
```

- Once multiple number of such packets are received by both of the victims, they will start believing it and accordingly will update the cache. Let's have a look at both the machine caches to verify this. Refer to the following screenshots:

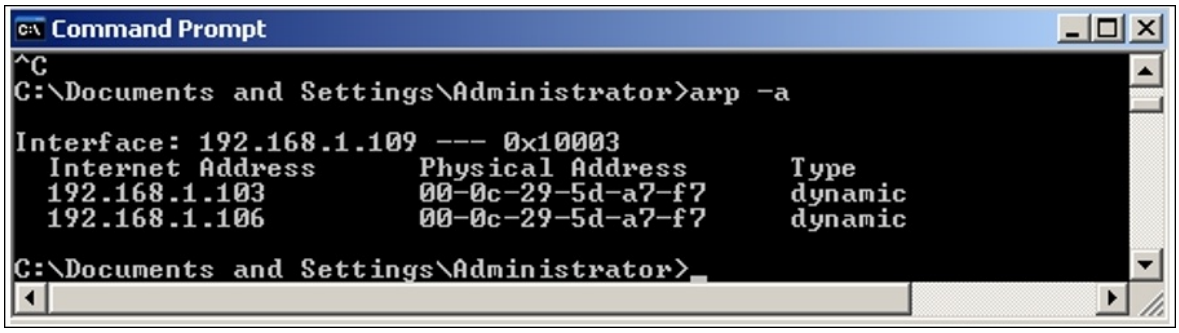


Figure 7.8: Poisoned window's cache

```
Anonymous:~ NotFound$ arp -a
? (172.16.136.1) at 0:50:56:c0:0:1 on vmnet1 ifscope permanent [ethernet]
? (172.16.158.1) at 0:50:56:c0:0:8 on vmnet8 ifscope permanent [ethernet]
? (192.168.1.1) at d0:5b:a8:7:73:6c on en1 ifscope [ethernet]
? (192.168.1.100) at f0:c1:f1:63:41:95 on en1 ifscope [ethernet]
? (192.168.1.106) at 0:c:29:5d:a7:f7 on en1 ifscope [ethernet]
? (192.168.1.109) at 0:c:29:5d:a7:f7 on en1 ifscope [ethernet]
```

Figure 7.9: Poisoned Apple's cache

- Now, whatever traffic is sent between these two devices will be forwarded through the attacking box. For verification purposes, I turned off the Windows server machine and tried sending ICMP packets from the Apple machine. Refer to the following output shown for the ICMP destination host unreachable replies coming from `192.168.1.106` (Kali):

```
Anonymous:~ NotFound$ ping 192.168.1.109
PING 192.168.1.109 (192.168.1.109): 56 data bytes
92 bytes from 192.168.1.106: Redirect Host(New addr: 192.168.1.109)
Vr HL TOS  Len   ID Flg  off TTL Pro  cks      Src      Dst
 4  5  00 0054 8554   0 0000  3f  01 7230 192.168.1.103  192.168.1.109
```

  The preceding output assures that the packets are being forwarded through `192.168.1.106`, hence making our ARP poisoning attack a success.
- Now, the question is how to secure yourself from such attacks. The best

thing I would suggest is to make manual entries for the device's MAC address in the local cache of the communicating client. This will definitely ignore unsolicited ARP reply packets while modifying the local cache. Refer to the following screenshot:

Figure 7.10: Adding a static entry to local ARP cache

Once you add a static entry in every possible host in your network, it won't be possible then to modify the local cache using the `arp` spoof tool. Similarly, for HTTPS traffic, you can use the SSL strip tool available online in order to sniff secure traffic.

Analyzing brute force attacks

Most of you must be aware of the popularity of brute force attacks. The chances of success are not high. Yet, many security professionals and malicious users implement their password-guessing ability with the help of modern tools. Brute force attack is just a way in which you try to log on to a particular service/application using the password dictionary that might have been created on the basis of the target's profile. Tools such as Cewl, Crunch, and John let you create dictionary files. Even you can salt the passwords. Discussing how to create one for yourself is out of the scope of this book, but I would recommend that you have a look at these tools (all of them come preinstalled with Kali Linux).

To analyze these common and malicious attacks, I will attempt to brute force two important services: Telnet and FTP. You might be aware of these two services and how much they are being used in corporate networking infrastructure. Telnet is used to perform administration of devices such as routers, switches, and different kinds of web servers remotely. FTP is used to transfer files efficiently with the assurance of integrity and confirmed delivery of the data.

First, take a look at most widely used protocol for remote administration that is often overlooked from a security standpoint. Using simple brute force techniques, any script kiddie can gain access to your network, and the consequences of such acts can be really destructive in terms of money and availability of the service. If dealing with consumers, then their records that might be worth millions, leading to full remote code execution of the administrative systems.

For this illustration, I have a Windows server machine running at `192.168.1.109` and an attacker at `192.168.1.106`. The attacker will first prepare its dictionary file and then will proceed to use an automated tool to attack over the Telnet administration service running under the Windows server machine. The traffic generated for such activities will be logged in through our wonderful sniffer for our analysis. I tried connecting to the Telnet service like a normal user using these steps:

- Using the Telnet command followed by the IP address, I was able to get connected to the service. In return, it printed a banner for me: `Welcome to Microsoft Telnet service.`
- Then, I supplied the wrong user credentials, which was not accepted by the server. Hence, it showed a login error, which stated `bad username or password.`
- Then, I supplied a legitimate set of credentials, which were identified and accepted by the service.
- Once the user is authorized, the Windows command prompt with certain authorization is presented along with a banner. `Welcome to Microsoft Telnet Server.`
- After I got connected, I was able to issue remote commands (Windows) from my machine itself.
- Then, at the end, to terminate the connection gracefully and to free up all resources that were allocated to use for smooth functioning, I issued the `exit` command that gave a message `connection closed by foreign host`.

Here is the screenshot illustrates the normal functioning of a Microsoft Telnet server:

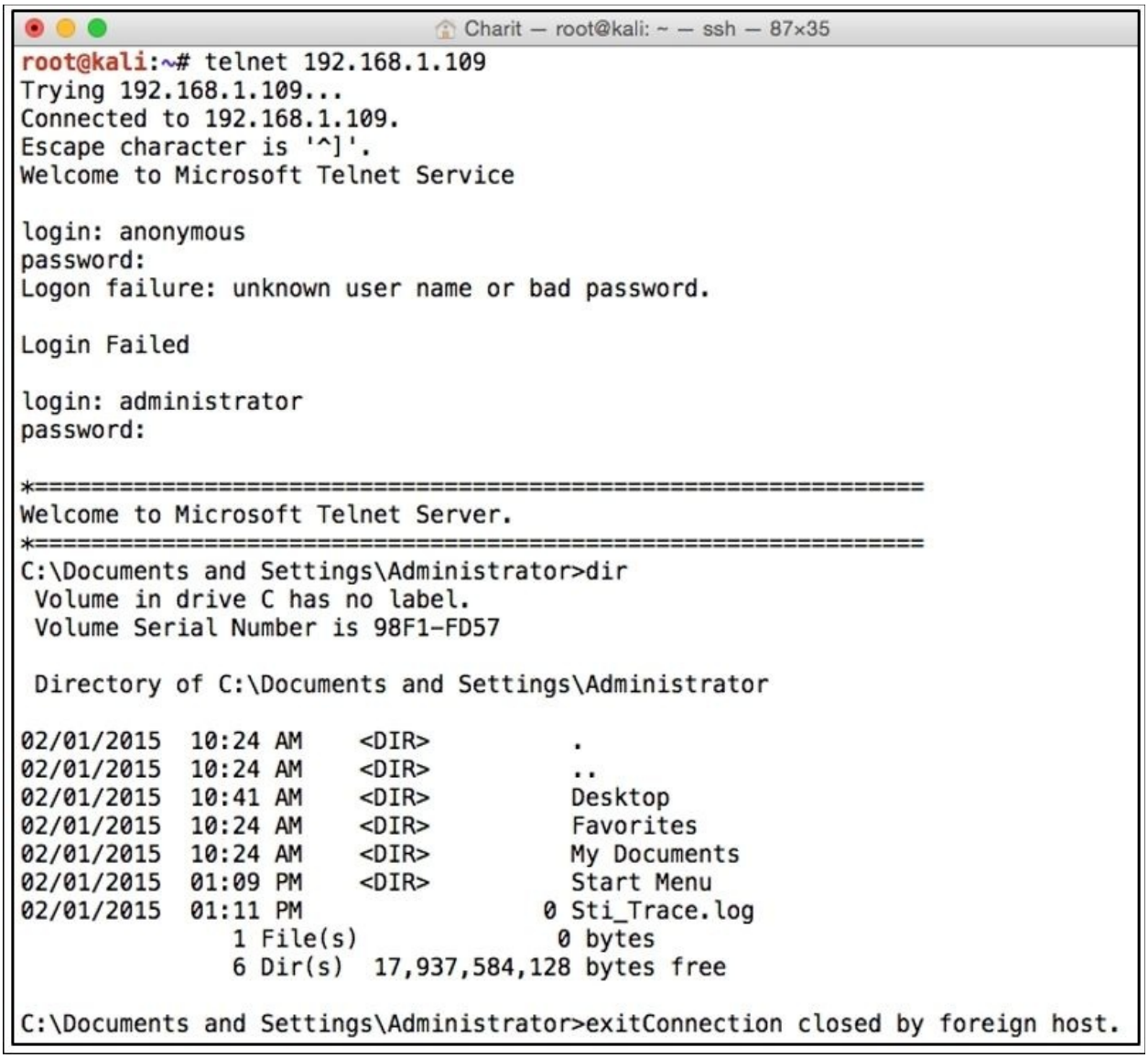


Figure 7.11: Telnet normal session

The traffic generated was also captured by Wireshark. Instead of showing the traffic, I decided to show you the whole communication in plain text format that you can achieve by assembling the TCP stream by right-clicking on the list pane and choosing **show TCP stream** (the Telnet server is configured with an echo option, so there is a chance we might see some characters echoed back from the server to the client). Refer to the following screenshot:

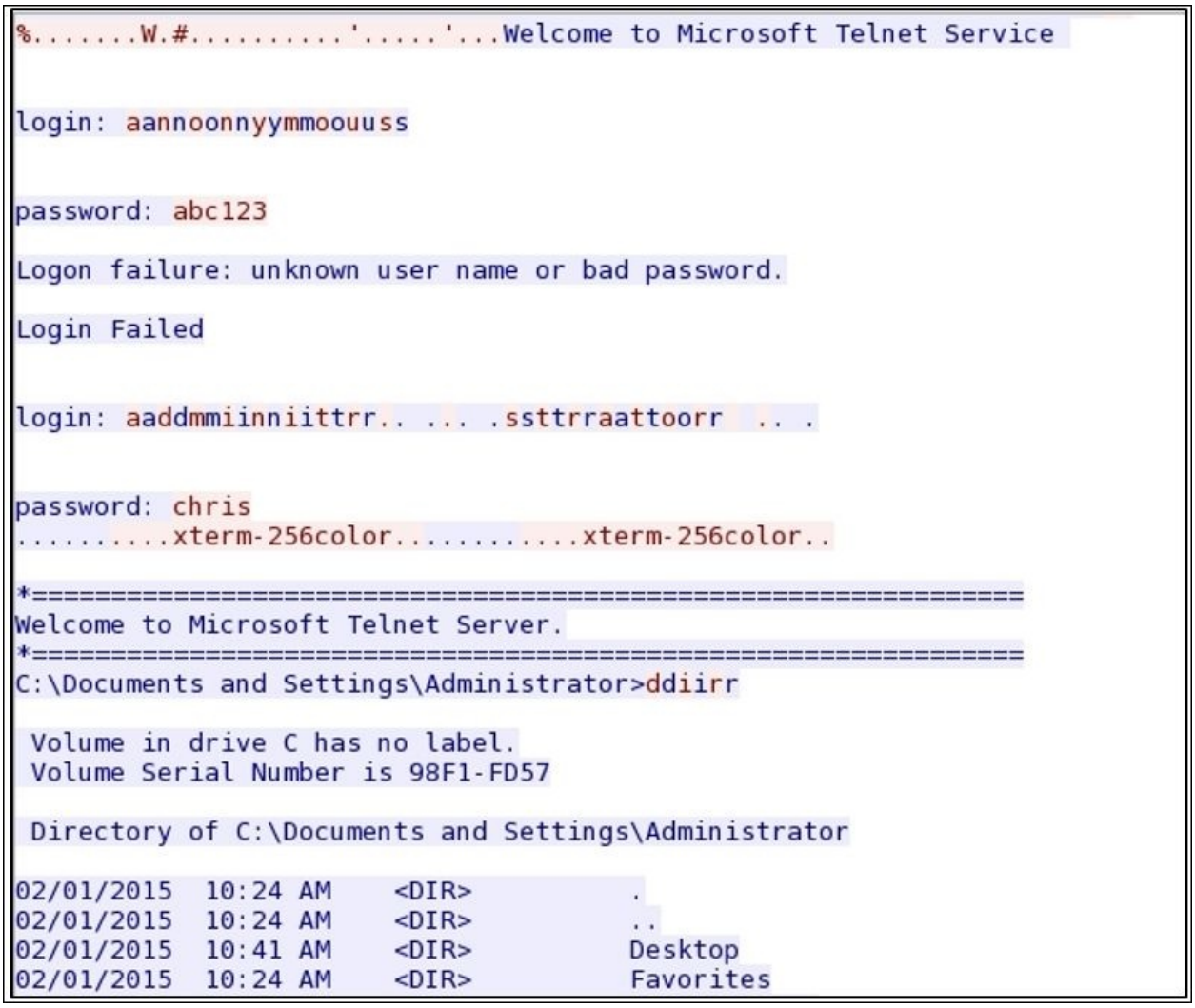


Figure 7.12: Telnet follow TCP stream

Everything we typed and received in response from the server is being shown in simple plain text readable form by just following the TCP stream.

Now, after seeing how a normal session looks, if you want to learn how to perform a brute force attack, follow these steps:

- Create a virtual pen-testing lab that consists of at least two machines: one will be an attacker (Kali) and the other machine can be of your choice (make sure you can install Telnet on it).
- Try pinging the target to test the connectivity. Issue the Telnet command to

create a normal session and test whether everything is working fine.

- Now, open Kali and issue the `medusa –h <target ip> -U <usernames file> -P <password file> -M telnet` command. Refer to the following screenshot:

```
root@kali:~# medusa -h 192.168.1.109 -U user.txt -P pass.txt -M telnet
Medusa v2.0 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Alice (1 of 8, 0 complete) Password: abc (1 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Alice (1 of 8, 0 complete) Password: efg (2 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Alice (1 of 8, 0 complete) Password: chris (3 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Alice (1 of 8, 0 complete) Password: mno (4 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Admin (2 of 8, 1 complete) Password: abc (1 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Admin (2 of 8, 1 complete) Password: efg (2 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Admin (2 of 8, 1 complete) Password: chris (3 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: Admin (2 of 8, 1 complete) Password: mno (4 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: root (3 of 8, 2 complete) Password: abc (1 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: root (3 of 8, 2 complete) Password: efg (2 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: root (3 of 8, 2 complete) Password: chris (3 of 4 complete)
ACCOUNT CHECK: [telnet] Host: 192.168.1.109 (1 of 1, 0 complete) User: root (3 of 8, 2 complete) Password: mno (4 of 4 complete)
```

Figure 7.13: Brute force—Telnet

At last, using a different set of combinations, we were able to brute force the server. The traffic generated because of all these attempts made one after another is of special interest to us.

- There is a lot of TCP and TELNET traffic generated in the file, which include traffic patterns such as the three-way handshake and transfer of data between the server and client through Telnet. However, not everything is of interest to us. Refer to the following screenshot:

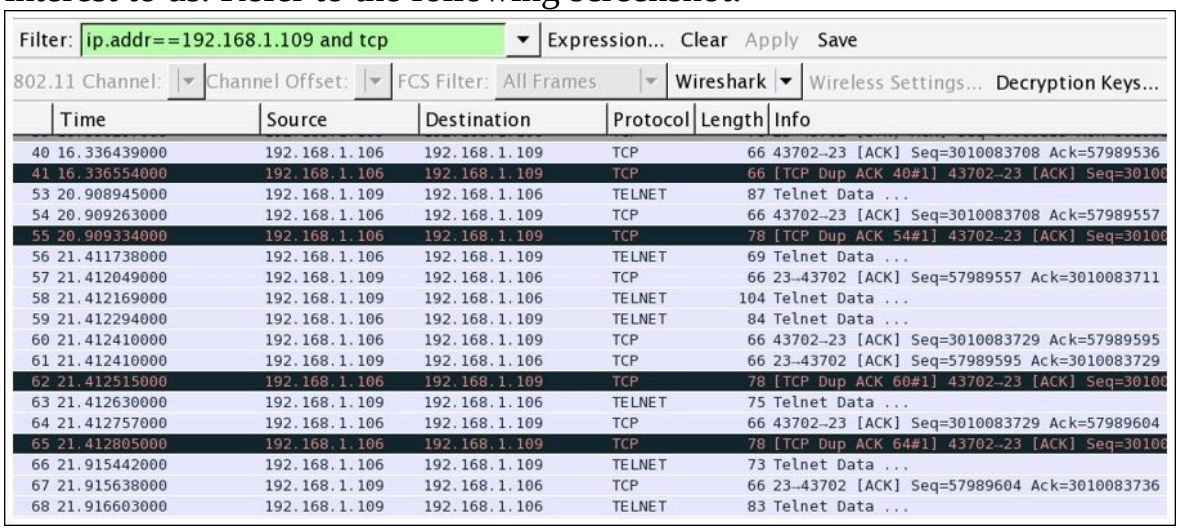
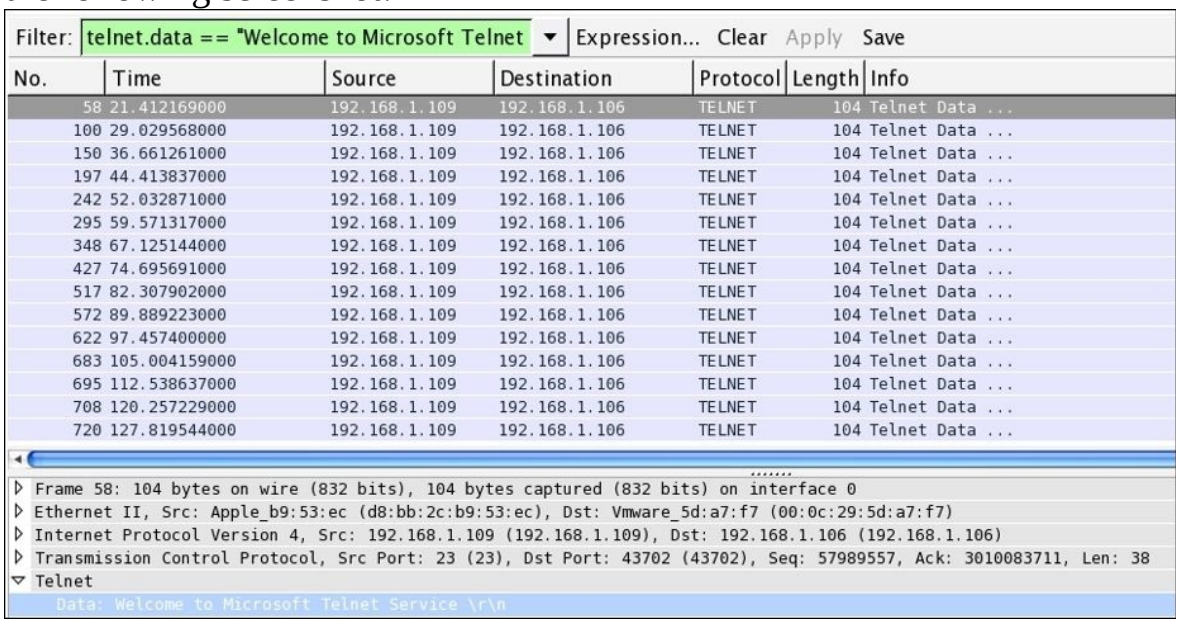


Figure 7.14: Telnet and TCP traffic between the server and our client

- To view only the malicious traffic, I applied another display filter that will show only the various connection attempts between the two hosts. Refer to the following screenshot:

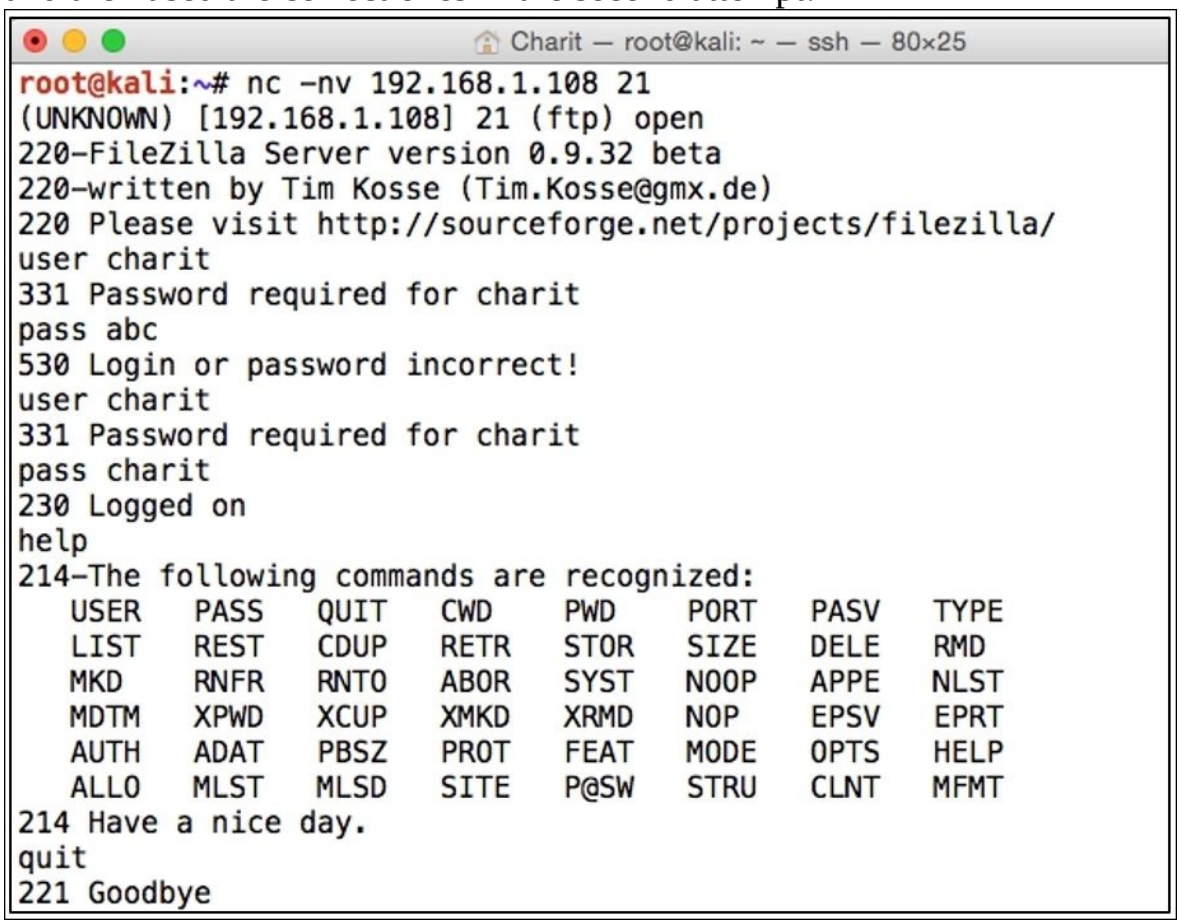


- Now, observe the display filter `telnet.data==Welcome to Microsoft Telnet Service` along with the Time column. The string I applied in as the filter is the same as the one we received as a banner while connecting to the service. The banner is printed approximately 15 times in a span of 100 seconds (less than a minute).
- Does this now seem suspicious to you now? If it is, then you can take preventive measures to protect your infrastructure by creating useful signatures for the same traffic pattern that will help you in getting alarmed.

Next, it's time to look at another popular service, FTP, that we discussed in earlier chapters in detail. Let's look at how a brute force attack would look like against the FTP service. FTP is a very crucial service. If attacked by any means, the service will crash or become unusable for the legitimate users. It can cause big trouble to the network admins with serious downtime. To deal with such activity that happens in day-to-day operations, you need to be prepared by being aware of the malicious traffic patterns that you can compare with the baseline traffic pattern we created earlier.

For testing and analysis purpose, I configured one FTP server at `192.168.1.108` over a Windows 7 machine and the attacker is at the same place over IP `192.168.1.106`. I used a Kali Linux operating system to duplicate the attack and normal traffic pattern scenario. Follow these steps if you want to duplicate it for educational purpose only:

- Configure the client and the server using whatever platform suits your needs best and make sure the connection between the FTP server and the client works freely without a single glitch.
- Now, first, we will try to visit the server using a legitimate user and will record the traffic. Later, we will use the **Follow TCP stream** option in Wireshark to view the traffic details in easy to understand plain text format.
- Refer to the following screenshot where I initiated the connection between the server and the client using the netcat client available over the Kali platform. I then logged in using the wrong credentials in the first attempt, and then used the correct ones in the second attempt:



- After I successfully logged in, I issued the `help` command to view the commands available for execution. Then, I issued the `quit` command to terminate the connection gracefully. Refer to the preceding screenshot.
- Our sniffer captured the whole conversation. Instead of viewing the traffic in the list pane, we are again seeing the assembled TCP stream. Refer to the following screenshot:

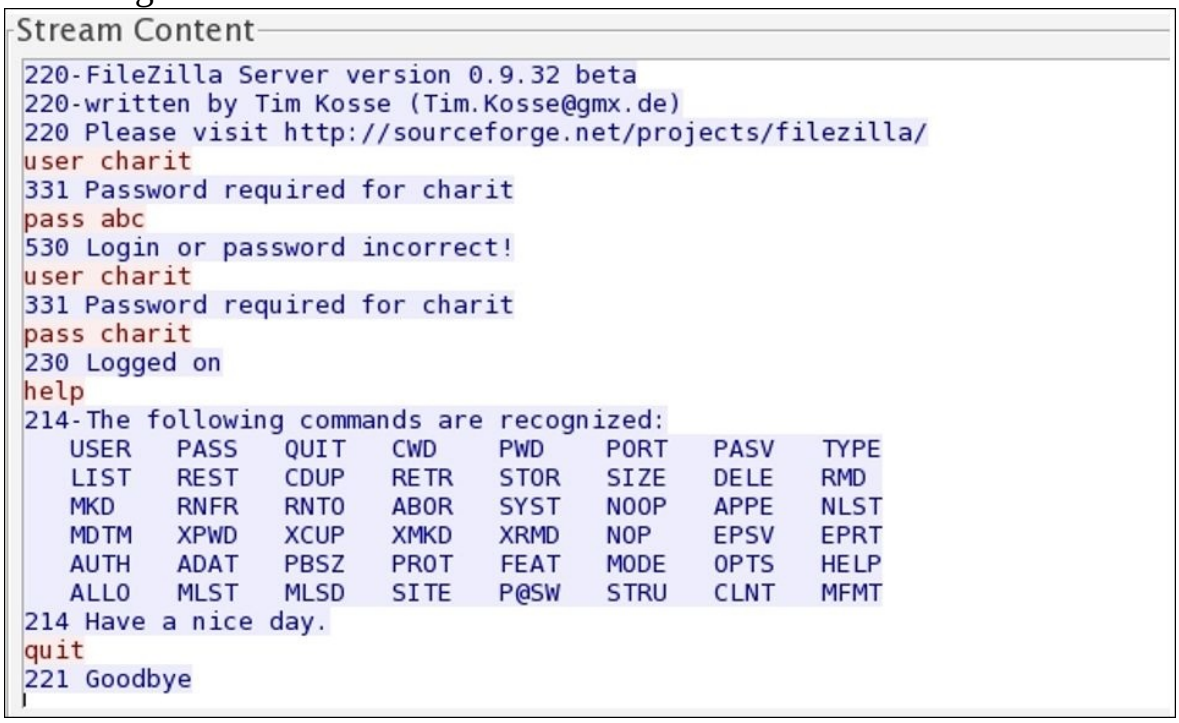


Figure 7.15: FTP assembled stream

- Now, as we have seen the normal traffic patterns that you would witness in every day operations, it's time to look at something malicious, such as the brute force attack attempts executed against your FTP servers. I used a different brute force tool that is it also popular among the category `THC-hydra`.
- Before you issue the command, make sure you have you own custom-made dictionary file that suits you well for your target (refer to the openwall website at http://www.openwall.com/wordlists/ to get the best dictionary files available).
- Once you have the dictionary file and the target up and running, issue the `hydra –l <username> –P <password file> ftp://<you target's IP`

`address>` command. Refer to the following screenshot:

```
root@kali:~# hydra -l charit -P pass.txt ftp://192.168.1.103
Hydra v7.6 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2015-09-12 18:16:00
[DATA] 11 tasks, 1 server, 11 login tries (l:1/p:11), ~1 try per task
[DATA] attacking service ftp on port 21
[21][ftp] host: 192.168.1.103   login: charit   password: charit
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2015-09-12 18:16:04
```

- The traffic generated was also captured by our sniffer. Instead of displaying all the traffic, I used a display filter `ftp.request.command==PASS` in order to view only traffic that might be malicious. The following screenshot shows what display filter I used to query malicious repetitive packets.

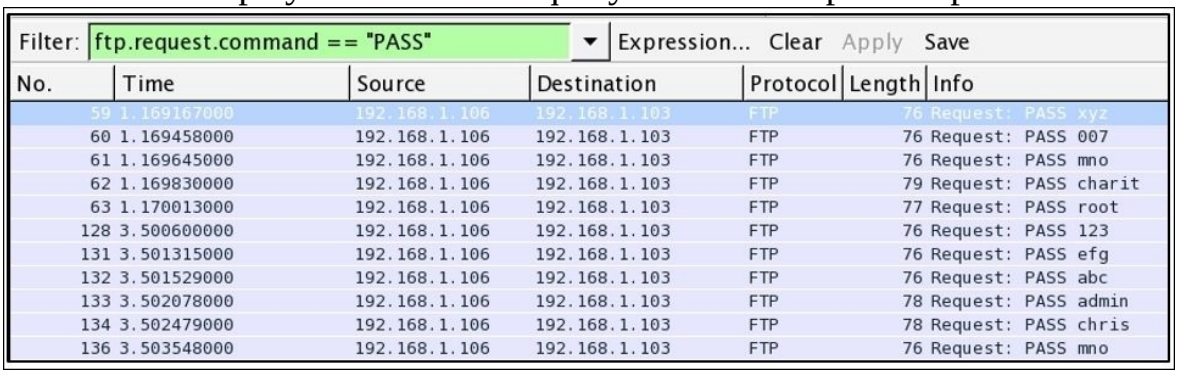


Figure 7.16: FTP Brute Force attack traffic pattern

- It is easily identifiable that the traffic is malicious because, in a span of maximum 85 seconds (calculated using the time column), there were approximately 10 password attempts made. This does look dangerous, and activities of such kind should be monitored closely in order to protect your resources facing the Internet.

There is one more way through which you can point out such traffic patterns. The best advisable option using Wireshark is to create a different coloring scheme using the same display filter expression that we used in order to point out the malicious traffic even faster. Refer to the following screenshot where I did the same and created a different coloring scheme for both TELNET and FTP

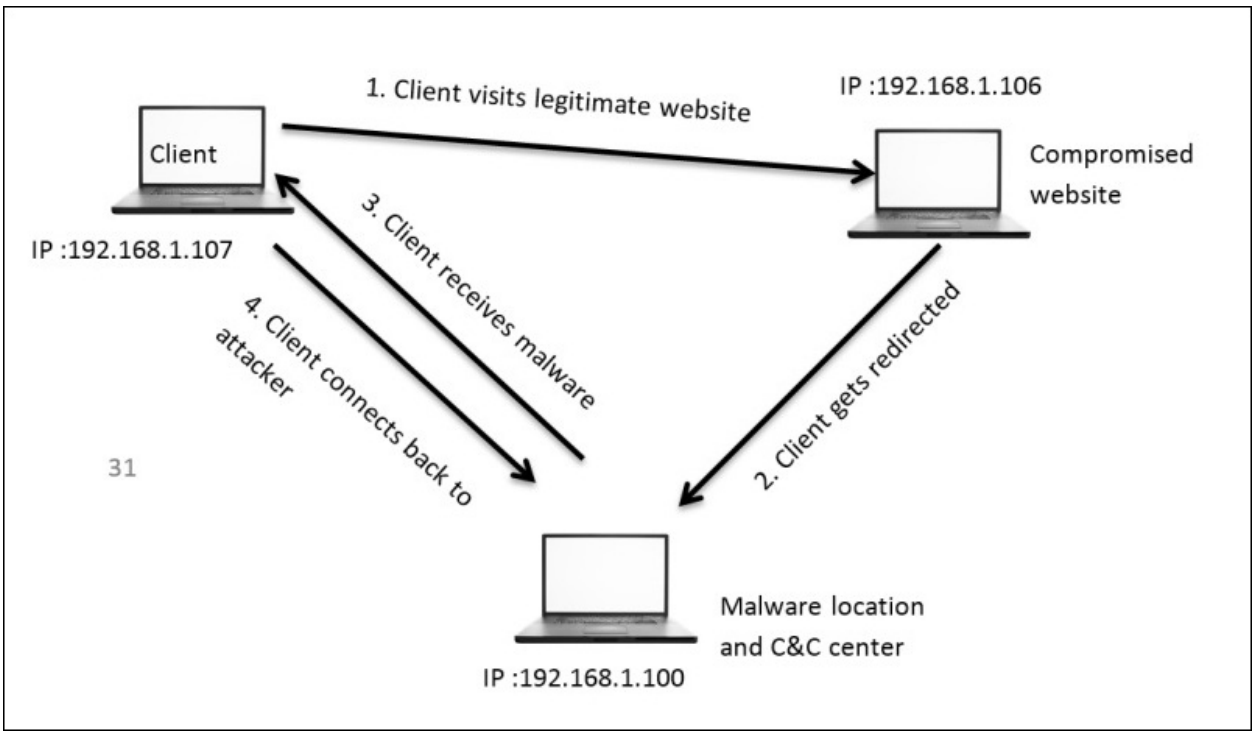
traffic:

Figure 7.17: Coloring scheme for malicious traffic

There are various other application layer protocols (HTTP, SSH, SMTP, and so on) that fall prey to these brute forcing techniques and might result in heavy losses for corporate infrastructures. In order to make these services secure, you can force encryption over the service that you are configuring and use strong password policies, such as an alphanumeric password with minimum length. You can also enforce a password change policy at a regular intervals, such as 30 days or something. Last but not least, you can make the employees aware of such activities. Any form of social engineering attacks executed against an employee can leverage the attacker to gain access to the infrastructure more easily.

Inspecting malicious traffic

In some previously mentioned topics, we have witnessed a few scenarios that generated malicious traffic. Some of the common protocols, such as HTTP, DNS, ARP, IRC, that are seen in the list pane can carry malicious traffic. So, knowing about the malware traffic analysis is definitely an important skill every network and security professional should be well versed with. In today's digital world, various advance have been made. Yet, threats including malware infection persist. Every organization should consider threats of such nature to be critical. For illustrating the threats that are caused due to various malicious traffic, I have configured a few things in my virtual lab. The traffic generated because of the activities between the client and the server would be captured in parallel, which we will use to analyze later. Refer to the following screenshot:



Malwares are supposed to perform a couple of tasks once installed on the victim's machine, such as passing on the secret content to the person in command, receiving commands from the server, and infecting and corrupting systems. Even if you have the best security solutions installed in your

infrastructure, you are still open to wide attack vectors, including malware infections.

Now, we have understood the basics of how malicious traffic is being generated, and we also have a clear image of the infrastructure that we will work with. So, without wasting even a second more, let's go ahead and start the process. Follow these steps if you want to replicate the scenario in your own virtual lab:

- You require three machines connected to the same LAN. Make sure they are able to talk to each other, that is, verify the connectivity.
- On the IP address `192.168.1.106` stays a legitimate website, which the client is habituated to visit. However, this time, the client is not aware of the infection that causes redirection to another webserver. Refer to the following screenshot of the legitimate server:

Figure 7.18: Legitimate website

- To simulate the redirection, I have configured my Apache server running on 106 to redirect every request coming to IP `192.168.1.100` and download the `efg.exe` malware from there.
- So, next time the client visits the website running at `192.168.1.106`, it gets redirected to a new webserver address, which directly asks the client to run a file named `efg.exe`. Refer to the following screenshot:

Figure 7.19: Client gets redirected to IP 192.168.1.100 and is asked to run the application.

- If the client clicks on **Run** they might not be aware of the dangerous effects the malware can pose to the client's machine and the network client is a part of. The publisher of the application is not verified, so the browser is not able to verify it. This results in giving an unknown publisher error. If the client still proceeds and clicks on **Run**, the malware will be installed. Refer to the following screenshot:

Figure 7.20: Unknown publisher error

- Now, let's suppose that, if the client hits run, then the malware will be downloaded to the client's machine. It will be executed later on, thus creating a connection back to the command and control center.
- If the connection back to the attacker was successful, then without the knowledge of the client, the attacker can copy files, delete files, take screenshots, take webcam snaps, record voice through the mic, corrupt system files, and so on. You might have heard of various malwares such as ransom wares, spywares, and adwares.
- The whole traffic generated because of all these activities is being captured. Let's take a look at it. Instead of showing you the traffic, I assembled the TCP stream first between the client and the legitimate server.
- To understand the way our malware works, we need to look at more details, which can be presented to us by Wireshark. Refer to the following screenshot that shows the assembled TCP stream:

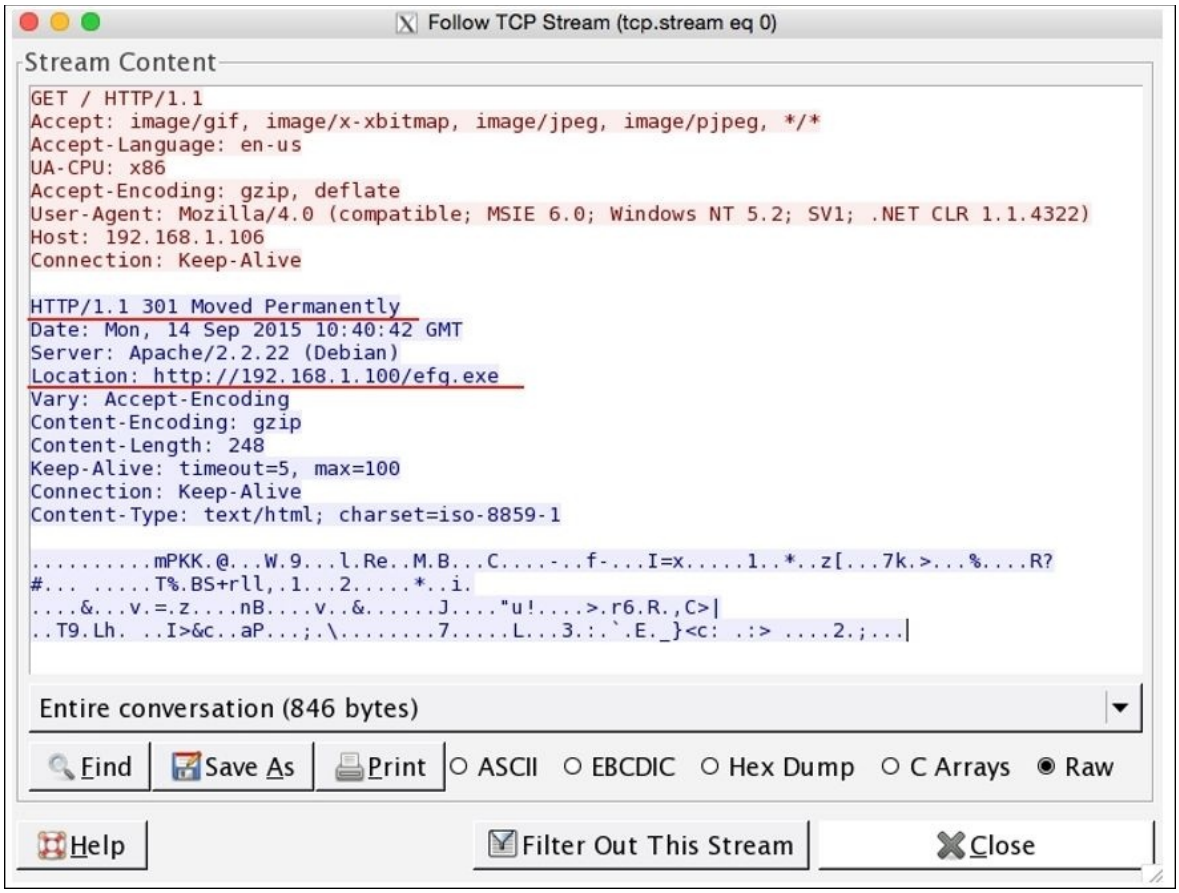


Figure 7.21: TCP stream between the client and real (compromised) server

As you can clearly see, the client is trying to visit the webserver, and the request is being forwarded with HTTP redirection to the new address `192.168.1.100`, trying to download the malicious file.

- Once the client gets a redirection response, the client again initiates a three-way handshake with the new server and tries to download the file. After a couple of packets were exchanged between the hosts in the later frames, the clients received a `200 OK` status message, suggesting successful download of the malware.

In the following screenshot, you can see that the malware signature can be easily recognized by any `IDS/IPS` in place:

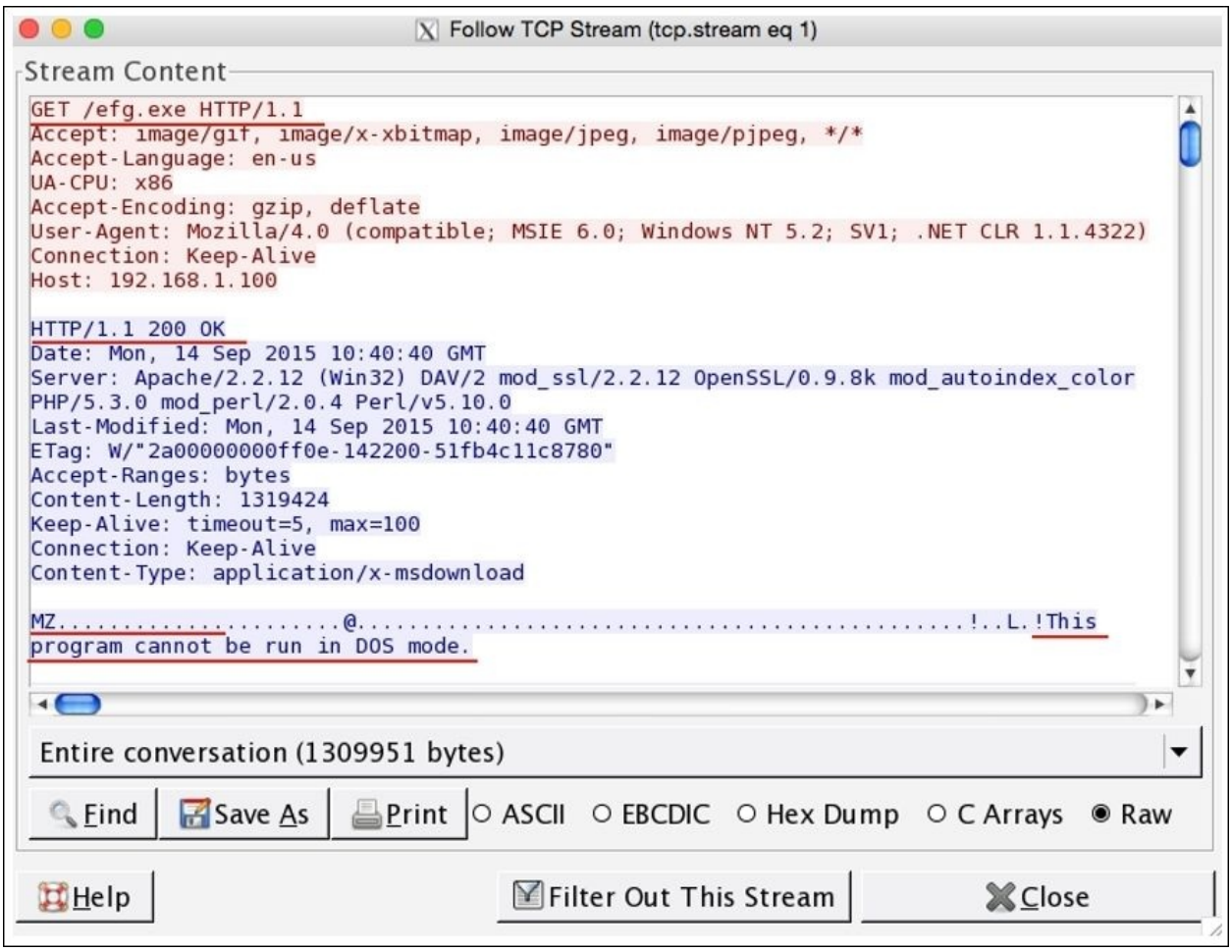
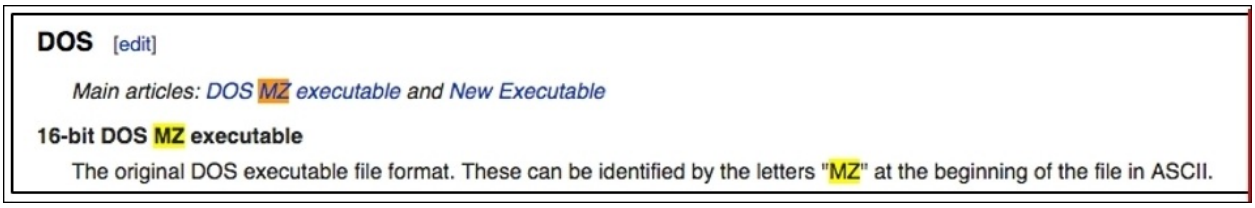


Figure 7.22: Malware signature

The `GET` request was initiated by the client in search of efg.exe, to which the server responded with a `200 OK` status message. Later, you can see the known malware signature starting with the characters `MZ` followed by some random character. A quick Google search regarding the same will reveal its behavior and pattern. Our search also reveals that it is an executable file, as Wikipedia states 16/32 bit DOS executable files can be identified by the letters `MZ` at the beginning of the file in ASCII. Refer to the following screenshot:



Until this point, its clear that the is a Windows executable file is clear which might be malicious.

Moving on with our investigation regarding the malicious file, I would like to export the `efg.exe` file using Wireshark.

1. Go to **File** | **Export Objects** | **HTTP**. You will see a dialog similar to the one shown here:

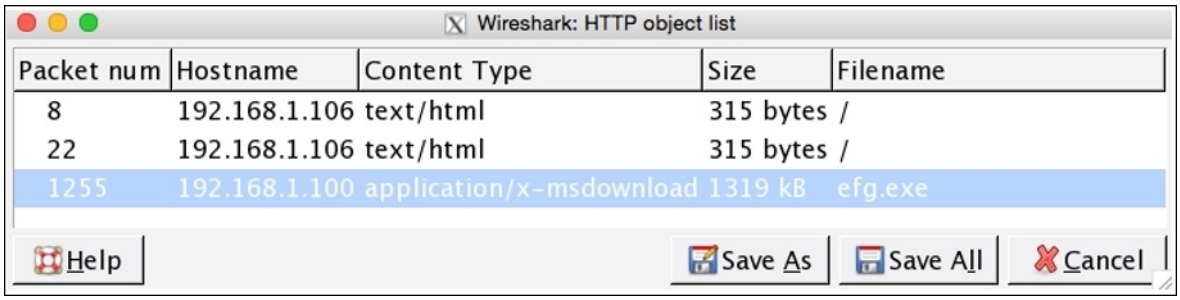


Figure 7.23: Exporting HTTP objects

2. Now, to export the file, you need to select the conversation that states the name of the file along with it. Then click on **Save As** and save the file at a location of your choice.
3. The best option would be to upload this file to websites such as http://www.virustotal.com, which will cross examine the PE-executable file with numerous antivirus software online and will show you a detailed analytical report. Refer to the following screenshot:

Figure 7.24: Uploading `efg.exe` to virustotal.com

4. Now, click on **Scan it!** to let the website examine the file and wait for the results:

Figure 7.25: `efg.exe` examination completed

31 out of 56 antivirus software detected the executable file as malicious, which is quite alarming.

5. Further, I manually examine the conversation between the infected machine and the command and control center by looking at the hex dump in the following TCP stream window. I observe something. Refer to the following screenshot:

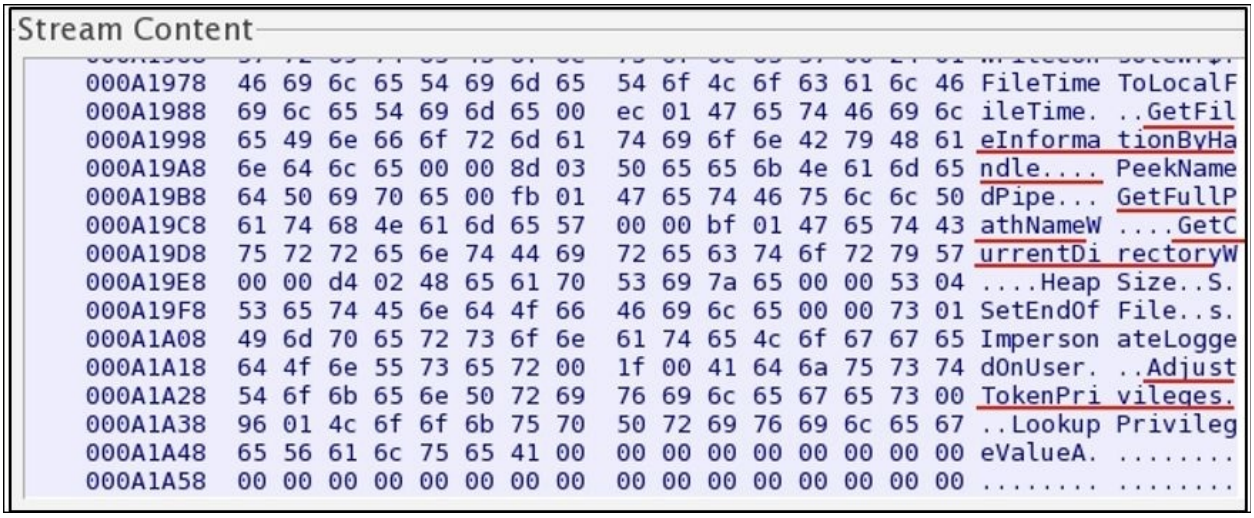
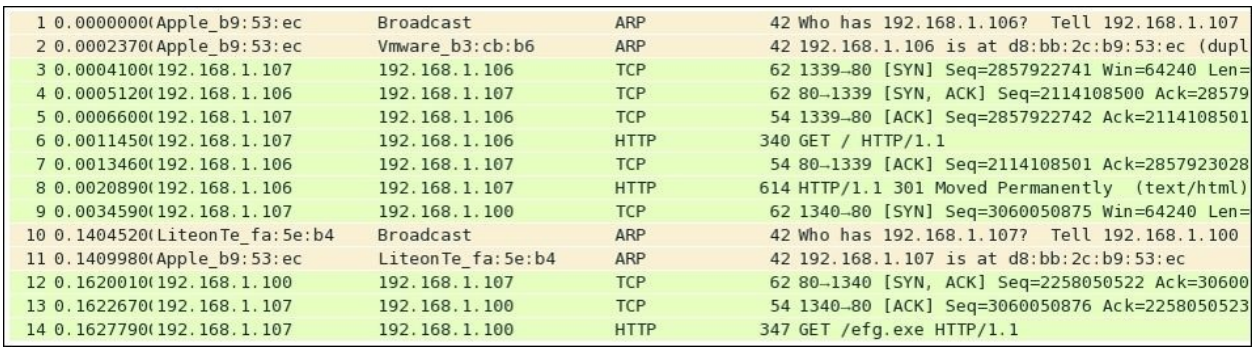


Figure 7.26: Hexdump in TCP stream dialog

It seems that the server machine that has taken the control of the victim issues some command to gather quick information regarding the machine. The highlighted content on the right-hand side of the window states strings such as `Get File Information`, `Get full PC name`, `Get Current directory`, `Adjust token Privileges`, and so on.

As per my analysis, the file that got installed to the windows box is definitely malicious. It might have caused some serious damage to the individual machine as well as the network. The best advisable solution is to isolate the machine from the network, unless it is being disinfected using specialized tools.

To conclude this section, I have one more thing to depict using the list pane in Wireshark. Refer to the following screenshot:



1	0.0000000	Apple_b9:53:ec	Broadcast	ARP	42 Who has 192.168.1.106? Tell 192.168.1.107
2	0.0002370	Apple_b9:53:ec	Vmware_b3:cb:b6	ARP	42 192.168.1.106 is at d8:bb:2c:b9:53:ec (dupl
3	0.0004100	192.168.1.107	192.168.1.106	TCP	62 1339→80 [SYN] Seq=2857922741 Win=64240 Len=
4	0.0005120	192.168.1.106	192.168.1.107	TCP	62 80→1339 [SYN, ACK] Seq=2114108500 Ack=28579
5	0.0006600	192.168.1.107	192.168.1.106	TCP	54 1339→80 [ACK] Seq=2857922742 Ack=2114108501
6	0.0011450	192.168.1.107	192.168.1.106	HTTP	340 GET / HTTP/1.1
7	0.0013460	192.168.1.106	192.168.1.107	TCP	54 80→1339 [ACK] Seq=2114108501 Ack=2857923028
8	0.0020890	192.168.1.106	192.168.1.107	HTTP	614 HTTP/1.1 301 Moved Permanently (text/html)
9	0.0034590	192.168.1.107	192.168.1.100	TCP	62 1340→80 [SYN] Seq=3060050875 Win=64240 Len=
10	0.1404520	LiteonTe_fa:5e:b4	Broadcast	ARP	42 Who has 192.168.1.107? Tell 192.168.1.100
11	0.1409980	Apple_b9:53:ec	LiteonTe_fa:5e:b4	ARP	42 192.168.1.107 is at d8:bb:2c:b9:53:ec
12	0.1620010	192.168.1.100	192.168.1.107	TCP	62 80→1340 [SYN, ACK] Seq=2258050522 Ack=30600
13	0.1622670	192.168.1.107	192.168.1.100	TCP	54 1340→80 [ACK] Seq=3060050876 Ack=2258050523
14	0.1627790	192.168.1.107	192.168.1.100	HTTP	347 GET /efg.exe HTTP/1.1

Figure 7.27: Unusual behavior noticed in list pane

Observe the behavior of the packets from the beginning, as it started with the ARP request sent by the Windows machine because it was trying to look for a legitimate web server locally configured. Followed by the three-way handshake, the client initiates a GET request in frame 6, which the server acknowledged in the following packet. Then, the server states that the resource the client is looking for has been moved to another location, and the client is required to go there. After this, the client generates an SYN request in frame 9. Then, the command and control center generates the ARP packets asking for the client's physical address in order to get in touch with it and to transfer the file. Then, at last, in frames 12 and 13, the three-way handshake is completed, which ends in generating a GET request from the victim's machine in order to start the transfer of the exploit as seen in frame 13. The consequences of such traffic patterns can be highly devastating. A good network/security admin should be aware of such traffic patterns and can use such traffic behavior to create firewall/IDS-IPS signatures that can generate quick alerts. They can help in avoiding and making their infrastructures ready to fight with these malicious traffic.

Solving real-world CTF challenges

Capturing the flag events is the most common thing that happens in security conferences. The objective is to learn and play with the challenges based on real-world scenarios that can assist you quite well in learning the methodology. Popular conferences such as DEF Con, PlaidCTF, CSAW, and Codegate can be searched for if you are interested in cracking flags. Basic programming, networking, forensics, and common sense are the skills required to take part in these challenges.

I have made a couple of challenges for you and we will be solving them as well in a step-by-step approach. I have made all of them pretty simple in order to give you an idea of how the CTF thing works and definitely the approach you are supposed to follow. So, let's begin and capture some flags.

First CTF: Leverage the weakness in remote administration services

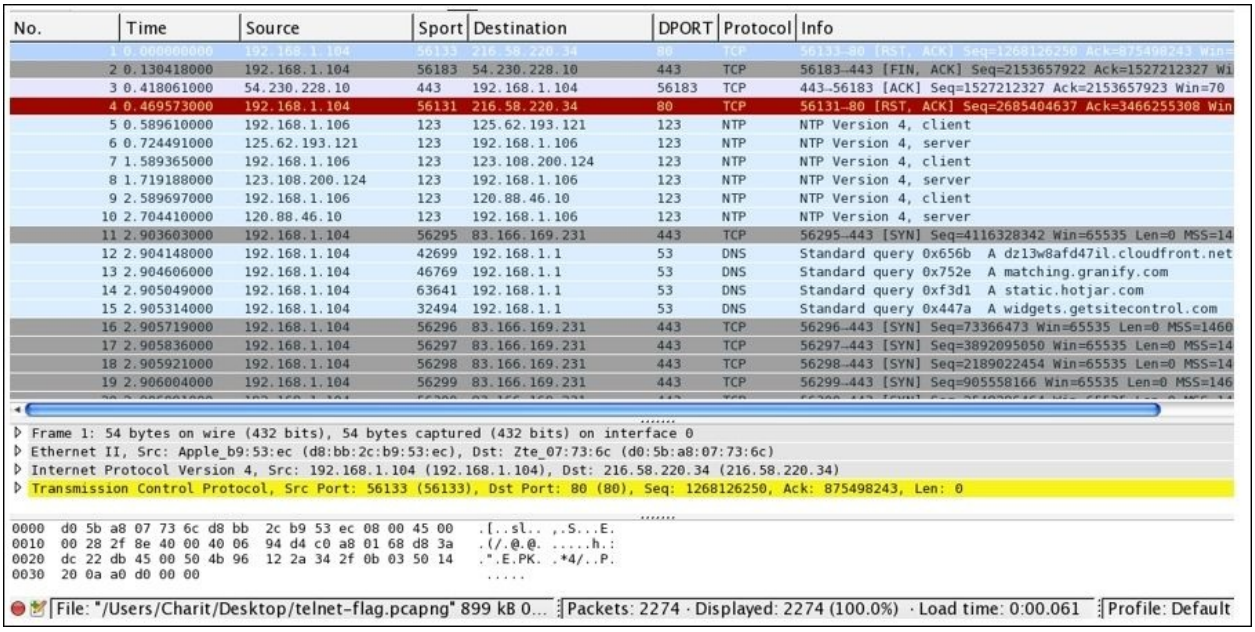
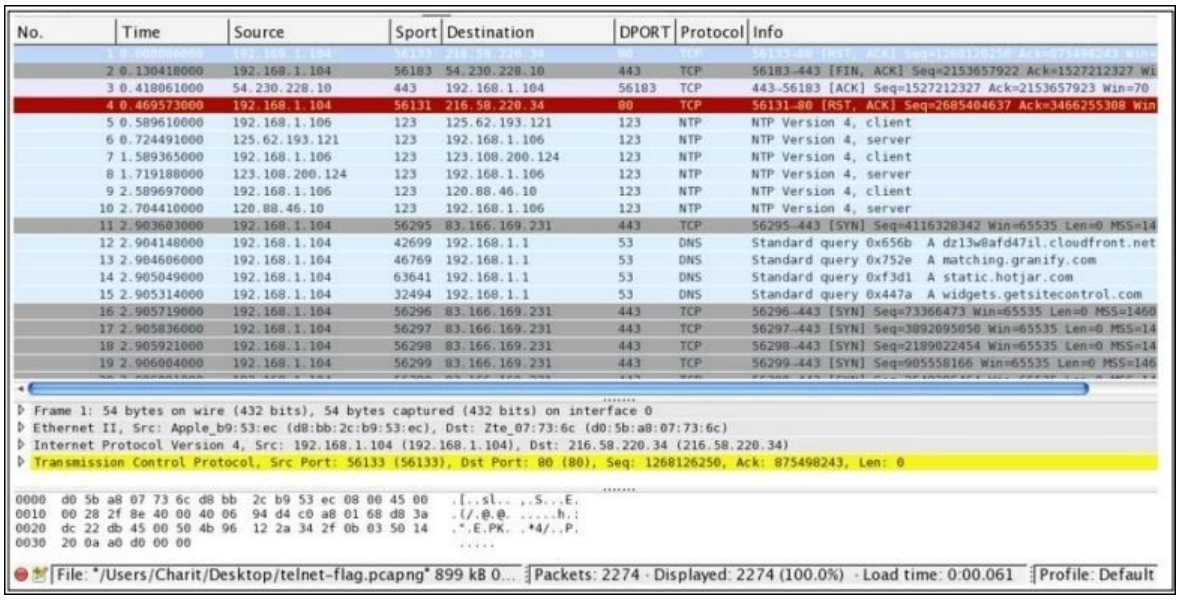


Figure 7.28: CTF1 trace file

- **Solution**: We have a `telnet-flag.pcap` file that lists multiple packets in the list pane. The question is asking us to take advantage of remote

administration services. How many services do we know which are used for remote administration RDP, Telnet, and SSH? To better understand the scenario, let's open our trace file in Wireshark first. Refer to the following screenshot:



As you can see, there are more than two thousand packets in our trace file. It would be practically impossible to scroll to the bottom to see each packet. The best option would be to look into the protocol hierarchy window, which will give us a brief regarding all protocols involved in the whole trace file. From here, it would be easy for us to identify the remote administration services. The protocol hierarchy window can be accessed from the Statistics menu. Refer to the following screenshot:

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s
Frame	100.00 %	2274	100.00 %	823058	0.146
Ethernet	100.00 %	2274	100.00 %	823058	0.146
Internet Protocol Version 4	99.65 %	2266	99.96 %	822722	0.146
Transmission Control Protocol	95.73 %	2177	97.16 %	799706	0.142
Secure Sockets Layer	27.92 %	635	42.44 %	349318	0.062
Secure Sockets Layer	0.48 %	11	0.87 %	7149	0.001
Malformed Packet	1.10 %	25	0.17 %	1375	0.000
File Transfer Protocol (FTP)	0.44 %	10	0.09 %	756	0.000
Telnet	3.43 %	78	0.73 %	6000	0.001
Hypertext Transfer Protocol	0.09 %	2	0.20 %	1636	0.000
User Datagram Protocol	3.78 %	86	2.75 %	22656	0.004
Network Time Protocol	0.26 %	6	0.07 %	540	0.000
Domain Name Service	2.95 %	67	2.43 %	19972	0.004
NetBIOS Name Service	0.31 %	7	0.09 %	716	0.000
NetBIOS Datagram Service	0.26 %	6	0.17 %	1428	0.000
SMB (Server Message Block Protocol)	0.26 %	6	0.17 %	1428	0.000
SMB MailSlot Protocol	0.26 %	6	0.17 %	1428	0.000
Microsoft Windows Browser Protocol	0.26 %	6	0.17 %	1428	0.000
Internet Control Message Protocol	0.13 %	3	0.04 %	360	0.000
Address Resolution Protocol	0.31 %	7	0.04 %	294	0.000
Text item	0.04 %	1	0.01 %	42	0.000
Address Resolution Protocol	0.04 %	1	0.01 %	42	0.000

Figure 7.29: Protocol hierarchy CTF1

Among all the protocols listed, I can see only one that is used for remote administration, and we can use it to move on with our CTF process. So, I applied the display filter `telnet` in order to see only relevant traffic. Refer to the following screenshot:

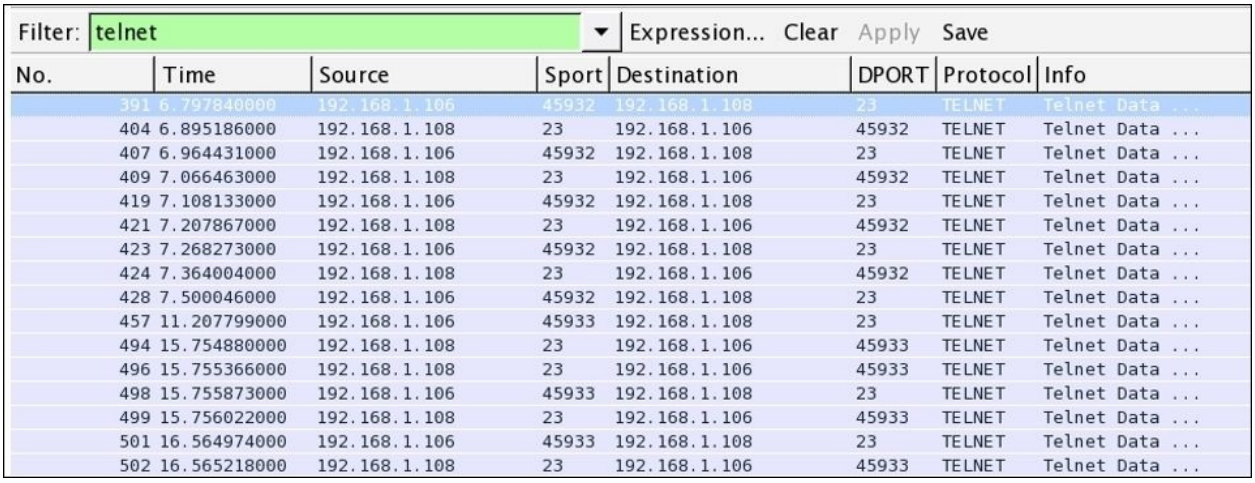


Figure 7.30: Telnet traffic CTF1

Now, the next step would be to follow the TCP stream of these packets, which will reveal more information regarding the Telnet session.

This is what the question was about: leveraging the weakness in a remote administration service. Telnet sessions can be viewed in plain text format, and we finally leveraged the weakness to take advantage of viewing the session's information in plain text format. The flag is the password used by the user to log in to the Windows machine to perform maintenance activities.

FLAG : Sup3rs3cr3t

The following screenshot illustrates how the TCP stream windows will look after the packets are assembled. Also, the Telnet session's password can be seen clearly.

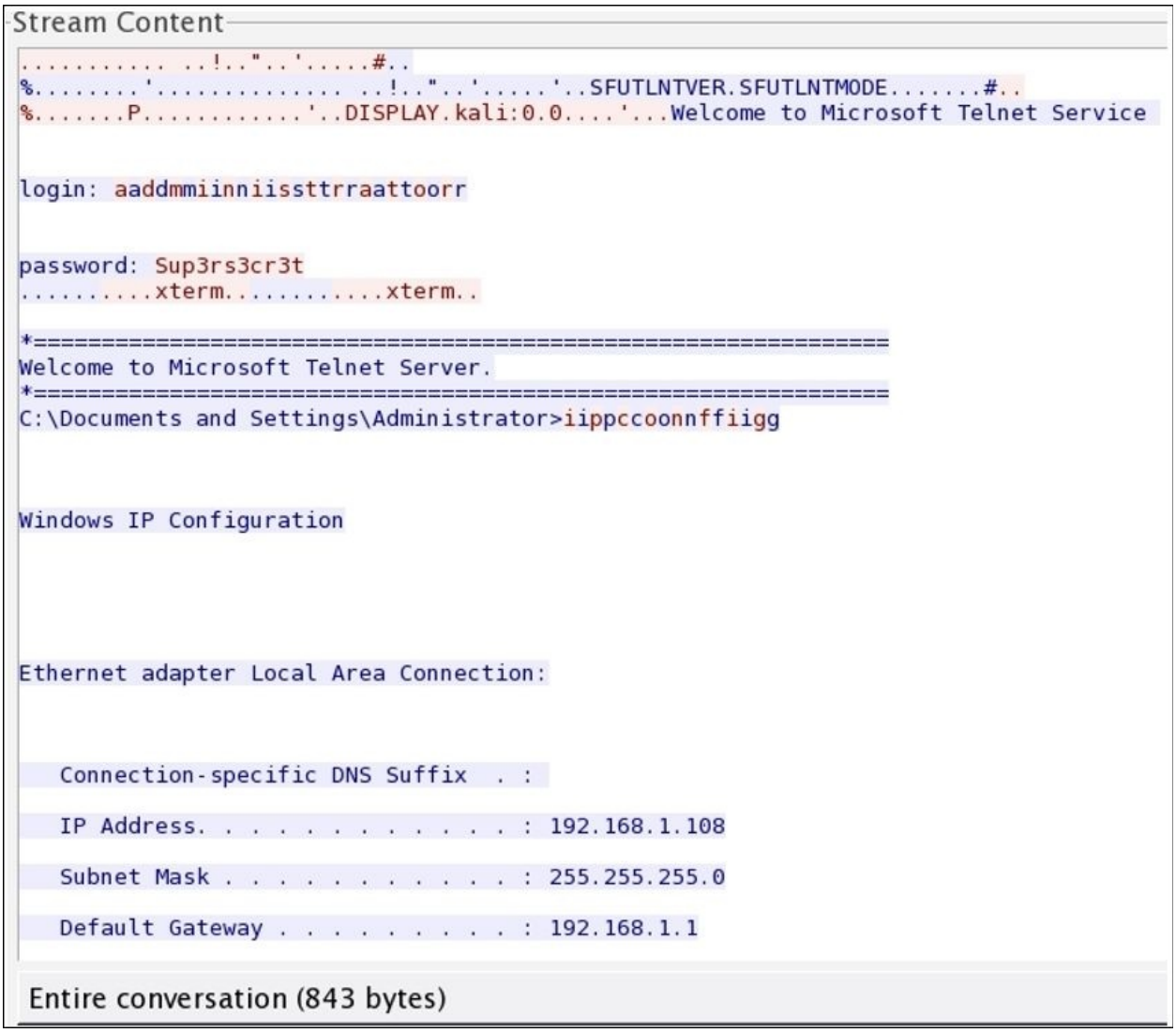


Figure 7.31: TCP stream dialog CTF1

I hope you have understood the basic approach of CTF solving. We would follow similar approach in solving further CTF challenges.

This time I have designed a CTF that utilizes another common protocol and will let you learn the basics of the CTF challenge approach.

Second CTF: Image magic

Solution is in the title of this CTF and it is pretty small and attractive, though we

have no idea what we are looking for, but for sure there is something related to images. Wireshark performs magic every time; this is what my perspective tells me about the challenge.

Following an approach similar to the one we talked about first, we would open the trace file in order to learn basic stats related to the traffic capture that will give us an overview of the protocols used during the session. Refer to the following screenshot:

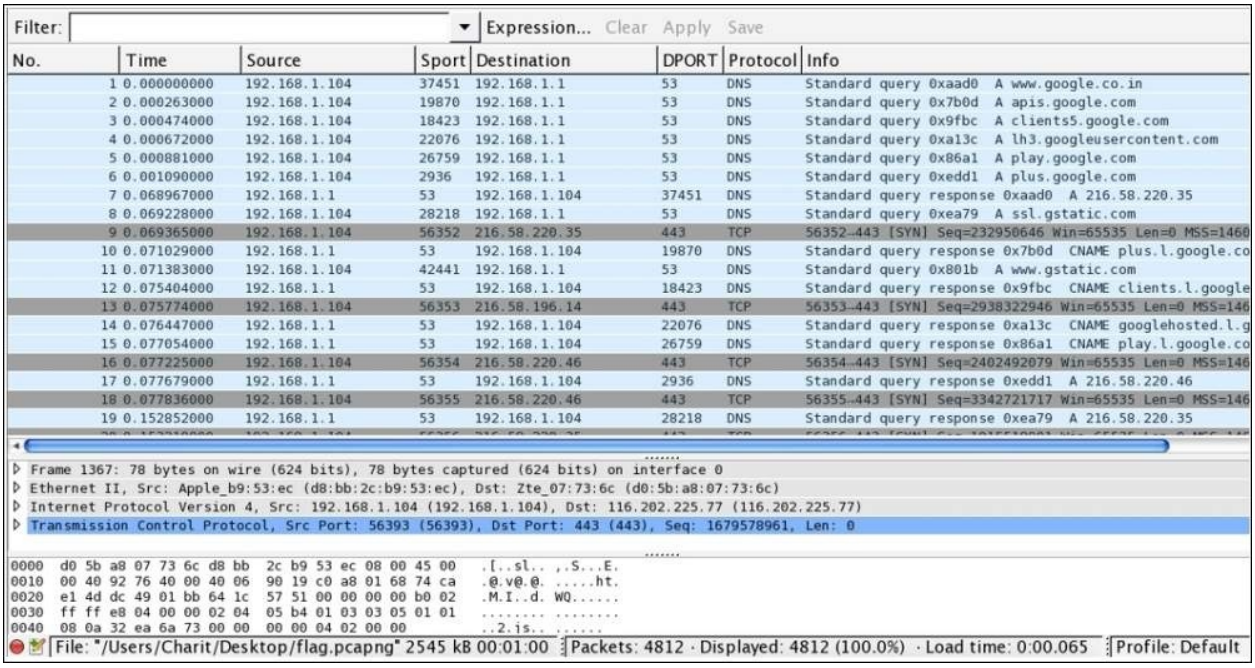


Figure 7.32: Trace file CTF2

The trace file starts with a lot of DNS packets, which don't look very useful for our analysis. Looking at the following status bar in Wireshark, we can say that there are around 4,800 frames definitely captured. This one is not something that we can inspect element by element, so we need the help of our best guy: protocol hierarchy dialog (now I hope, without any specific instruction, that you can open the dialog):

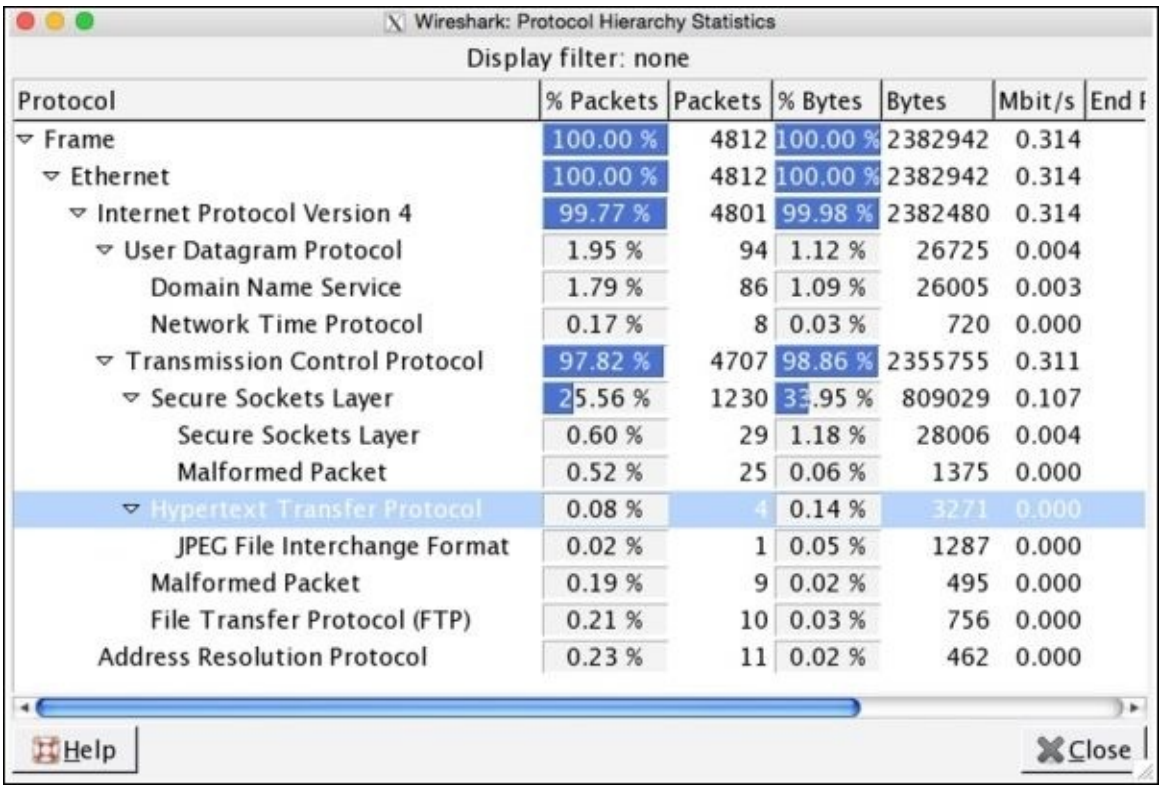


Figure 7.33: Protocol hierarchy CTF2

In the list of various protocols, I spotted JPEG, which is an image extension, and is listed under the HTTP section in the dialog. We can conclude from this that there is some relation between these two ,so our display filter could become HTTP, which will keep us moving in the right direction.

As soon as I type HTTP in the display filter box and press enter, I am presented with just four packets. One of those listed is a `.jpg` file with the name flag. Refer to the following screenshot:

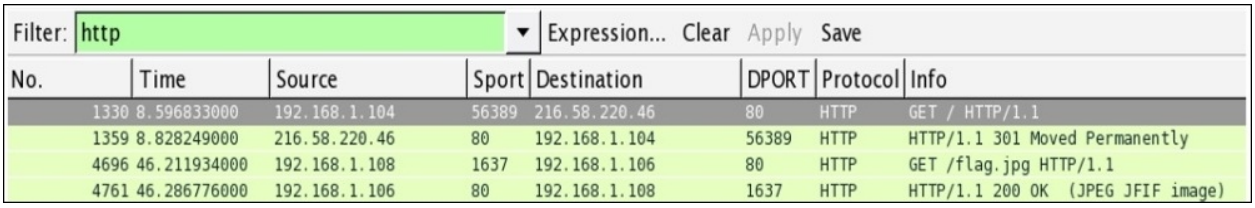


Figure 7.34: Display filter HTTP—CTF2

Frame number 4,696 lists a `GET` request for a `alg.jpg` file. Investigating, further by looking at the TCP stream of this packet, confirms that there was a `.jpg` file requested by the client at `192.168.1.108`. Refer to the following screenshot:

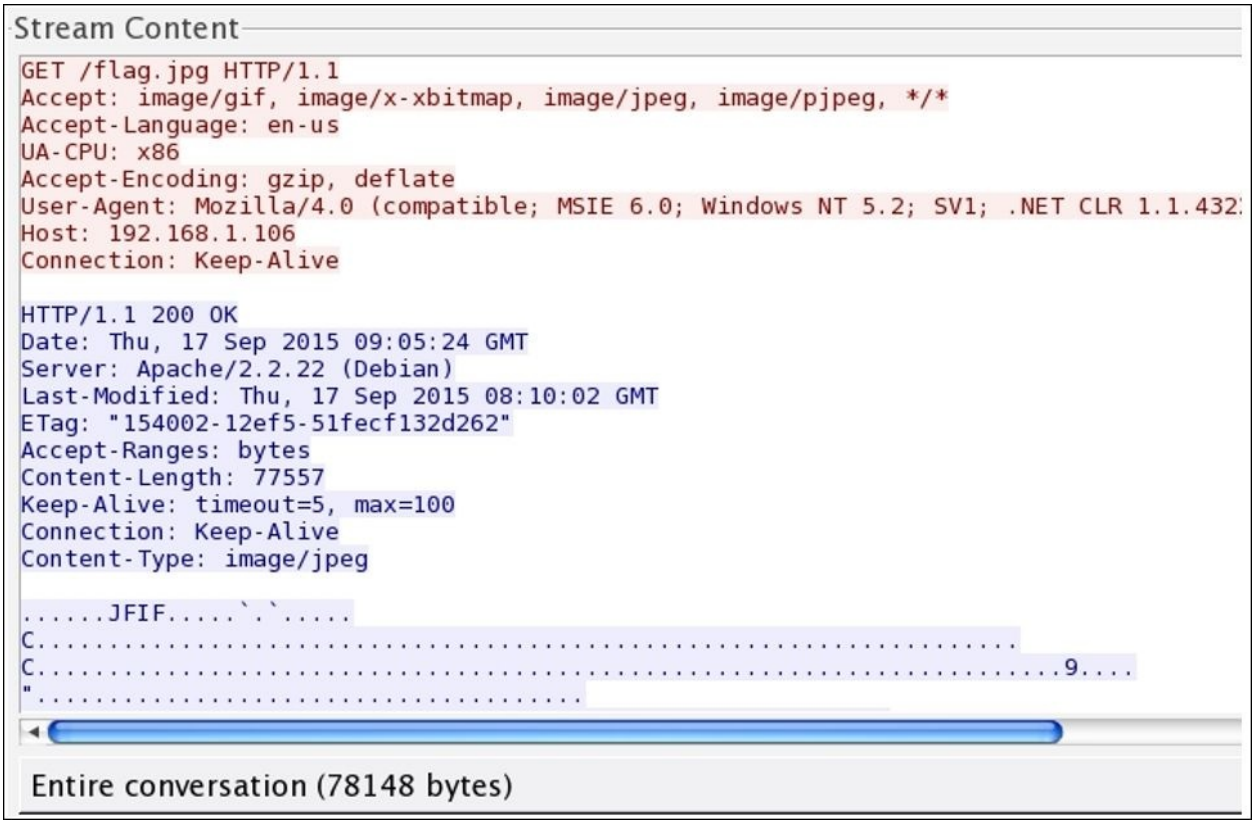
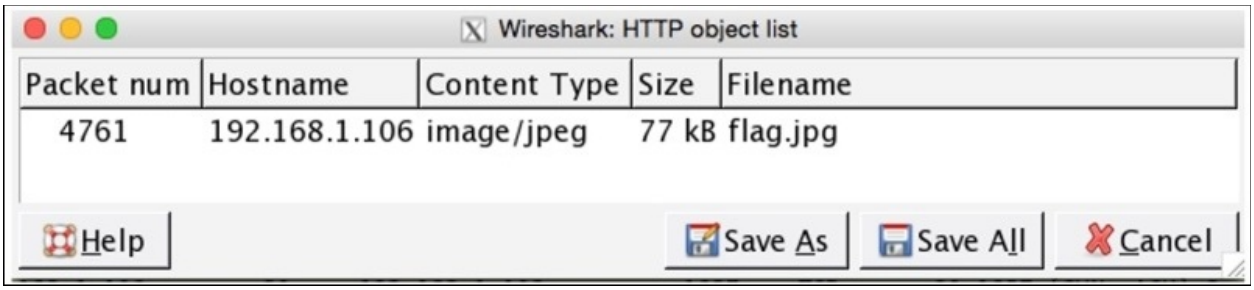


Figure 7.35: TCP stream—CTF2

The request made by the client is now confirmed and verified. The next step would be to export this object from the stream. Go to **File | Export Objets | HTTP**.



The window just lists one `flag.jpg` file. Follow the mentioned steps in order to export the image object. First select the row one showing the images object then click on **save as** and save the file at any desired location. When finished, open the file to view the flag content. Refer to the following screenshot to see the content of the exported object.

Figure 7.36: CTF2

This challenge was pretty interesting, because you learned about a different idea behind CTF challenges.

Our final challenge also introduces us to a new idea behind CTF's.

Third CTF: Are you Pro Enough!!

Title of the challenge is pretty challenging in itself. However, we will solve this together. So, let's open the trace file first.

At first glance, it looks like other trace files we have seen with numerous useless packets filled in. Without getting ourselves confused with the overwhelming amount of information there, let's follow the approach that we have been following so far. Refer to the following screenshot:

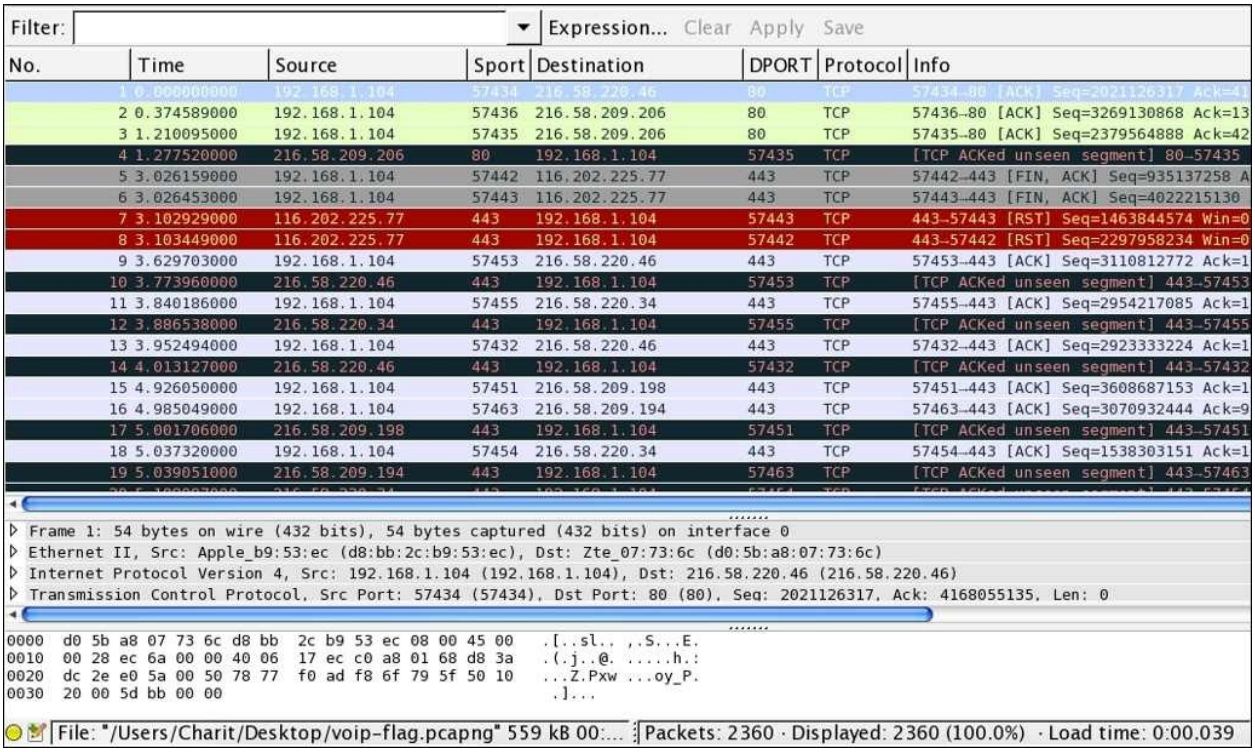


Figure 7.37: Packet list pane—CTF3

Look at the protocol hierarchy window that can help us in revealing more about the CTF challenge we are dealing with. Refer to the following screenshot:

Protocol	% Packets	Packets	% Bytes	Bytes
Frame	100.00 %	2360	100.00 %	479211
Ethernet	100.00 %	2360	100.00 %	479211
Internet Protocol Version 4	98.43 %	2323	99.40 %	476348
Transmission Control Protocol	9.03 %	213	5.35 %	25623
Secure Sockets Layer	1.40 %	33	2.47 %	11835
Secure Sockets Layer	0.04 %	1	0.27 %	1315
NetBIOS Session Service	0.42 %	10	0.40 %	1913
SMB (Server Message Block Protocol)	0.42 %	10	0.40 %	1913
Hypertext Transfer Protocol	0.08 %	2	0.19 %	903
Media Type	0.04 %	1	0.10 %	463
User Datagram Protocol	89.32 %	2108	94.03 %	450617
Session Initiation Protocol	0.76 %	18	2.13 %	10210
Domain Name Service	1.06 %	25	0.75 %	3586
Network Time Protocol	0.34 %	8	0.15 %	720
Data	28.86 %	681	30.17 %	144566
Real-Time Transport Protocol	56.86 %	1342	59.91 %	287096
Real-time Transport Control Protocol	0.30 %	7	0.17 %	810
Real-time Transport Control Protocol	0.30 %	7	0.17 %	810
Real-time Transport Control Protocol	0.04 %	1	0.02 %	110

Figure 7.38: Protocol hierarchy—CTF3

As expected, we get a new insight about the trace file, and we can observe that the UDP traffic percentage is about 89 percent, which is quite a big number. It lists Real Time Protocol under it. So, let's go ahead and create a display filter for RTP traffic, which can take us to the next step in solving the riddle. Refer to the following screenshot:

Figure 7.39: RTP display filter—CTF3

It seems like a call session is in progress between the two hosts at `192.168.1.107` and `192.168.1.105`. Next, using the playback feature in Wireshark, I will reassemble the stream and will try to play back. Go to **Telephony menu** | **VoIP Calls** and select the SIP call in row 1 and click on **Player**. Refer to the following screenshot:

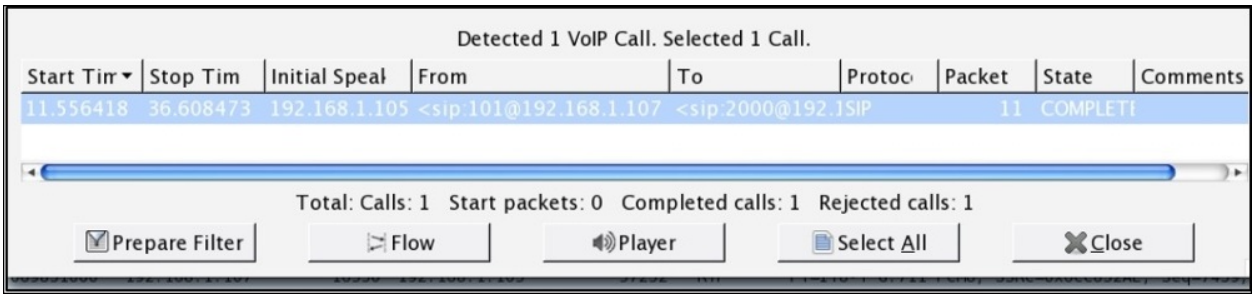


Figure 7.40: VoIP calls dialog—CTF3

Once the call session is visible, select it and click on the player where you will be asked to give the jitter value. Specify `200` as the value and click on **Decode**:

Now, you should be able to see the assembled VoIP stream available for playback. Select the first part of the communication and click on Play. The person communicating from Side A side says, *Start the transfer of the rabbit* and playing Side B's part we can observe that it is just an echo of Side's A message. Refer to the following screenshot:

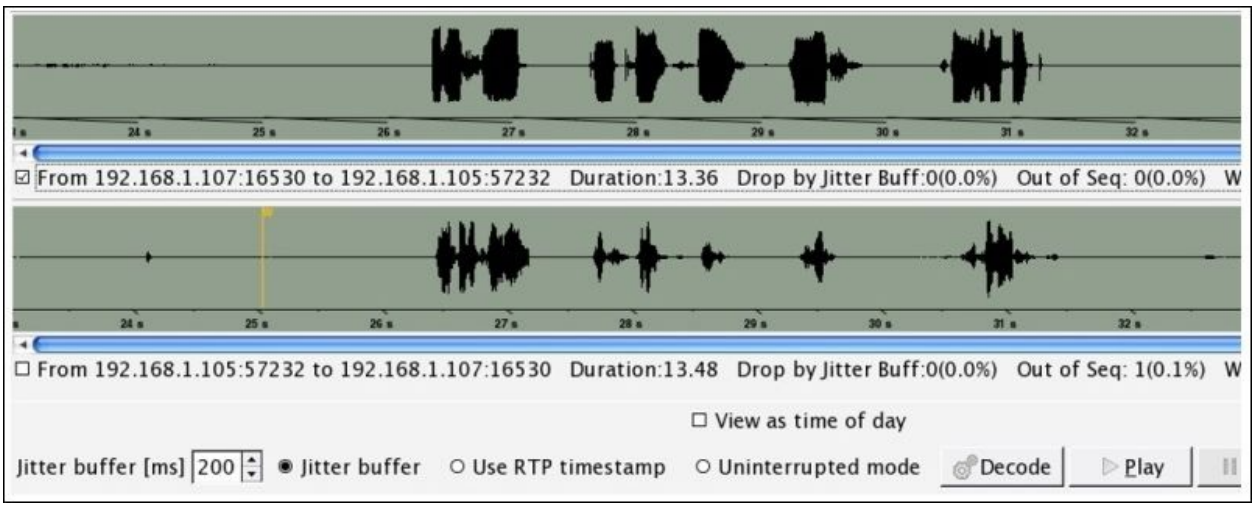
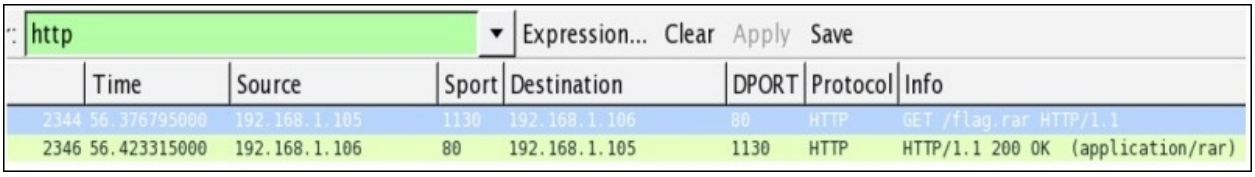


Figure 7.41: Reassembled VoIP call for playback—CTF3

We did not get many clues from this message. Let's look at the protocol hierarchy dialog once again and see what we have in the TCP section. Other than the HTTP protocol, there isn't much useful information. Under the HTTP tree, there is a media type, which means something got transferred between the hosts on the network (as the person on VOIP call said `start the transfer`). We applied HTTP as a display filter, we got the following screenshot:

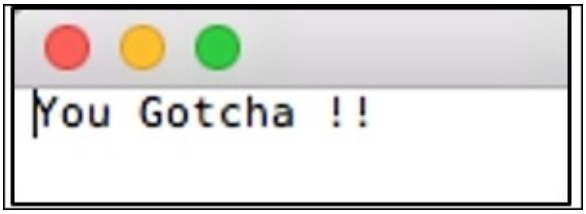


As is clearly visible, a `flag.rar` file got transferred. Let's export this to a `.rar` file for extraction. Go to **File** | **Export Objects** | **HTTP**, select the first row, and click on **Save as** to save the `.rar` file. The file got successfully saved, but when we tried opening the file, it asked for a password, which we don't know have:

Figure 7.42: `Flag.rar` ask password

Did you notice what the person said over the call "start the transfer of the rabbit", so why don't we check `therabbit` as password to this archive file.

Luckily, our first guess worked. This might not happen every time we solve CTF challenges. There is a file inside it called `flag.txt` that reads **You Gotcha!!** Refer to the following screenshot:



This section was particularly real fun! I enjoyed solving it for you. I hope the approach and flow we followed would prove useful for other CTFs that you might start solving after reading this chapter. Best of luck to you for your independent analysis, and remember that using out-of-the-box thinking and a bit of common sense is also required.

Summary

Use Wireshark to keep your network secure by defending against the most common form of infiltration attempts. Analyzing the packets with security perspective will give you a new insight into how to deal with malicious users.

Activities such as port scanning, footprinting, and various active information-gathering attempts are the basis of attacking methodologies that can be taken advantage of to bypass your security infrastructure.

Guessing passwords for a legitimate service is called a brute force attack. If the same form of attack is combined with dictionaries, which consist of millions of passwords, the chances to break in get higher. Through Wireshark, you can view such attempts made against a service in your network.

Using a legitimate looking piece of software, a malicious user can gain entry into your network. These days, the most common form through which malwares are being distributed is emails. Another attack form, such as phishing, when combined with malwares, becomes seriously dangerous.

Wireshark can help you in analyzing malware behaviors, and using the behavior analyzed, you would be able to create the necessary signatures for your IDS/IPS firewalls in place.

Capture the flag events are commonly conducted at security conferences. Multiple educational exercises are provided to the participants to experience real-world scenarios. The real CTF is where a TEAM A tries to penetrate into TEAM B's network and vice versa at the same time. Both the teams are responsible for securing against the malicious attacks sent in. There are multiple categories in CTF events, such as reverse engineering, protocol analysis, programming, cryptanalysis, and so on. Mastering Wireshark can ease your way while dealing with protocol analysis related CTFs.

Observing things scattered around with a security professional's perspective will let you see things differently. From a person inside the corporate infrastructure, things might feel OK. However, from outside, you might be very vulnerable. Security professionals are like immunity to the IT industry, and analyzing the

packets using Wireshark is one of their weapons in the arsenal.

Practice questions

Q.1 What is the difference between the active and passive information gathering techniques?

Q.2 Which information-gathering technique is stealthier and why?

Q.3 What do you understand by the term banner grabbing?

Q.4 Use the netcat utility in Linux to connect to a running HTTP service.

Q.5 What is the difference between the `-sT` and `-sS` switches used in `nmap` scans? Can you use both at the same time?

Q.6 Use `nmap` to perform OS fingerprinting on a machine and then redirect the output of the scan to a file for later use.

Q.7 Without using `nmap`, can you fingerprint an OS using Wireshark?

Q.8 How OS fingerprinting attempts made against you can lead to serious damage?

Q.9 Figure out the techniques to evade firewalls deployed in corporate environments using `nmap`.

Q.9 Is it possible to combine two attacking methodologies, ARP spoofing and DNS poisoning, in order to achieve bigger and better results?

Q.10 Try brute forcing a service in you lab environment and analyze the traffic pattern using your own custom-made dictionary files.

Q.11 Try leaning about brute forcing tools already installed in Kali Linux and figure out which tool is more suitable for RDP brute force attacks.

Q.12 What other filter expression can be useful while analyzing the malicious FTP traffic patterns?

Q.13 Is it possible to force encryption over the FTP session so that the following

TCP stream won't show the traffic in normal text form?

Q.14 Why is it important to isolate an infected PC that emits unusual traffic from your network, and what traffic patterns related to it make it malicious?

Q.15 Visit various online CTF challenge websites and try solving a few of them. Do you still find it difficult to understand the challenge, or does it seem a bit easier now?

Chapter 8. Troubleshooting

This chapter will teach you how to configure and use Wireshark to perform network troubleshooting. You will also master the art of troubleshooting network issues using Wireshark. The following are the topics that we will cover in this chapter:

- Using Wireshark to troubleshoot slow Internet issues
- Lab up
- Troubleshooting network latencies
- Lab up
- Troubleshooting bottleneck issues
- Lab up
- Troubleshooting application-based issues
- Lab up
- Practice questions

The loss of packets during transmissions is one of the most common problems that all network administrators deal with in their day-to-day lives. However, thankfully, we have various built-in error recovery features in the transmission protocol that come to our rescue to deal with the problems. However, it is essential to understand how these error recovery features work in order to troubleshoot the problems by just looking at the packets flow in the list pane if and when human intelligence is required. Troubleshooting latencies or any application-based issues in your network requires you to have an understanding of the traffic flow and the way packets interact with each other. Before we start getting our hands dirty with a troublesome network, we need to understand some basics of the recovery features that would help you diagnose and figure out the root of such problems. Consider yourself blessed that you have the privilege of using Wireshark—the most popular and well-versed tool for network packet analysis—which is an open source tool. This won't state the problems for you, but the time required to troubleshoot network-related issues is drastically reduced.

Now, you might feel like asking the question: "how does it looks like or how you can identify such happenings?" Just as every coin has two sides, the network communication has two ends: a sender and a receiver. On the sender side,

recovery features are handled by the **Retransmission Timeout** (**RTO**) values, which are a sum of **Round Trip Time** (**RTT**) and mean of standard deviation. On the receiver side, recovery mechanism is handled by keeping a track of SEQ and ACK values that are shared between the communicating hosts.

You definitely have heard about flow control features, we discussed the same in previous chapters while dissecting TCP-based communications. Flow control features are used in order to keep the transmission more reliable by taking help of dynamic functionalities such as sliding window and zero window notifications. Now that you have the basic understanding of, I want you to understand things in detail. Note that we will talk about TCP-based communication most of the time in this chapter.

Recovery features

TCP retransmissions and duplicate ACKs are the tactics that are used while recovering from a failed packet transmission or an out-of-order packets transmission scenario. Commonly, network latencies (the total time it takes for a packet to be sent along with the time its ACK is received) are observed, due to which the performance of networks are significantly disturbed. When the amount of retransmissions and duplicate ACK packets are seen very often in the list pane, most probably, there is a chance that your network is facing high latencies; if not, then just sit back and relax. My point is that you should be concerned about such activities, and if possible, mix some network management techniques with your protocol analysis that can keep you updated all the time with what's happening inside

The devices use TCP retransmission in order to send data reliably. Values such as RTT and RTO are maintained by the sender of the data in order to facilitate a reliable form of communication. The sender initiates the retransmission timer as soon as the packet leaves the ACK, and when the same is received, the sender stops the retransmission timer. The timer value here determines the timeout value. Now, if the sender does not receive the ACK, after a certain amount of time, the sender initializes the retransmission of the same packet. If the sender still does not receive any ACK, the timeout value will be doubled and the sender will retransmit the same packet again. The same cycle is followed until the ACK is received or the sender reaches maximum retransmission attempts. The sender, based on the operating system maintains a number of retransmission attempts, which are triggered when a certain timeout value is reached.

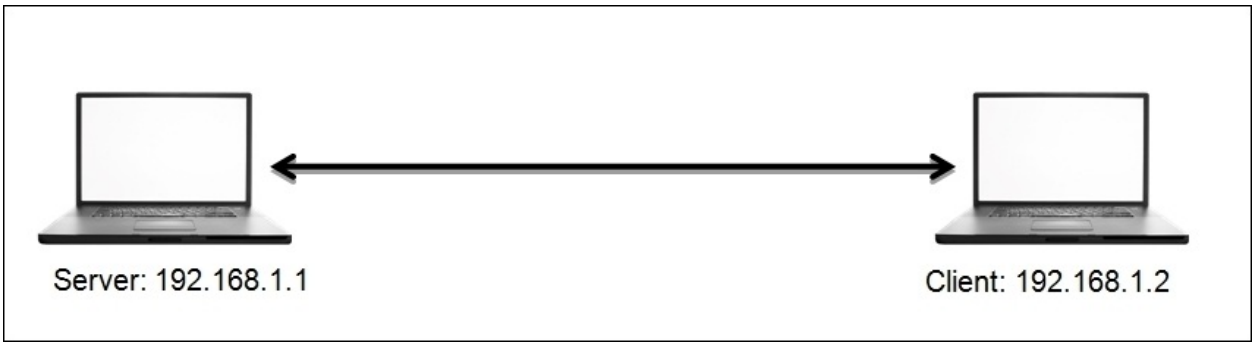


Figure 8.1: TCP duplicate ACK and retransmission

For instance, in the preceding figure, a client is located at `192.168.1.2` and the server is located at `192.168.1.1`. Here, the client is requesting some resource that the server holds, following which the transmission between the two hosts starts after the three-way handshake is successfully completed. For every data packet received, the client sends a `ACK` for the same. Now, suppose that for some random packet in the stream, the server did not receive the ACK even after the timeout value for the data packet expired. The server initiates the retransmission of the similar data packet again. The same process is followed unless and until the server receives an `ACK` for every packet, or the server at `192.168.1.1` reaches the maximum number of default attempts, five, in a row. Refer to the following figure that shows this retransmission process:

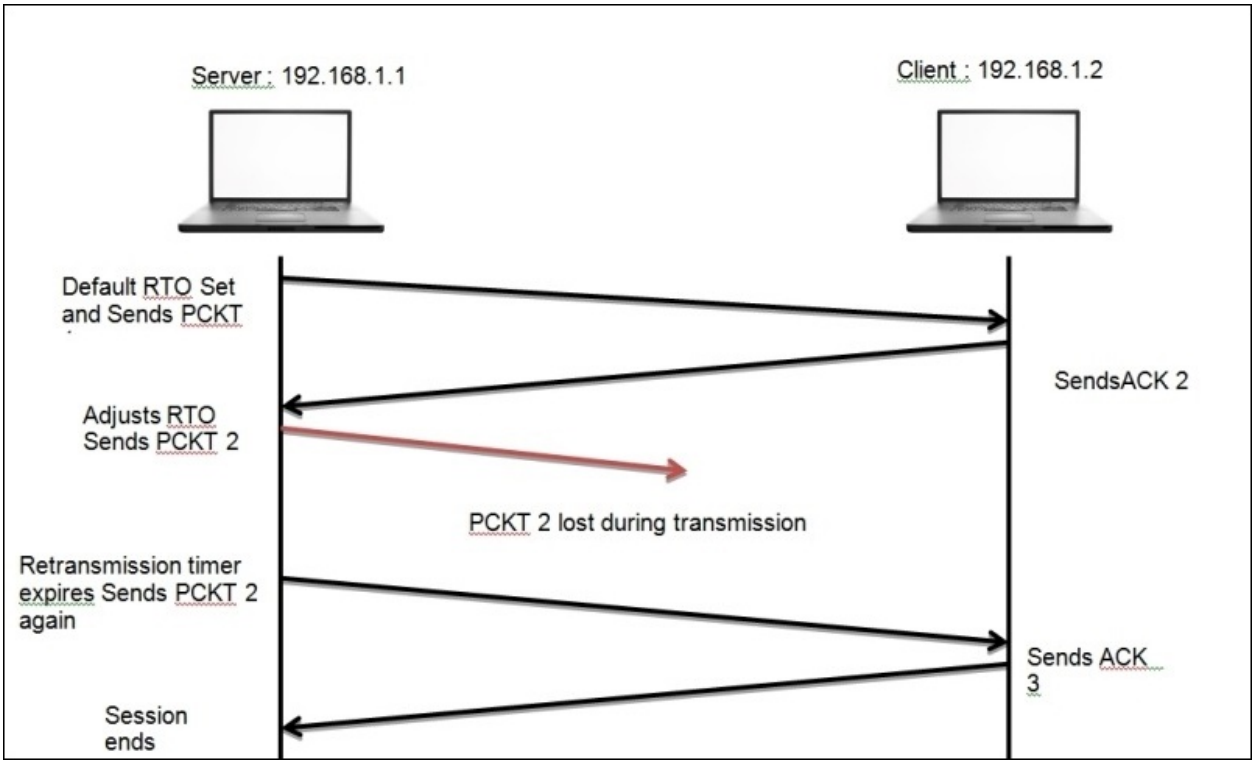
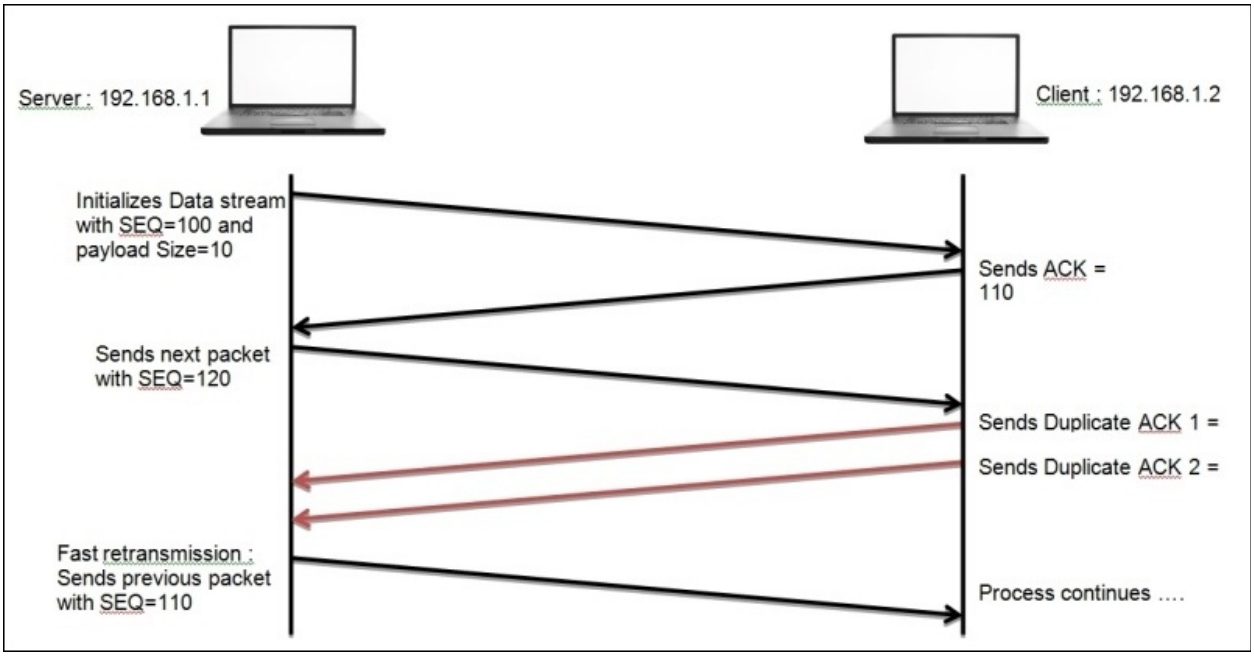


Figure 8.2: TCP retransmission

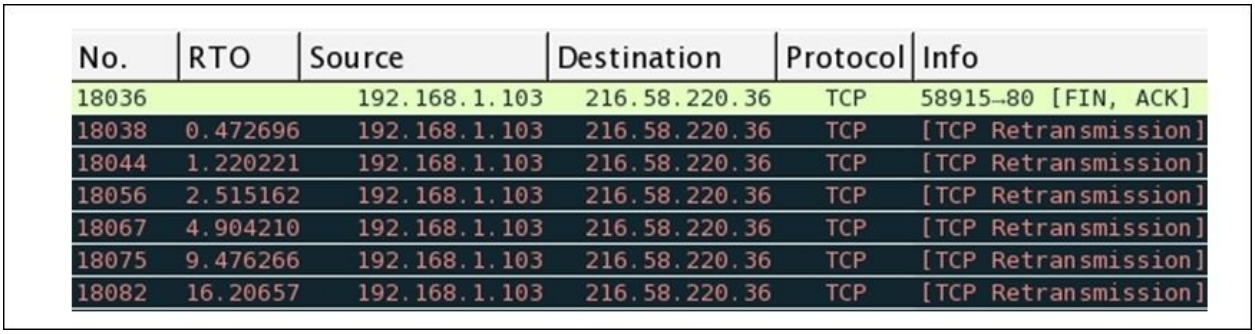
On the basis of the preceding simplified scenario, I suppose now that you have understood the gist of the retransmission process.

Now, we will discuss duplicate `ACKs` and fast retransmission, which is another recovery feature that the clients take care of. In the previous chapter, we discussed the `SEQ` and `ACK` numbers that are used in order to keep track of TCP-based communication. You might also remember how the `ACK` values were incremented using the data payload size, where we added the received packet `SEQ` value and data payload size value and the resulting sum became the `ACK` value. We sent this value with our `ACK` packet, and we expect to receive the next data packet marked with the same `SEQ` value. Suppose that the server starts sending data packets, and the first data packet is marked with a `SEQ` value of `100` with a data payload size equals `10`. Once the client receives the `ACK` packet, it prepares to send to the server with value set to 110 (remember the formula: *SEQ number received + Data payload size = ACK value*).

As soon as the server receives the `ACK` packet with the value `110`, it prepares for another data packet to be sent with `SEQ 110` with a payload size of `10`. After receiving this, the client will respond with `ACK 120`. The same process goes on till the end of the session. Now, suppose that instead of sending the next packet with `SEQ` set to `10`, the server sends a packet with `SEQ 130`, which is out of order, and after receiving this, the client would send a duplicate `ACK` set to `120` to the server to recheck and send the missing packet again from the data stream.



From the preceding scenario, I hope you have understood the process of duplicate ACKs and fast retransmission, which you can use while troubleshooting your realtime network for related anomalies. Before we go ahead and discuss flow control, I would like you to see real packets in my network that are related to both cases of error recovery that we discussed. Refer to the following *Figure 8.3* and *Figure 8.4*:



No.	RTO	Source	Destination	Protocol	Info
18036		192.168.1.103	216.58.220.36	TCP	58915→80 [FIN, ACK]
18038	0.472696	192.168.1.103	216.58.220.36	TCP	[TCP Retransmission]
18044	1.220221	192.168.1.103	216.58.220.36	TCP	[TCP Retransmission]
18056	2.515162	192.168.1.103	216.58.220.36	TCP	[TCP Retransmission]
18067	4.904210	192.168.1.103	216.58.220.36	TCP	[TCP Retransmission]
18075	9.476266	192.168.1.103	216.58.220.36	TCP	[TCP Retransmission]
18082	16.20657	192.168.1.103	216.58.220.36	TCP	[TCP Retransmission]

Figure 8.3: TCP retransmission packets

In the preceding screenshot, a client located at `192.168.1.103` sends `FIN` and `ACK` to the server at `216.58.220.36`. After this, the client would expect to receive a `ACK` packet in the next place. However, the client does not receive anything back from the server. Now, after the RTO time expires, the client starts sending the same packet after double the time, and the process of sending `TCP` retransmission packets after a certain period of time goes on until the client receives an `ACK` packet or reaches the maximum number of retransmission attempts. Observe the RTO column and how the value starts doubling up until it reaches a maximum limit.

With the next scenario in *Figure 8.4*, I want you to witness the duplicate `ACK` packet that is being generated because of a malformed packet sent by the server at `216.58.220.46` to the client at `192.168.1.103`. As soon as the client receives it, a duplicate `ACK` packet is sent in response to the malformed packet that is seen out of sequence.

Observe that the `6027` frame with `SEQ = 1920` and `Data payload size = 46` is being sent across from one host to another. Next, in the response frame `6070`, a

malformed packet with a random `SEQ` value was sent in response. Due to this, the host at `192.168.1.103` generates a duplicate `ACK` packet and sends it to the host on the other side with the `SEQ` and `ACK` values similar to the frame `6027`. Now, this time in response, the host at `216.58.220.46` sends a valid ACK frame `6115` with `ACK` incremented to `1966 (1920+46)`, as expected, and then the communication goes on.

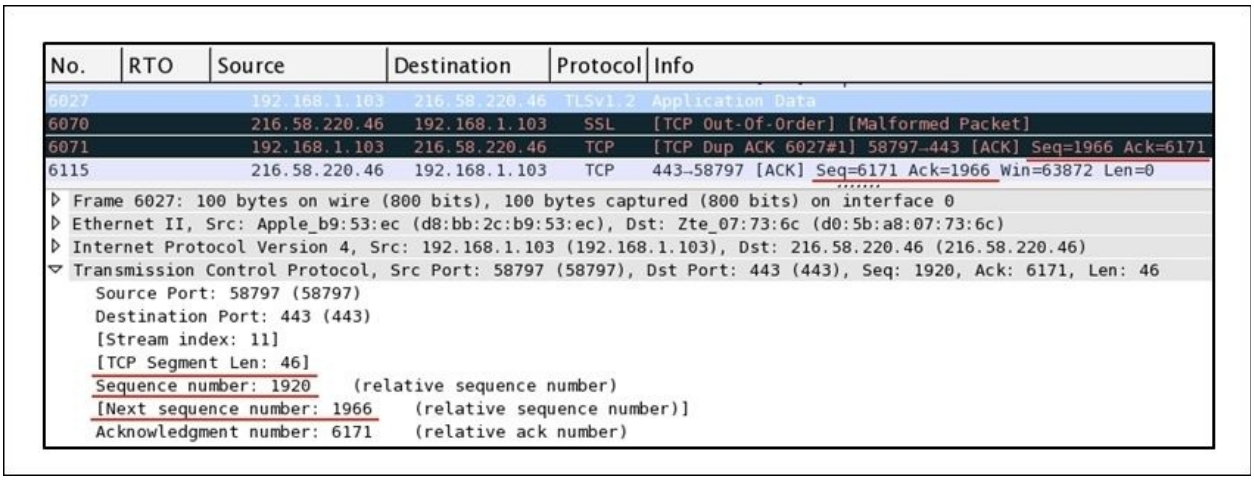
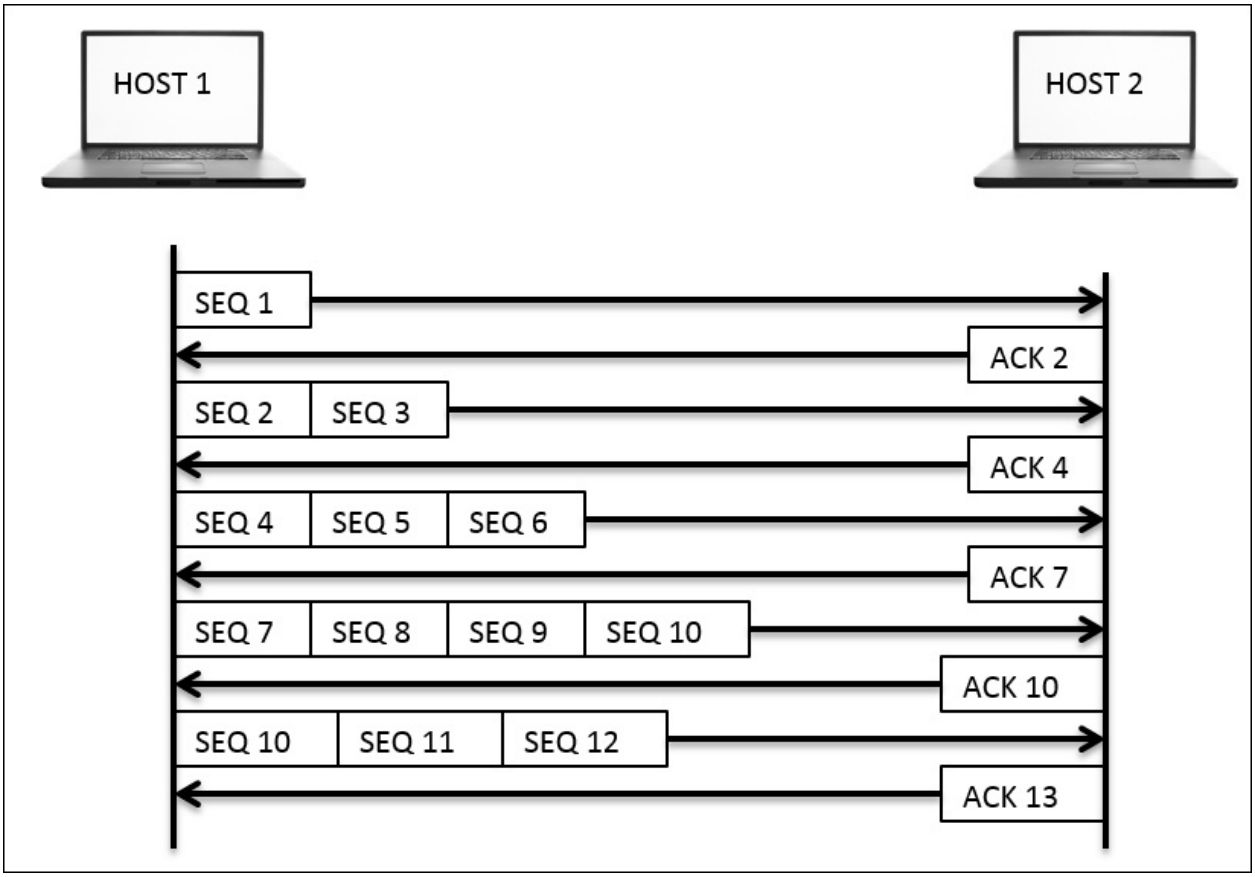


Figure 8.4: Duplicate ACK

With these real-life examples, I expect that you have understood the behavior of TCP error recovery features more precisely.

The flow control mechanism

This is another feature used by the TCP protocol to avoid any data loss during the transmission. Using flow control, the sender syncs the transmission rate with the receiver's buffer space with a motive to avoid any future data loss. Consider a scenario where the recipient has a buffer space of 1,000 bytes available at an instance, and the sender side is capable of sending up to 5,000 bytes per frame. Now, using this information, both the hosts have to sync their window size to 1,000 bytes only to avoid any data loss. Refer to the following figure that shows this feature:



The preceding figure depicts the way both the communicating hosts negotiate the window size for transmission purpose. Observe the behavior, beginning from the frame with **SEQ 1** where **Host 2** responds with **ACK 2** to specify that the frame was successfully received.

Next, **HOST 1** tries to increase the transmission rate to two frames and sends them with **SEQ 2** and 3. **Host 2** responds with **ACK 4**, which denotes that both frames were successfully received. Similarly, we succeed in increasing the rate to three frames.

Next, **HOST 1** increases the rate to 4 and tries sending packets with **SEQ 7**, 8, 9, and 10. This time, **HOST 2** responds with **ACK 10**, which means that **Host 2** receiving the window size can afford maximum 3 frames at an instance, and the sending side should adjust to it.

Next time, when **Host 1** transmits, the windows size would be set to 3 frames, which the recipient can afford to process on his/her end. The window size is not set to a permanent value; it can vary until the whole transmission is completed, and the whole process is called the **TCP sliding window** mechanism and is used to avoid data loss during a transmission.

Think about what would happen if the recipient side is left with no buffer space, that is, 0 bytes. It can handle at some moment during the transmission. What will the TCP do in such case? Will the communication channel drop or the TCP will come up with something more reliable.

Yes, the TCP has another data loss recovery feature called the **Zero window** notification. Here, the recipient side sends a Windows update packet set to 0 bytes and asks the sender to halt the transmission of frames. In response, the sending side will understand the situation and respond with a **Keep Alive** packet that is sent at a particular duration while waiting for the next Window Update packet from the client. Refer to the *Figure 8.6* that illustrates the same.

HOST 1 starts communicating after the three-way handshake process has been completed. After a few packets get transmitted successfully, the receiving side buffer space gets filled up with other resources, so **HOST 2** responds with a Zero Window packet telling **Host 1** to halt sending packets until further notice. Accepting the **Host 2** zero window packet, **Host 1** starts transmitting **Keep Alive** packets in order to keep the connection active and waits for further notice. Once **Host 1** receives the new window size and ACK for the frames that were transmitted, it will start sending the data packets again in accordance with the receiver's buffer space.

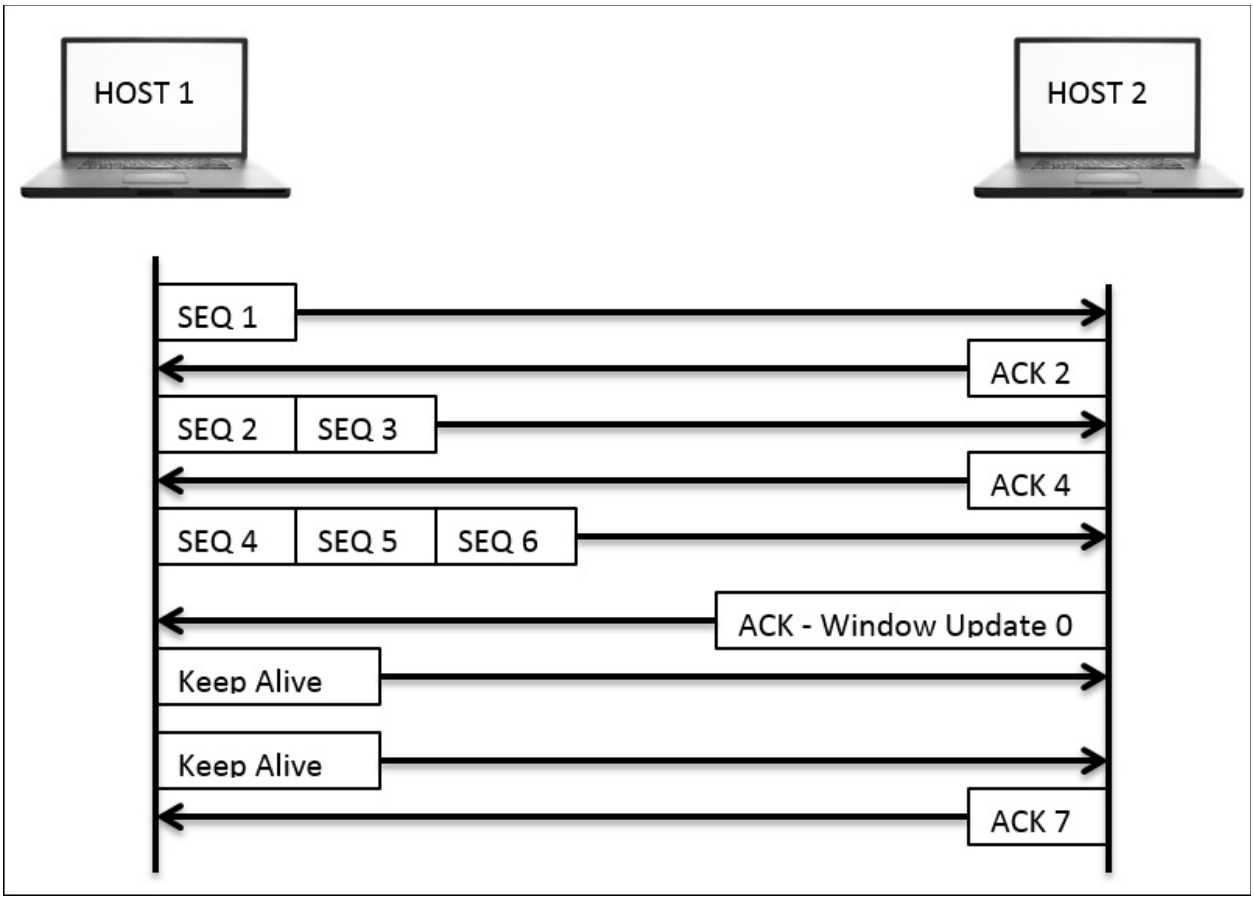


Figure 8.6: The zero window notification

The technique we discussed here is quite efficient in preventing any data loss that might happen during a transmission or due to an overwhelmed sender. The TCP hosts a great mechanism to control the transmission process, thus making it more reliable for any type of communication.

Troubleshooting slow Internet and network latencies

The discussion that we had on delays observed in the list pane can be categorized in two categories: the normal/acceptable delays and the unacceptable delays. Yes, you heard me right, there are some forms of delay that are acceptable, and you should not waste any precious time of yours in troubleshooting any of those cases.

Assign a category to your current scenario on the basis of the test results that you have obtained from the client site (try to put sniff packets from the complaining client's perspective) into one of the following categories: wire latency, client latency, and server latency. Seeing your scenario with the perspective of one of these cases will assist you in solving the problem with a more process-oriented approach, hence making the task less complex, which will end up getting sorted out in lesser time with lesser resources.

Before you start troubleshooting such scenarios, I would highly recommend that you change the default list pane view by customizing the existing time column (customize the time value to `seconds since Previous Displayed Packet`), which would work as a column to figure out latency issues, that is, it will show you the total amount of time between two related packets in a sequence. Refer to the following figure to customize the time column.

To further elaborate the best practices that are followed, I will discuss a step-down approach, which you can use as part of your checklists. Make sure that you understand one thing clearly: tracking an issue can be quite critical on a server side because you may see thousands of packets flying in and out per seconds. This can be really messy and would only end up in making the whole problem more intense. Looking at thousand of packets to figure out the source of slow Internet connection doesn't sound feasible. So, the best option would be to filter out things, prioritize them, and look at the problem from the client's end first.

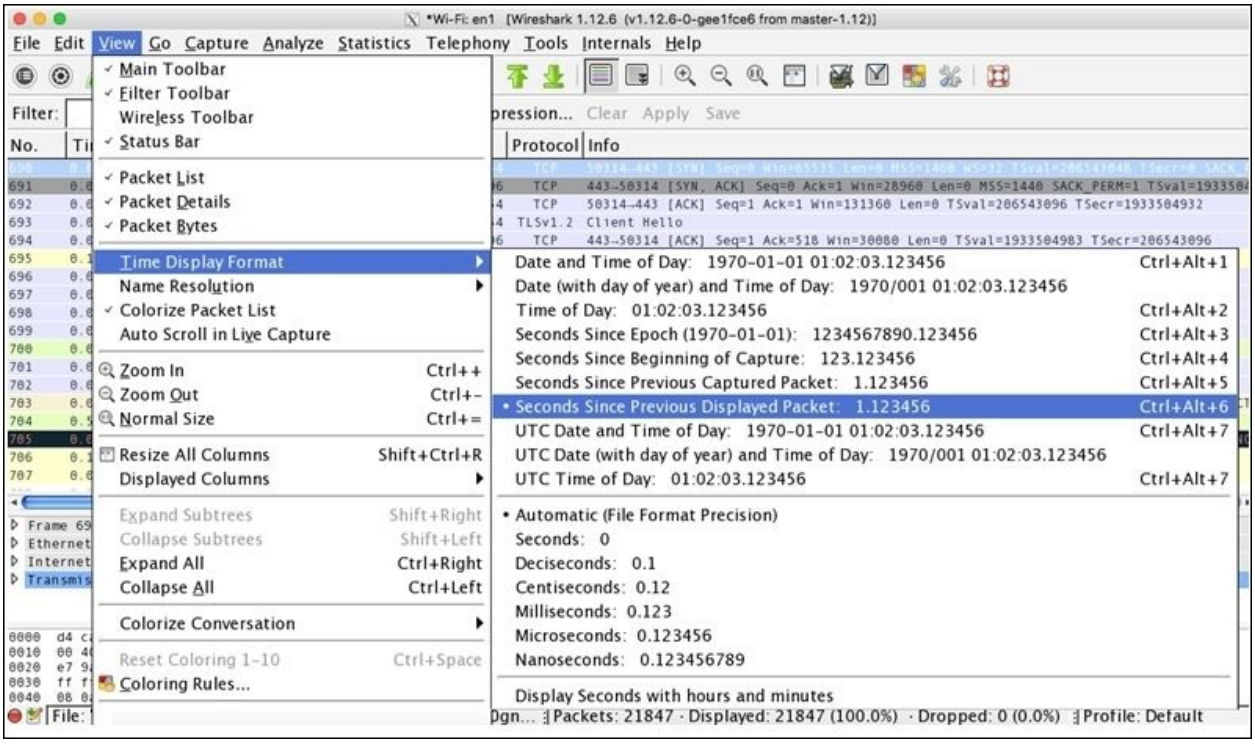
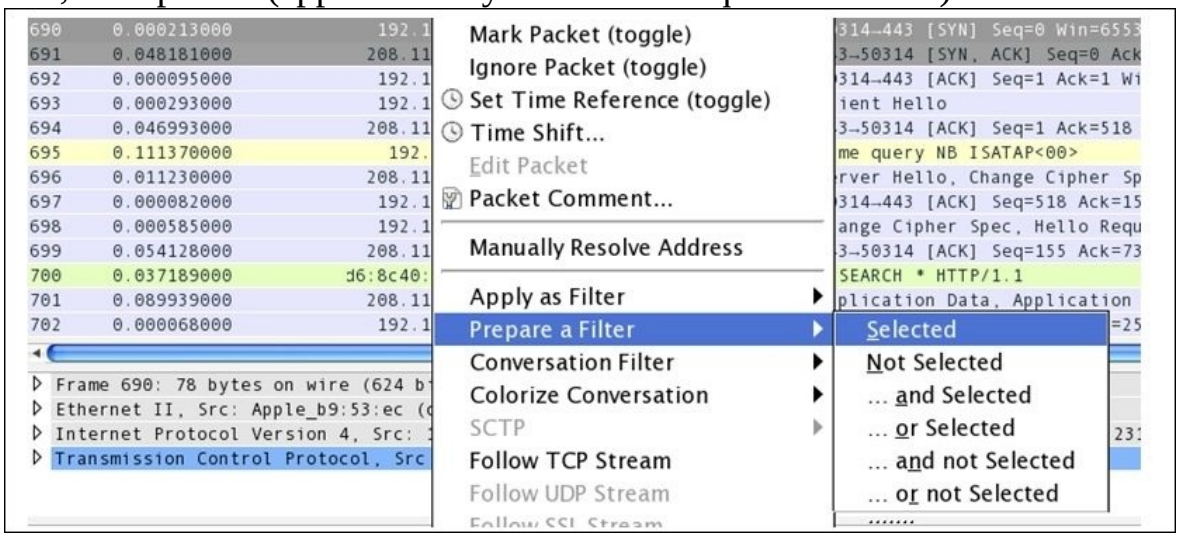


Figure 8.7: Customizing the time column

- Starting your investigation at the client's end makes it much simpler because you won't be dealing with several packets that may not be relevant to your scenario. On the other side, if there is even a hairline chance that you won't be able to see the packets that are relevant to you, this might make the troubleshooting experience a bit challenging.
- Apart from all the challenges that you might face at the client's end, the first thing you should ask your client is to replicate the problem if possible, or if the problem is occurring in a time-based manner, then you should wait at the client's end in order to witness and understand the scenario. The ultimate goal should be to capture the relevant packets and get a crystal clear understanding of the problem that the client is facing from their perspective.
- Now, when you have the trace file in hand, you can look at the process where the client is trying to connect to the server: the whole process where the client issues a DNS query with an objective to attain a server's logical location over the Web. If the local DNS cache already holds the IP address

of the server, then you might not observe any DNS packets; instead, a direct SYN packet would be seen in the list pane sent to the server to initiate the independent connection. What you need to make sure here is that if the DNS queries are seen in the list pane, then the round trip time should be low, as expected (approximately less than or equal to 150 ms).



The next would be the three-way handshake packet that you will be observing in the list pane. The best option would be to isolate the communicating hosts that can help you in eliminating any further communication. You can just right-click on the communication and create a filter as illustrated in *Figure 8.8*

- Once you have filtered out the problematic connection between the hosts, the next task would be to observe the total time. The time between duration when the SYN packet was sent and the corresponding SYN/ACK packet was received. This can be compared with the baseline that you already have to come up with a variance that could help you in pointing out whether the connection is slow or is working fine. Refer to the following screenshot that illustrates the same:

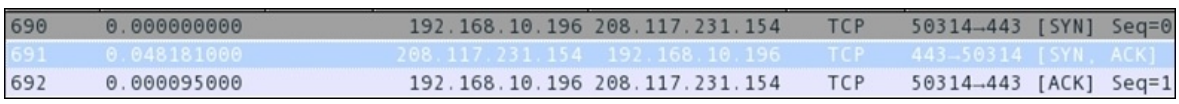


Figure 8.8: The time between the SYN and SYN/ACK packets

- As you can see, the time between the SYN and SYN/ACK packets is relatively low, and this seems to be a good working connection. This kind of connections can be helpful while you are designing a baseline for your network. At a later point in time, the same can be used to compare with problematic scenarios. Refer to the following screenshot that show DNS and TCP packets of the same communication:

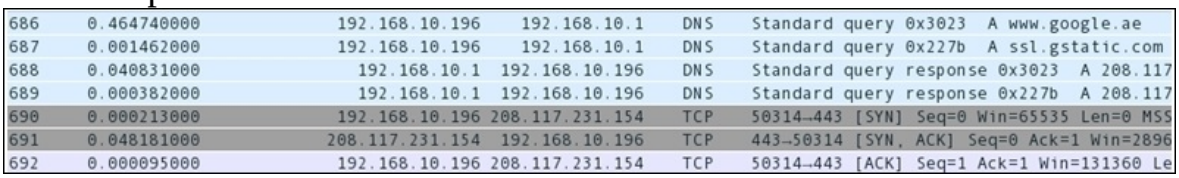


Figure 8.9: The ideal baseline trace

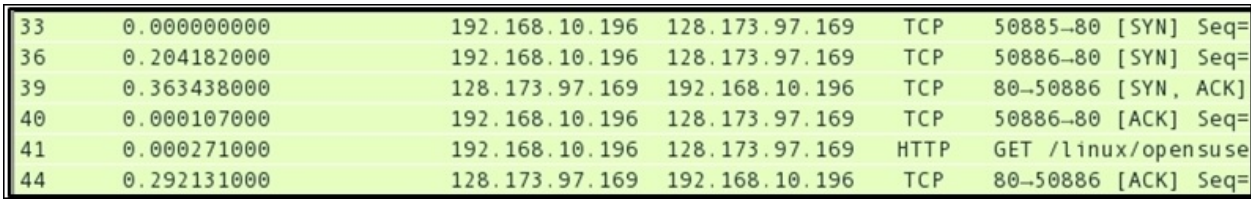
- The client issues a request to visit the `google.ae` (frame `686`) website, which the local server acknowledged in order to first look for the IP address in a local cache. Once the local DNS server completes, the search process, the client receives DNS responses including Google's IP address, which can be used to visit the website (frame `688` and `689`).
- As soon as this process completes, the client at `192.168.10.196` issues a SYN request to one of Google's IP address in order to visit the web page. Without any further delay (less than tenth of a second), the server responds with `SYN/ACK`, and the process goes on.

Let's suppose that the total time between the `SYN` and `SYN/ACK` packets is high by approximately 0.90-1.0 seconds. At first glance, you ignore this an move ahead, and you will observe a quick `ACK` packet sent in response from the client followed by a HTTP `GET` request (in case the client is visiting a website). Next, the `ACK` packet acknowledging your `GET` request surprisingly takes more than a second to come. Now, this points to some serious latency issues. The question is, who will be the one you are going to blame—the client or the server? The client did its part by sending the `SYN` packet on time. Then, is it the server who is handling a high load of traffic and is quite busy with other applications, because of which you are handling high round trip time? The answer is neither the client nor the server. Then why is the round trip time high? The probable answer for such cases in my knowledge would be the wire. Yes, you heard it right. The wire

can also take part in making your network slower then expected. So, while troubleshooting slow networks, if you observe high round trip times associated with the `SYN/ACK` and `ACK` packets, then you can be sure that your client and server are not the source of the issue.

What you can do is start examining the devices between the hosts, such as the routers, switches, firewalls, proxy servers, and so on. Although the example we talked about doesn't give you the exact source of the problem, it definitely gives you a clear understanding that both the communicating hosts are not promoting any form of latency.

Now, for better understanding, I would like to show you the same in practical terms. Refer to the following screenshot that lists out a few packets shared between two hosts, starting from a three-way handshake:



33	0.000000000	192.168.10.196	128.173.97.169	TCP	50885→80 [SYN] Seq=
36	0.204182000	192.168.10.196	128.173.97.169	TCP	50886→80 [SYN] Seq=
39	0.363438000	128.173.97.169	192.168.10.196	TCP	80→50886 [SYN, ACK]
40	0.000107000	192.168.10.196	128.173.97.169	TCP	50886→80 [ACK] Seq=
41	0.000271000	192.168.10.196	128.173.97.169	HTTP	GET /linux/opensuse
44	0.292131000	128.173.97.169	192.168.10.196	TCP	80→50886 [ACK] Seq=

Figure 8.10: Wire latency

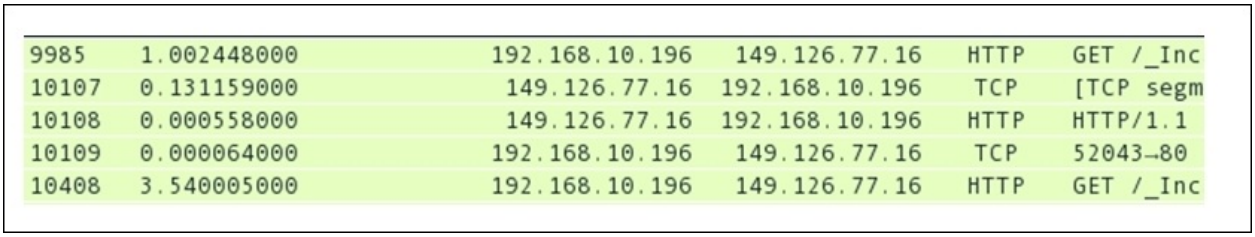
First, the client located at `192.168.10.196` and the server located at `128.173.97.169` start communicating. In the beginning, we see that a three-way handshake takes place between the client and the server, but did you notice the amount of time it took for the `SYN/ACK` packet to come (more than 0.36 seconds). Look at the frame `39`, and it is something that you should take care of. Moving on, we saw one more similar event after the `GET` request was issued, where the `ACK` packet took approximately 0.30 seconds to come back. The latency observed is not because of the client or the server, as we discussed earlier. The latency here is promoted by the devices that lie on the wire. The best troubleshooting option in such cases would be to look at the routers, switches, or any firewalls that were implemented without wasting time in troubleshooting the source and the destination.

Client- and server-side latencies

You might think about the scenarios where you would come across or see latency issues that the client/server promotes. Let me explain this to you with some real-life examples; first, we will take a look at the latencies promoted by the clients.

A few days ago, I was just visiting some random websites over the Internet to look for some research material, and meanwhile, Wireshark was running in the background and capturing every packet I was tying to visit. I surfed the Web for approximately 3-4 minutes and then closed the browser as well as stopped Wireshark from sniffing any packets. After the whole thing, I decided to look into the trace file to investigate any client-side latency issues.

Refer to the following screenshot from my trace file, which shows frequent client-side latencies that will eventually affect the performance of my network:



9985	1.002448000	192.168.10.196	149.126.77.16	HTTP	GET /_Inc
10107	0.131159000	149.126.77.16	192.168.10.196	TCP	[TCP segm
10108	0.000558000	149.126.77.16	192.168.10.196	HTTP	HTTP/1.1
10109	0.000064000	192.168.10.196	149.126.77.16	TCP	52043→80
10408	3.540005000	192.168.10.196	149.126.77.16	HTTP	GET /_Inc

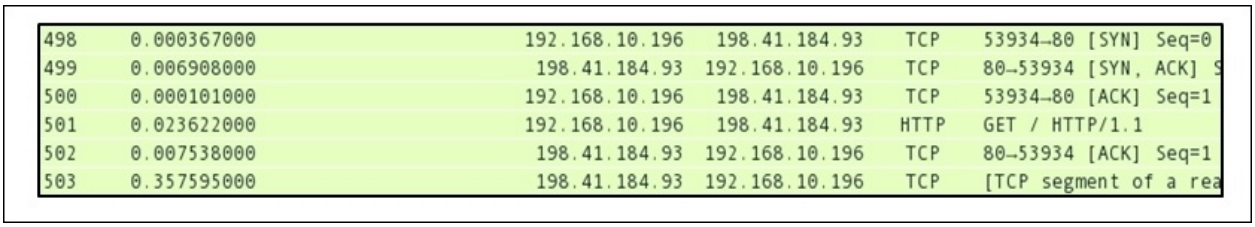
Figure 8.11: Client-side latency

As you can see in the frame `9985` and frame `10408`, there are `GET` requests that my machine at `192.168.10.196` had issued, and the amount of time it took was 1 second for the first time and more then 3.5 seconds the next time. I became curious and started thinking about why this happened and what can be the most appropriate reason for such latencies.

Once I started further investigation, I saw that the three-way handshake process happened in a timely manner and there were no signs of latencies. Now, my attention went to my machine. Maybe, there is something that is tampering with my network connectivity. I looked at the resource allocation window in terms of

primary memory and CPU utilization. What I saw was that the CPU and memory utilization meter were showing high consumption, which led me to enquire more about the number of applications running. There were three virtual machines running that I forgot to turn off, which were utilizing all the memory. This, in my belief, is one of the strongest reasons, because of which I was experiencing latencies on the client side (my machine). I hope that, with this practical example, you might have understood how client-side latencies can be one of the reasons for low network and Internet performances.

Moving on with this simple example, let's get ourselves introduced with server-side latency issues. I followed the same approach of surfing the Web with random websites while capturing packets with Wireshark for a couple of minutes and then analyzing the cause of any form of latency that can be seen in the list pane. This time, I came across an interesting session between my machine and a website. First, I would like you to have a look at it. Refer to the following screenshot that illustrates this:



498	0.000367000	192.168.10.196	198.41.184.93	TCP	53934→80 [SYN] Seq=0
499	0.006908000	198.41.184.93	192.168.10.196	TCP	80→53934 [SYN, ACK] S
500	0.000101000	192.168.10.196	198.41.184.93	TCP	53934→80 [ACK] Seq=1
501	0.023622000	192.168.10.196	198.41.184.93	HTTP	GET / HTTP/1.1
502	0.007538000	198.41.184.93	192.168.10.196	TCP	80→53934 [ACK] Seq=1
503	0.357595000	198.41.184.93	192.168.10.196	TCP	[TCP segment of a rea

Figure 8.12: Server-side latencies

As you can see, the session between my machine at `192.168.10.96` and the server at `198.41.184.93` begins with a smooth three-way handshake without any sign of latencies. Next, the client issues a web request, following which the server sends an acknowledgement. Uptil here, everything has gone flawlessly, and there were no traces of latencies. However, when the server was about to start the data transfer, the server stopped for a while, as you can see in the frame `503`. The server took around 0.35 seconds to initiate the data transfer. This clearly illustrates that the server might have experienced heavy network traffic, or may be, the server was running several applications that were causing high CPU and memory consumption. There can be several other reasons as well for the latency that we just witnessed. Observing all of it, we can give a conclusion

that the server is the reason for the latency; in this case, the server was incapable of processing the client's request in a reasonable amount of time, which ended up as a minor latency issue.

You learned how the devices over the wire, the client side, and the server side can promote high latencies while you surf the Internet or even your internal `LAN` network can be a victim of the same. We talked about delays before the server's `SYN/ACK` packet is received. These delays can happen because of the device in between (over the wire) and may be witnessed due to the server's high response time. Let's make things more interesting with a small practical example about identifying high HTTP response time. This will be useful for you to identify high response time. Follow these steps to replicate the same in parallel:

1. Open your browser and visit some websites while Wireshark runs in the background listening to your packets.
2. Once you have visited at least 3-4 websites, you can stop the capture process.
3. Now, switch to Wireshark and make some necessary changes. First, disable **Allow subdissector to reassemble TCP streams**. Select any TCP packet in the list pane, then right-click on the TCP section in the details pane, and then click on the **Allow subdissector to reassemble TCP streams** option to disable it. Look the the following screenshot that illustrates this:

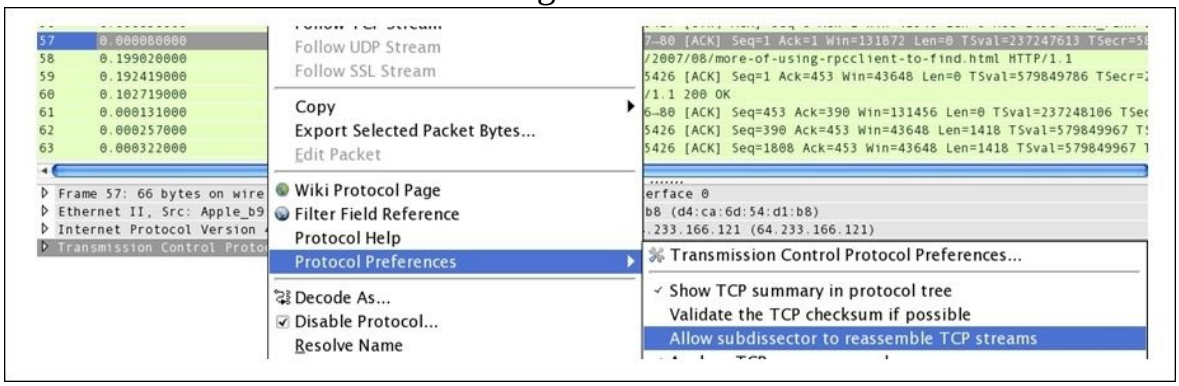


Figure 8.13: Disable the Allow subdissector setting

4. Next, we have to add the `http.time` delta column to the list pane in order to see things more clearly and to easily identify any traces of latencies.
5. Select any HTTP packet from the list pane and then expand the HTTP

protocol section in the details pane. Then, right-click on the **Time since request** parameter and click on the **Apply as Column** option. Refer to the following screenshot that illustrates this:

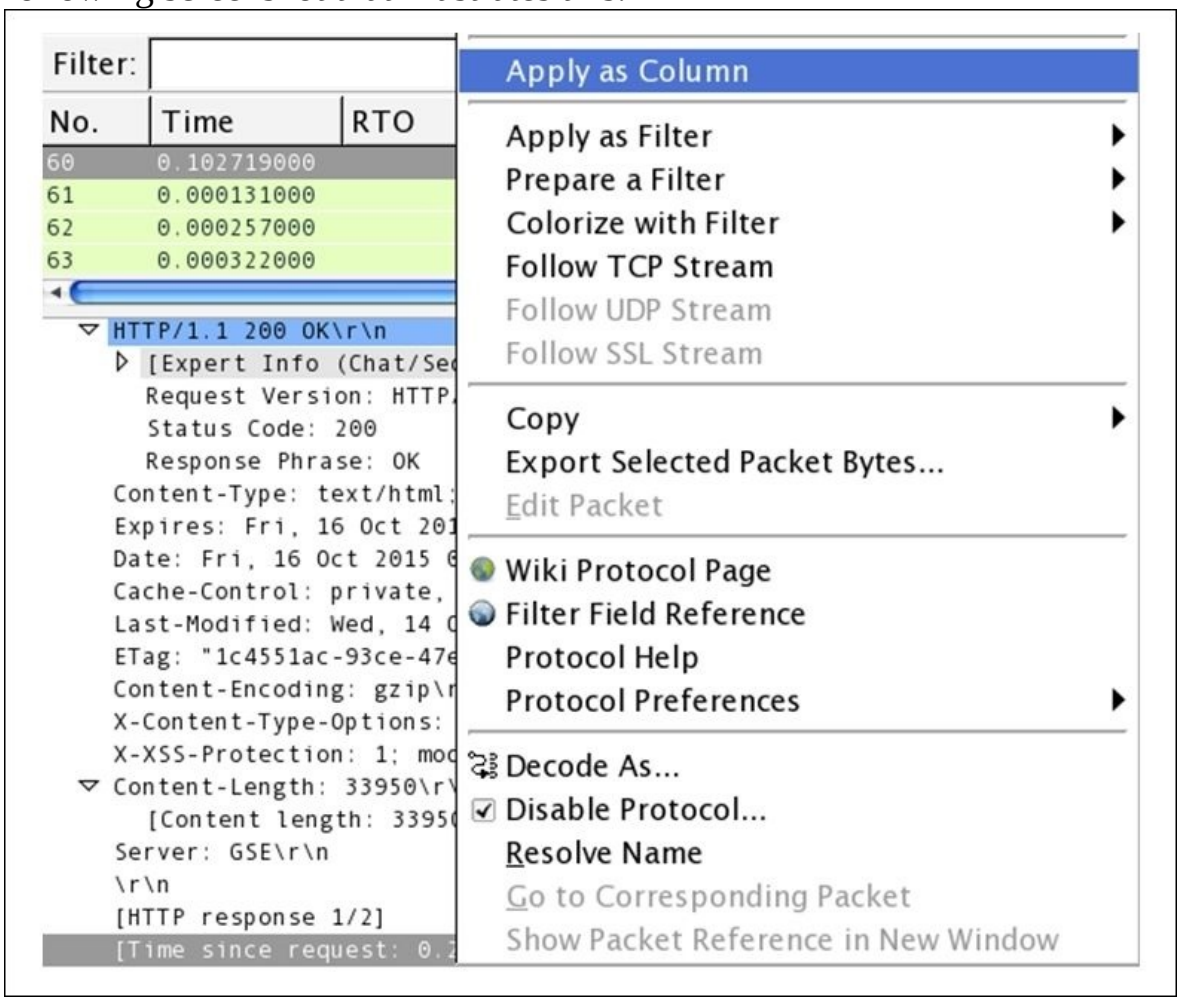


Figure 8.14: Apply Time since request as a column

6. Once this is done, you would be able to see the **Time Since Request** columns just before the info column in the list pane.
7. Now, you are left with just one step: to identify the highest response time from the web servers that you visited. Simply sort the newly added columns in a descending order to the highest response time. Refer to the following screenshot that illustrates this:

Figure 8.15: Sorting the http.time delta column

8. Once this is sorted, you would be able to see the highest response time at the top of the list pane, as shown in the following screenshot:

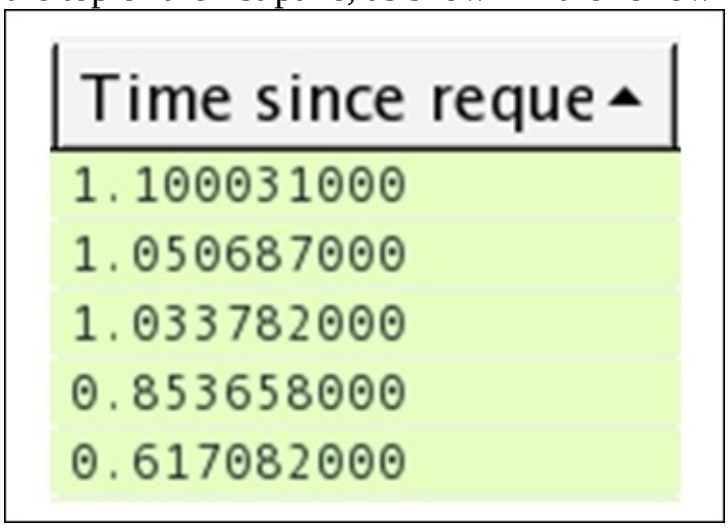


Figure 8.16: High HTTP response time

9. The session at the top of my list pane between my machine and a web server that I visited denotes quite a high response time of more than a second. See how easy it was to identify the `http` delays in order to make your troubleshooting job easy. I hope it would be easy for you to replicate the same.

You can also achieve this in a visual representation, where you can create an IO graph to identify high latencies. Refer to the following small illustration using which you can replicate the scenario (note that I am using the same trace file that

we saw earlier in the previous example):

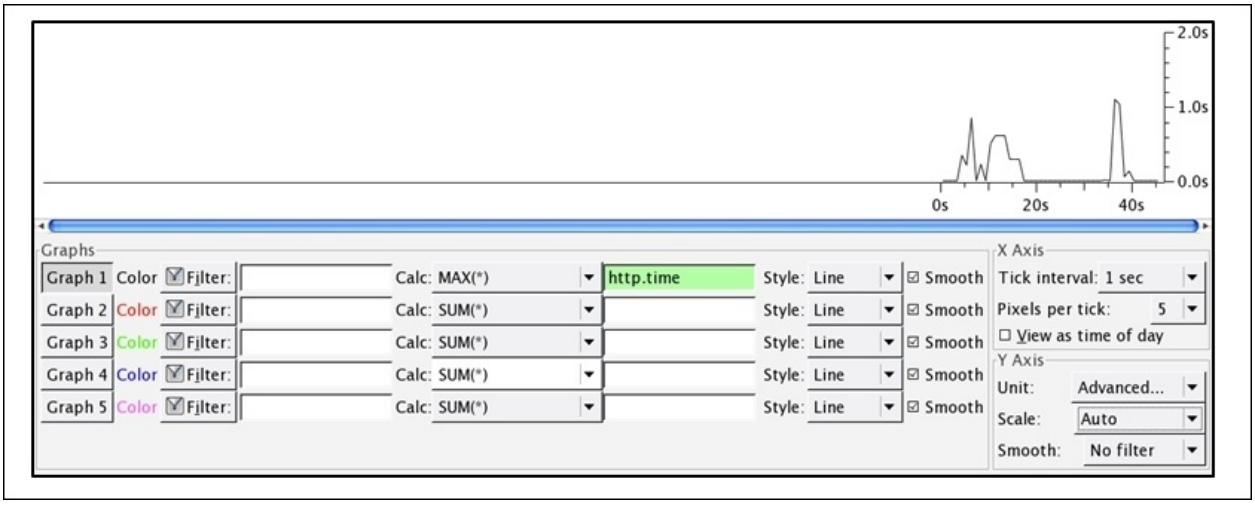


Figure 8.17: Using an IO graph to identify the delays in HTTP response

As you can clearly observe in the graph, the response time for the requests you made took more than a second to complete in a total browsing session of approximately 45 seconds.

There can be multiple situations where you will witness such traffic patterns; this one is definitely because of a web server that makes your web surfing experience bad. The reasons behind such a pattern can vary from a server in a heavy traffic load to a server hosting several applications, or it can be possible that the server you are trying to visit might be consulting some other web server in order to fulfill your request.

Next, let's see an example where DNS queries and their responses are responsible for causing your Internet or local networking experience to suffer. As we saw, other protocols in conjunction with DNS make the whole networking experience better, but at times, the same DNS protocol can cause trouble. Follow the next steps to identify the source of problems using DNS response time:

1. Open your browser and visit at least 3-4 websites. Wireshark should be capturing your web session packets while in the background.
2. Stop the capturing process and apply `dns` as a display filter in your trace file in order to see only `dns` packets.

3. Now, select any `dns` response packet from the list pane and expand the corresponding DNS section in the details pane for the same packet. Right-click on the **Time** parameter and click on **Apply as Column**. Refer to the following screenshot to see this:

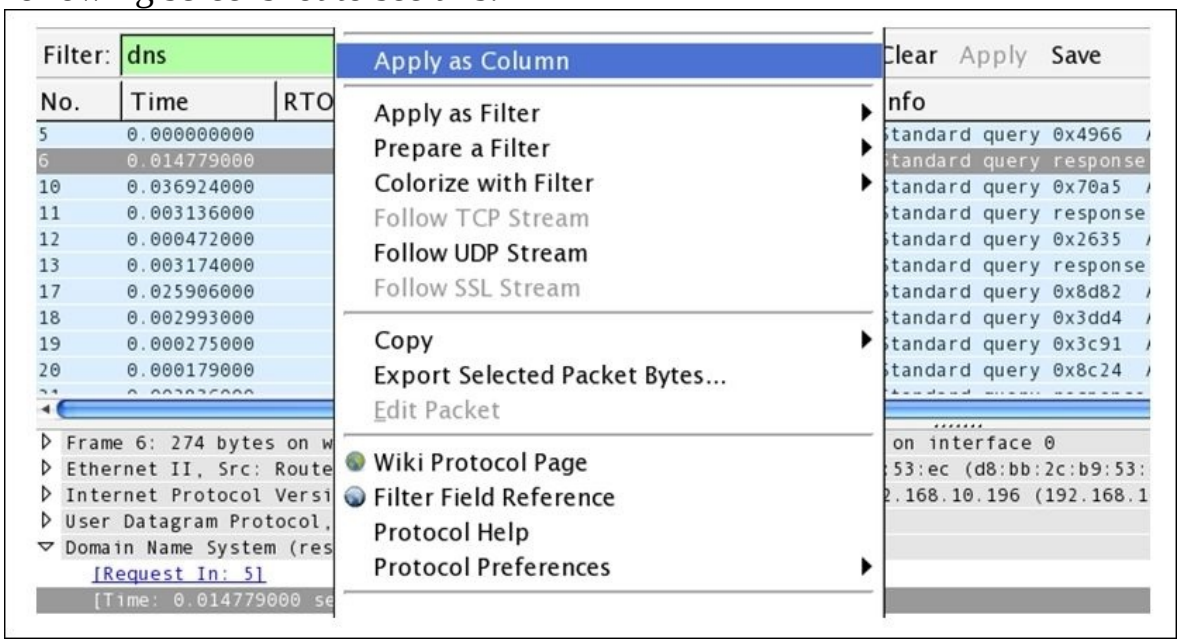


Figure 8.18: Applying DNS Time parameter as column

4. Once you've done this, you will see a time column next to the info column in the list pane.
5. Our next objective is to sort the column in a descending order to figure out the highest DNS response time. Refer to the following screenshot to replicate the same:

Figure 8.19: Sorting the DNS time column in a descending order

6. Once this is sorted, you would be able to see the session details in the list pane with the highest DNS response time that can be used to investigate further. If the server belongs to your premises, then you are the only one who has to take care of it. Refer to the following screenshot that illustrates this:

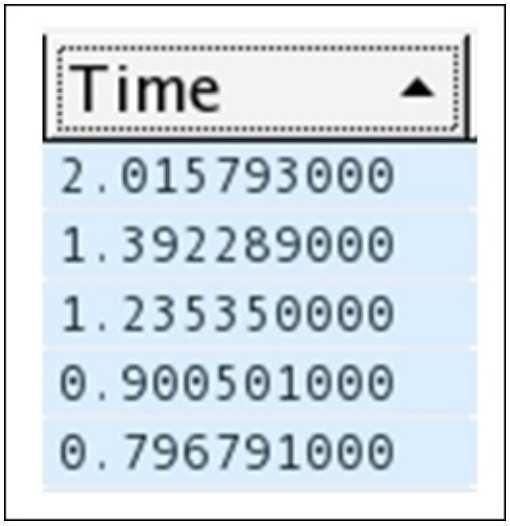


Figure 8.20: High DNS response time

7. Seems like some of the servers are responding really slow, and this badly affects your overall web surfing/networking experience.
8. Similarly, you can create an IO graph to see the whole scenario in a graphical form, and it would be far easier to visualize and understand the case. Refer to this screenshot that illustrates this:

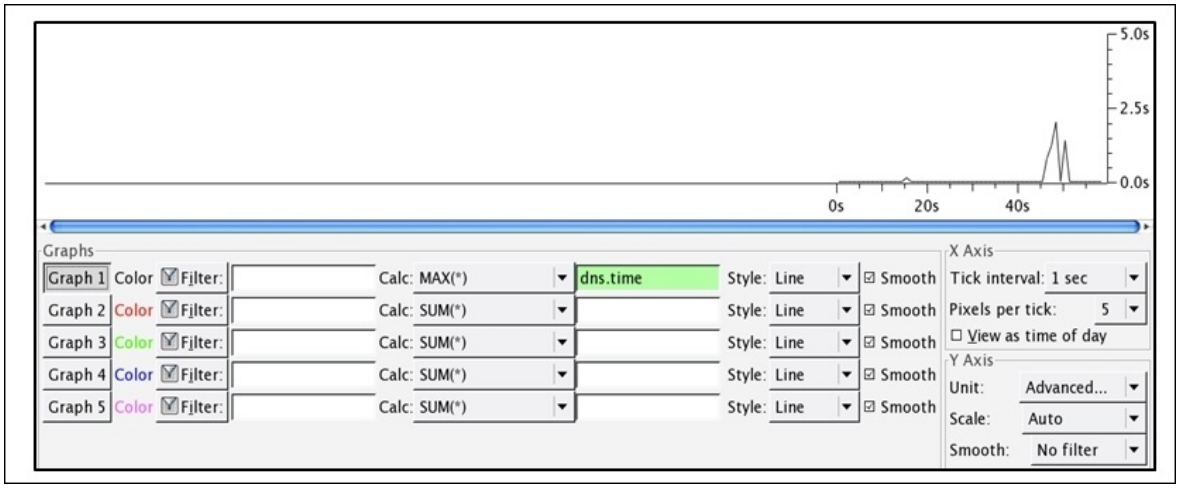


Figure 8.21: DNS high response time depicted with the help of an IO graph

You can easily observe in the preceding graph that the DNS response time was quite high and reached to an approximate of 2.5 seconds, and it is something that should be taken care of.

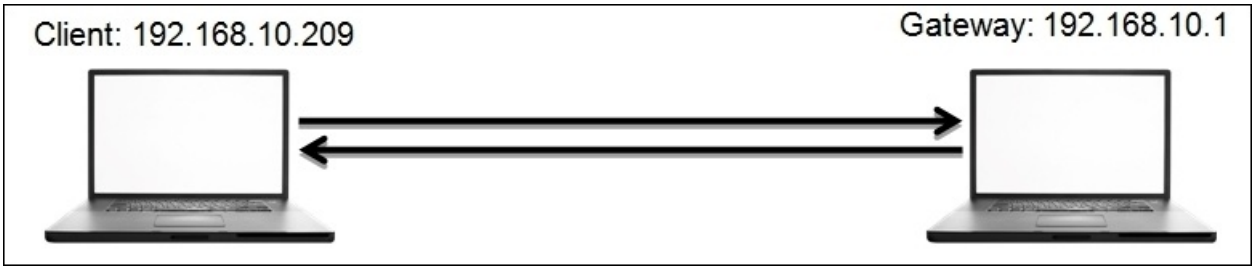
Through the preceding realistic examples, I hope you have understood the approach that can give you a kickstart in troubleshooting such scenarios in future corporate infrastructures, which you might be asked someday to troubleshoot.

Troubleshooting bottleneck issues

Next, we have a commonly occurring issue in corporate networks. You might have already gone through the harsh suffering of troubleshooting them using various hardware and software tools. The first thing to do is to understand what these issues are and what kind of problems we can we face.

When packets are queued up or there is a delay in the transmission process between the host, which is not expected to happen, you might think "why do such delays happen?" The answer to this depends on many factors such as when your system of the server side is not able to send/receive information with the speed at which it is being processed. These kind of issues severely affect the performance of networks by slowing the rate at which the TCP/IP packets are transmitted, because of which the data between the hosts starts moving back and forth at a comparatively slower rate.

Using my small LAN network, I decided to create an exercise, which you can also replicate on your end easily. For the infrastructure, I have a gateway at `192.168.10.1` and my client at `192.168.10.209`. Refer to the following figure that illustrates this:



What you need next is a network traffic generator. Research it a bit and try to use anyone that makes you feel comfortable. Lastly, you need a ping utility, which is already installed on every known operating system.

So, here's the scenario. I will start a non-top ping from the client to the server. While the client is pinging, I will launch the traffic generator application, which will try to interrupt the ping process by trying to consume the gateway's resources in order to create a bottleneck scenario for the client.

We will first see a normal traffic pattern in the IO graph so that we would work as our baseline when we would be required to compare with the bottleneck issue. Here is the screenshot for the normal traffic pattern shown in terms of an IO graph:

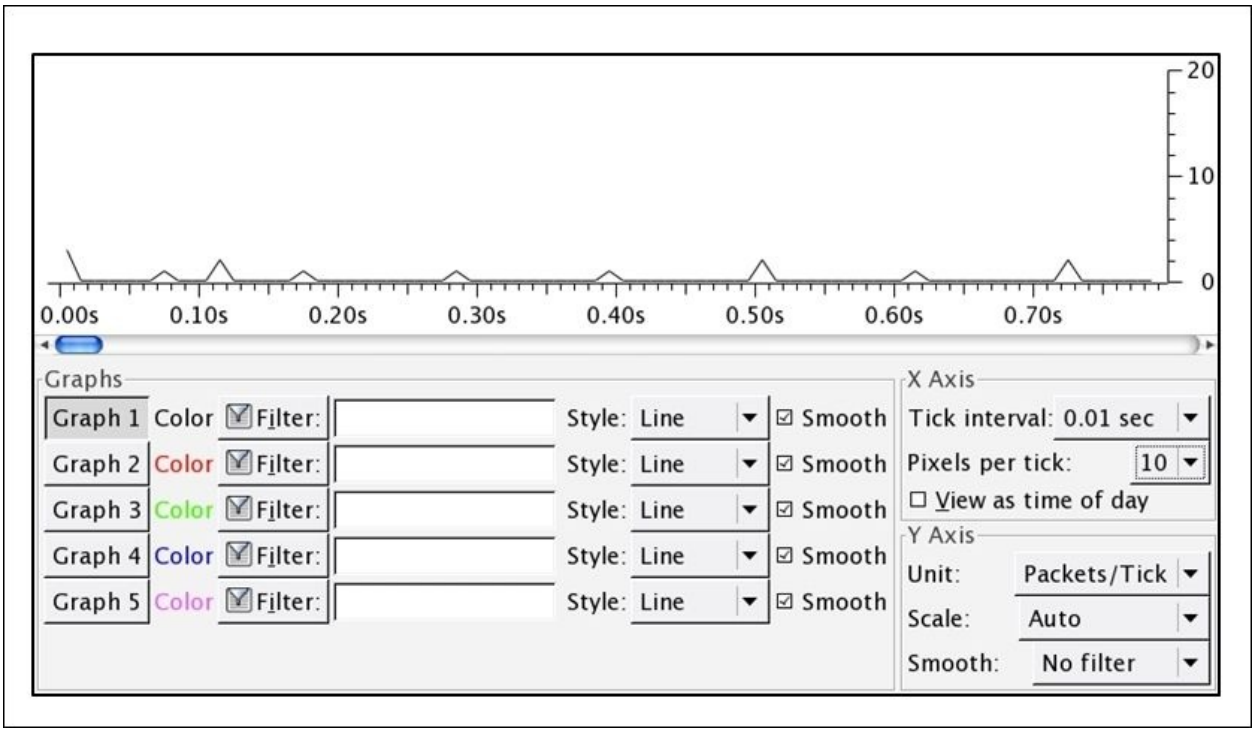


Figure 8.22: Normal traffic in an IO graph

In the preceding graph, no major deviation can be observed; hence, we can include such a traffic pattern while creating a baseline for our network. Just the ICPMP packets are sent from the client to the server without much trouble.

Next, I want you to see and observe the difference between the traffic pattern that we saw and the one below the IO graph, which was captured for the same network infrastructure. However, there was one more application that was involved in the replication of the event, which generated unnecessary traffic. This resulted in network clogging, which is popularly known as a bottleneck.

The application I used is the network traffic generator that can be used to deviate a normal traffic pattern. This results in a network bottleneck scenario and can

even result in a denial of service. Refer to the following screenshot for reference:

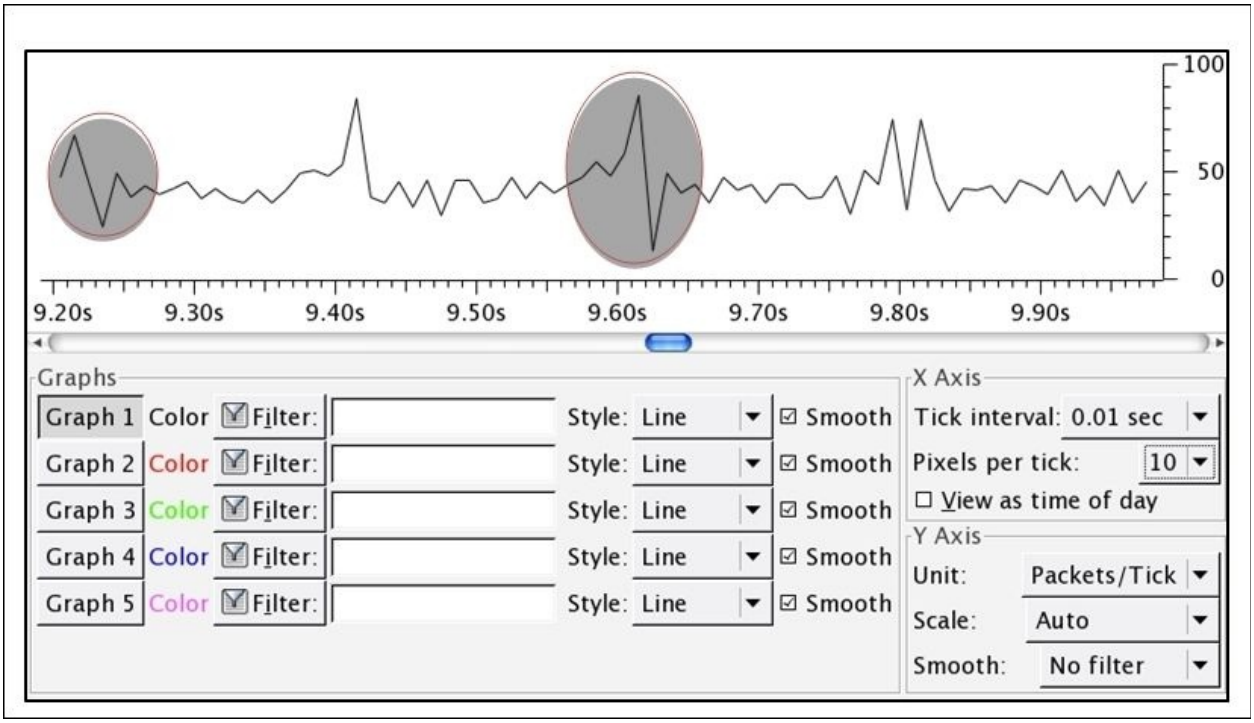


Figure 8.23: A bottleneck scenario

Bottleneck issues are represented by ups and downs, as shown in the preceding graph. The rate at which the throughput drops is the same rate at which it jumps up, and this pattern of deviation in normal traffic denotes that there is a bottleneck being formed.

When every technique you know about troubleshooting fails, then at the end, you can use the network baseline, which can prove worthy while dealing with the slowness of the network. As discussed earlier, a network baseline is just crucial information that you have collected through various points in your network. The sole purpose of the network baseline you have is to compare abnormal traffic with it in order to understand the level of deviation.

We already discussed slow DNS and HTTP responses that make up your web surfing experiences. If you already have a baseline regarding your network, then it would be thousand times easier for you to troubleshoot. You would be able to

identify the root cause of the situation you are dealing with, and definitely, this will save a lot of time for other analysis.

Remember one thing that the baseline created for two different networks can vary in vast aspects, so you should not compare them with each another. An interesting and creative way of creating a baseline would be to create separate baselines, that is, one for the network, one for the hosts in your network (how well they coordinate with each other without creating much noise), and one for the applications communicating over a network.

While creating baselines, you can also consider each and every site you are working with separately. In my opinion, the best approach would be break up each site with similar categories. When you are dealing with a WAN, a troubleshooting site baseline can prove useful. Several components can be considered while dealing with WAN sites, such as data transfer rate, several applications in use, the pattern of the broadcast traffic, and various other categories that you may come up with can come handy while making a standardized baseline for a particular site.

Troubleshooting slow networks is definitely a piece of art. I would say, you won't be able to get its real gist unless you get your hands dirty. With experience, you will gradually gain the insight required to solve problems ranging from slow Internet to complex infrastructure-related issues

Troubleshooting application-based issues

There can be scenarios where applications running in your network can be one of the major sources of issues that clients face. You cannot blame the network every time for not working popularly; there can be other reasons as well for the anomalies. When troubleshooting any application-based issue, capturing packets from one end won't be fruitful enough. You should try to move to analyzers all around and capture as many traces of the application's traffic as possible. Capturing from multiple points will give you a much closer insight into network-based applications.

As discussed earlier, you can create baselines by following certain different parameters. Similarly, for network-based applications, there can be a certain defined set of rules, by using which the best baseline for your network can be formed, for example, dependencies applications have another coordinating application, analyzing the startup and shutdown process, the rate at which the application transmits packets, various protocols that coordinate in order to make the application work flawlessly, the way an application interacts with the network once a new installation is in process, and so on.

While creating a baseline for application-based performance issues, it won't be feasible all the time to capture traffic directly from the complaining hosts because it may cause the hosts to suffer high-traffic load and might make it unusable. For your trace file, there might be an unusual number of dropped packets that would get captured and would make your application baseline less appropriate.

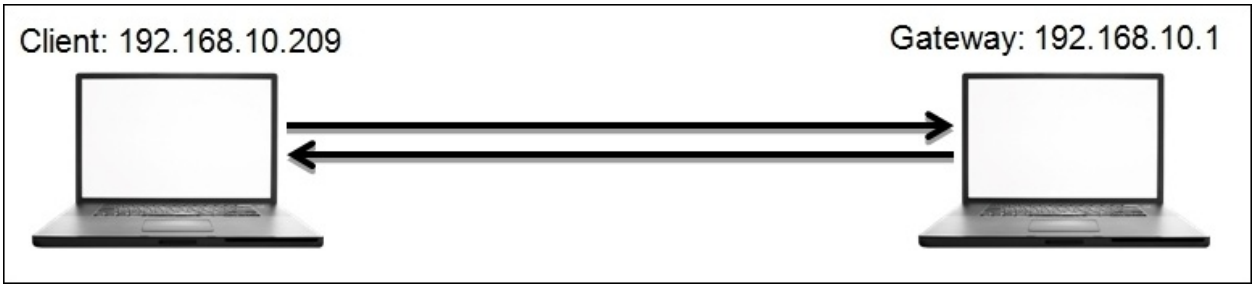
As long as dissectors in Wireshark are able to translate the application-based requests and responses in a plain-text format, you are good to go. In the following section, I will take two popular application protocols, HTTP and DNS, to illustrate a few basic scenarios that you can replicate in order to follow the methodology.

First, we will look at the HTTP application-based anomalies. Remember that you should be able to identify the responses from the error-prone application if you are aware of the response code. As you know, HTTP is based on the request/response model, where a client requests for a certain resource to the

server and the server responds with the valid resource if available; if not, then with a certain error code, which your browser is able to translate.

HTTP error codes are categorized into five sections of errors, where each error is based on certain logical parameters. To learn more about error code, visit http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html. For illustration purpose, I will explain the procedure so that you can figure out the most commonly seen error code, which is client errors.

The infrastructure I am going to use is pretty simple, easy, and similar to the one that we used earlier. The client is located at `192.168.10.196` and the gateway is located at `192.168.10.1`. I will try to make a few requests to the gateway and a few to any web server located in the wild (note that my intention is just to replicate error code that you can see in the list pane of Wireshark, and not to compromise any web server.)



At first, we will try to generate some client error code. Follow the next steps to walk through this; otherwise, you can just read it once and then replicate the whole scenario:

1. Open your browser and visit the default home page of your gateway. Hopefully, it will present you with a login screen like the one shown here:

Figure 8.24: The gateway's Login panel

2. Open Wireshark, and let it run in the background while capturing all your activities.
3. Enter an incorrect password in the password field and click on **Login**. This will show you the **incorrect login name and password** message on the screen or something similar.
4. Next, visit any random website and click on any link. After the link is successfully opened, change the web extension of the web page visible in the address bar to anything such as `.foo`, `.abc`, and so on. Doing this will give you an error on the web page, such as **page not found**. Just ignore it for time being.
5. Now, come back to Wireshark and stop the packet capturing process that we started earlier.
6. You should be able to see a number of packets in the list pane, but our concern in this section is to look at error code messages and nothing else.

7. Now, click on the display filter box and apply the `http.response.code >= 400` filter. Then, click on **apply**. Refer to the following screenshot that illustrates this:

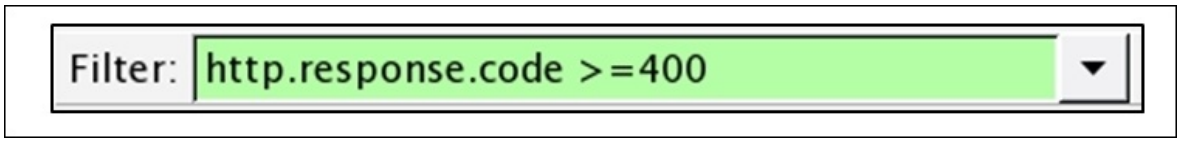


Figure 8.25: Display filter

8. Once the filter has been applied, you will be able to see only those packets that match the criteria. Refer to the following screenshot that illustrates this:

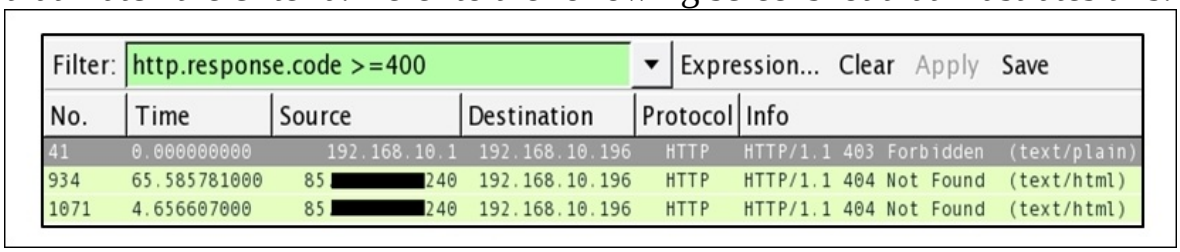


Figure 8.26: HTTP Response code >= 400

9. See, how easily you were able to identify error code from an enormous trace file.
10. You can also create a button for the same. Once you click on it, you will only be able to see relevant packets. You can colorize them for a better viewing experience.
11. We learnt about Coloring options in the earlier chapter. I want you to learn how to create a button for specific display filters this time.
12. Do not clear the current filter; just click on the **Save** button that is next to the **Apply** button in the display filter area.

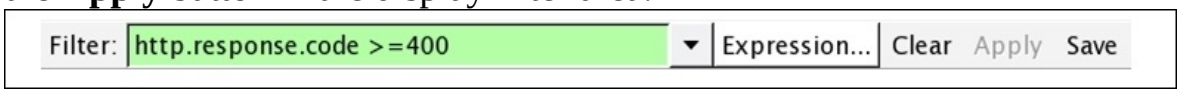


Figure 8.27: The display filter toolbar

13. Once you click on **Save**, you will be presented with a dialog. To provide a name for the button, specify any name of your choice and click on **OK**. Refer to the following screenshot that illustrates this:

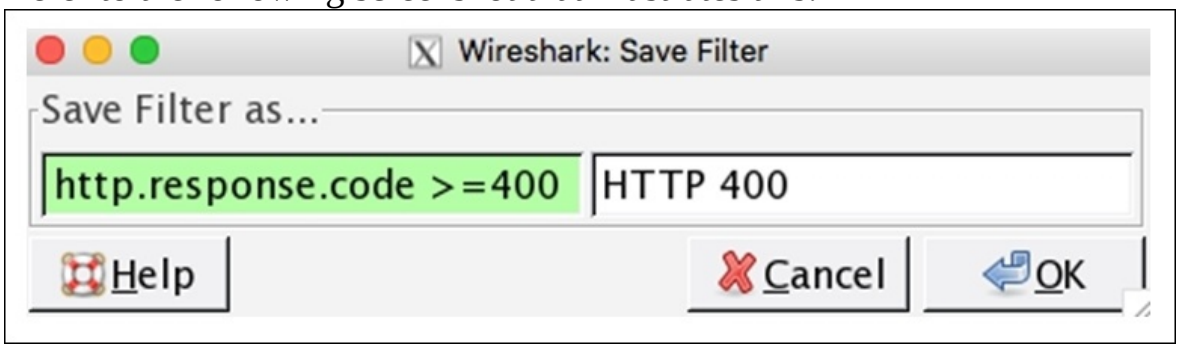


Figure 8.28: Creating a button

14. Once you click on **OK**, you will be able to see the button next to the **Save** button in the display filter toolbar area.
15. Now, whenever you want, you can create a similar display filter without typing it into the display filter box. You just need to click on the button that you created recently.

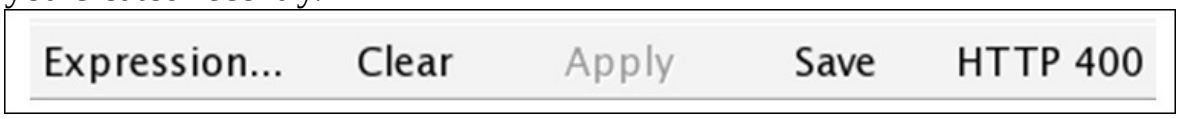
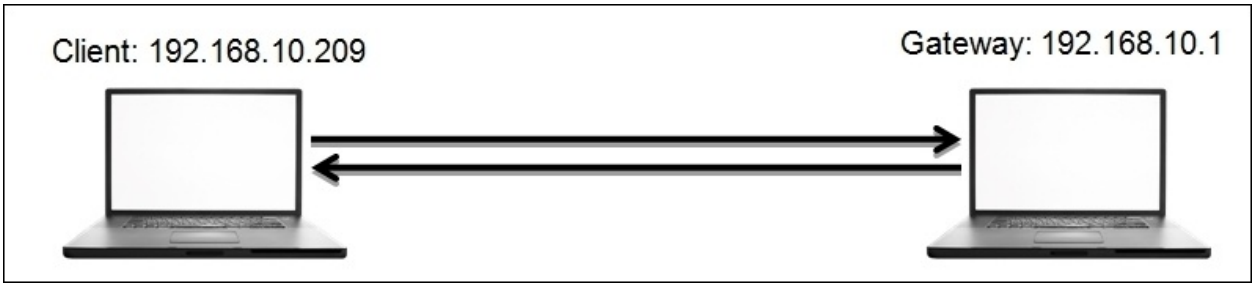


Figure 8.29: The newly added button

To make this more interesting, I would advise you to create a coloring rule for the **HTTP 404** error. This will definitely help you identify particular error types more conveniently.

Next, we will see another application protocol that is commonly used by various applications in order to translate a domain name to its IP address. Yes, I am referring to DNS. As we know, the DNS protocol runs over a UDP or TCP. There are various response code that relate to DNS errors that range from 0 to 21. The dissectors present in Wireshark do know about response code. Using this, Wireshark is able to show you messages relevant to the error code. To

replicate an error, I will visit a website that does not exist on the Web; hence, I will receive an error. But my gateway does not know about this, so it will try to resolve the IP address associated with that name. In return, we will see a DNS response containing an error. The infrastructure is the same that we used in the preceding examples. The client is located at `192.168.10.209` and the gateway is at `192.168.10.1`.

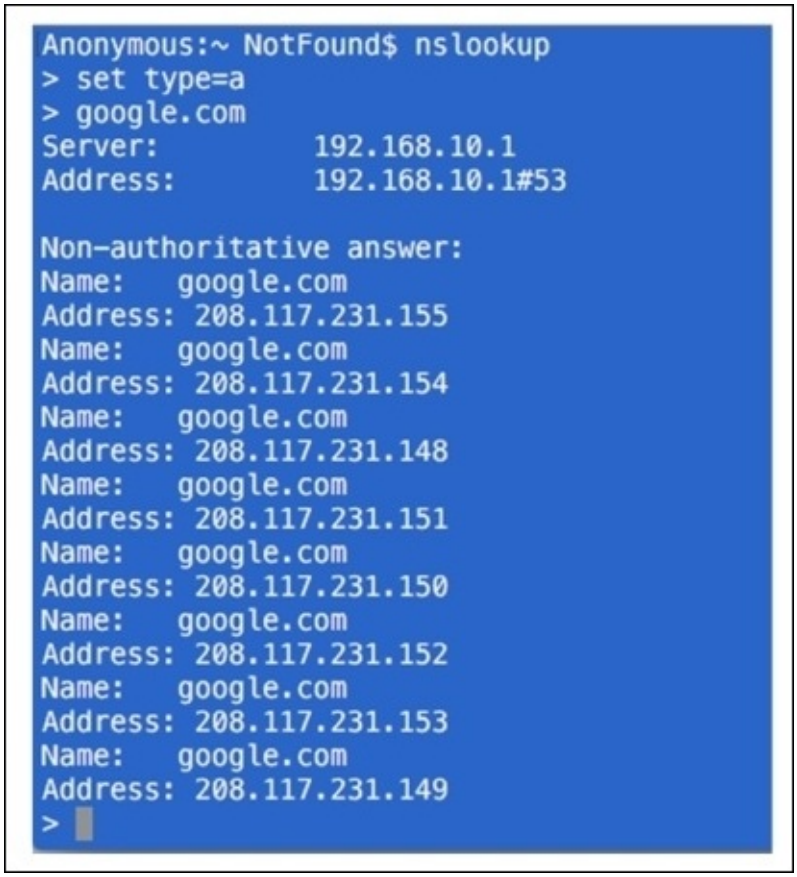


You can replicate the scenario step by step with me or do it later once you finish reading. Follow these steps to replicate the scenario:

1. Open Wireshark, and start capturing. Let it run in the background.
2. Open a terminal (Command Prompt) of whichever operating system you are using, type `nslookup` in it, and press *Enter*.
3. Now, you'll enter the interactive mode of the `nslookup` tool. If you are not aware of the tool, do read about it before you proceed. There are plenty of documents available for the tool. Refer to the following screenshot:

Figure 8.30: The NSLOOKUP tool

4. To generate DNS error response code, just type any domain name and press *Enter*. Before you specify a domain change the type of query to `A` by using the `set type=a` command and then give the domain you want.



5. First, we can try the same for a domain that exists, such as google.com. Then, you can try it for the nonexistent domain. Refer to the *Figure 8.31* shown here.
6. The preceding screenshot shows the various IP addresses that are associated with the google.com domain. The domain already exists. That's why we are able to see the reply. What if you try a domain that doesn't exist. Refer to the following screenshot that illustrates this:

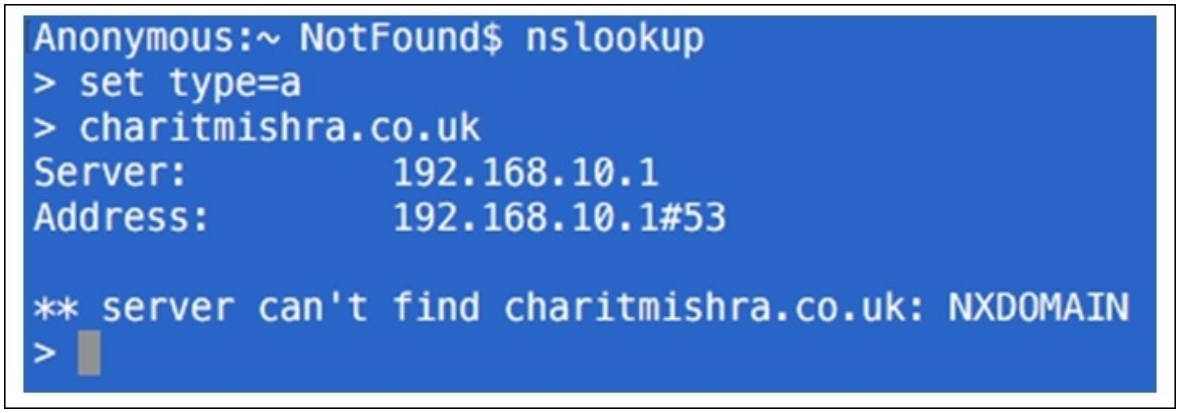


Figure 8.31: The nonexistent domain

7. I typed my name in place of the domain name and pressed *Enter*, and this is what I saw because there was no domain with that name. The DNS server was not able to resolve an IP address, hence resulting in the reply `server can't find`.
8. Now, you can go back to Wireshark and stop the capture process. We will now start analyzing error code.
9. The best option would be to segregate the DNS error response code from the normal frames in the trace file that we have. To achieve this, apply the `dns.flags.rcode == 3` display filter, which means that the shown DNS response frame with error code `3` is for nonexistent domains. For more information on DNS error code, visit https://tools.ietf.org/html/rfc2929.
10. Once you have applied the preceding display filter, you will only see relevant packets matching your filter expression.

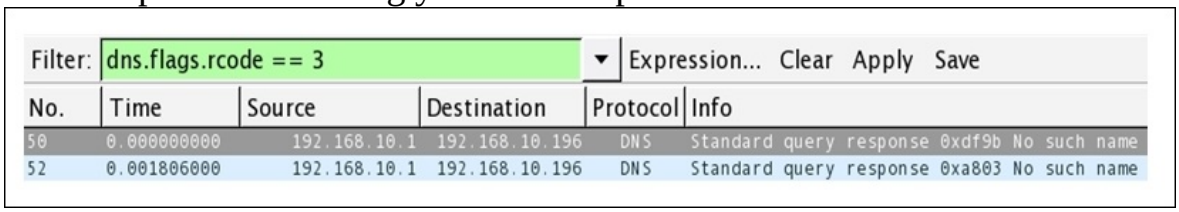


Figure 8.32: DNS error response

11. As you can see in the list pane, only packets that are related to error code `3` are visible.

12. If you want, you can save the filter expression in the form of a button for later use following the same approach we used earlier.

Troubleshooting application-based issues depends on how well you are aware of the error code. There might be a case that you can witness where you don't have the option of installing Wireshark for your assistance. You will be presented with error code for troubleshooting purposes. So I recommend that you at least know about the common error codes in the most popular application protocols that are normally used.

Summary

Troubleshooting is an art that comes with experience, and to become a master in it, you are required to practice things practically on your own.

There are various error recovery features that are provided by the TCP protocol that help us to recover from loss of packets that might happen commonly in a production environment.

TCP retransmission and duplicate ACKs are some of those techniques that are used by the TCP protocol in order to make the life of network administrators a bit more comfortable.

Slow network is one of those common problems that you have to face on a daily basis. Before you start solving these latency issues, you should know the basic methodology that you can follow, that is, to categorize your scenario in one of the latency categories: a wire, client, or server.

Solving bottleneck issues, such as packets getting queued up inside the sender buffer area and causing trouble, is quite important. The best approach in solving a bottleneck issue would be to take the help of IO graphs that you learned about in the earlier chapter to visualize a situation and get hold over it.

Applications use protocols such as HTTP and DNS. This is very common, but you must be aware of error codes these can present, and without using Wireshark, you should be able to identify the situation. I do not know every error code, even I can not do that. But the most common ones that you might witness.

Creating a baseline is one of the most convenient ways of dealing with issues in your network. When you have a trace file containing an optimized traffic pattern, then, by comparing the normal pattern with the deviated pattern, you can solve the issue in less time with few resources. Collect the network traces for your baseline from various locations in your network at least 2-3 times.

Practice questions

Q.1 Create a baseline from different positions of your network regarding various common protocols used in communication.

Q.2 Explain the various characteristics that TCP error recovery features have.

Q.3 Which protocols other than DNS and HTTP can be troublesome for you, and what approach will you follow in order to troubleshoot them?

Q.4 What do you understood by the term "bottleneck issues", and can they be ignored. If yes/no, why?

Q.5 Create a trace file for your own host and at least capture 10,000 packets. Then, analyze how many types of errors you are able to see for the HTTP protocols, and how many of them can you replicate.

Q.6 Using the baseline that you created earlier, try to match an unusual traffic pattern and observe what anomalies you can figure out by the comparison process.

Q.7 For the DNS protocol, replicate an error code other than 3 and capture traffic for the same.

Q.8 Prepare a checklist for the latency types we discussed and mention as many scenarios as you can think about in each category. Once you've prepared this, try using the same in a troubleshooting scenario. Does this speed up your overall process?

Q.9 Try creating coloring rules for error responses for various application protocols you want to and analyze what difference does it makes in the troubleshooting issue.

Chapter 9. Introduction to Wireshark v2

This chapter will introduce you to the amazing features launched with the latest version of Wireshark. The following are some of the prominent changes that users will become aware of, and all the sample examples in this chapter are being using version 2:

- Comparison between Wireshark v2 (QT) and the Legacy framework (GTK)
- The intelligent scroll bar
- The Translation feature
- Graph improvements
- Newer TCP streams
- USBPcap
- Summary
- Practice questions

Wireshark has been there with us for approximately two decades now; there weren't any major updates that we witnessed during its lifecycle. However, there were minor updates introduced to make the application more convenient and robust during this long period. But this time, we have a newly branded Wireshark v2 with glazing arsenal. Yes, we are really lucky to witness this major update for the most popular and amazing tool in the protocol analysis industry.

I am really excited to discuss the different sets of tools introduced with the latest release, but, before that, it is necessary that you get acquainted with the background of the QT and GTK frameworks. You definitely have to Google these either now or maybe after reading this chapter. However, make sure that you note them.

For your convenience, I will give you a gist and some background of these frameworks; the reason why I am emphasizing the difference between the two is that the newly developed version 2 of our protocol analyzer is developed using the QT framework. QT and GTK are frameworks used for the development of GUI cross-platform utilities such as Wireshark. In general, from the end user's perspective, the difference would be based purely on graphical changes, but

performance wise, GTK is more economical as compared to QT. For better understanding, these aren't just toolkits and frameworks; instead, these are sets of libraries used by developers to create better GUIs for end users. Basically, it's reusing the designs already made by others. The main advantage of reusing designs is that it allows the newly installed program to look more similar to the other already installed programs on your machine. For instance, let's see both the new and old version of the application parallelly; refer to the following screenshot for this:

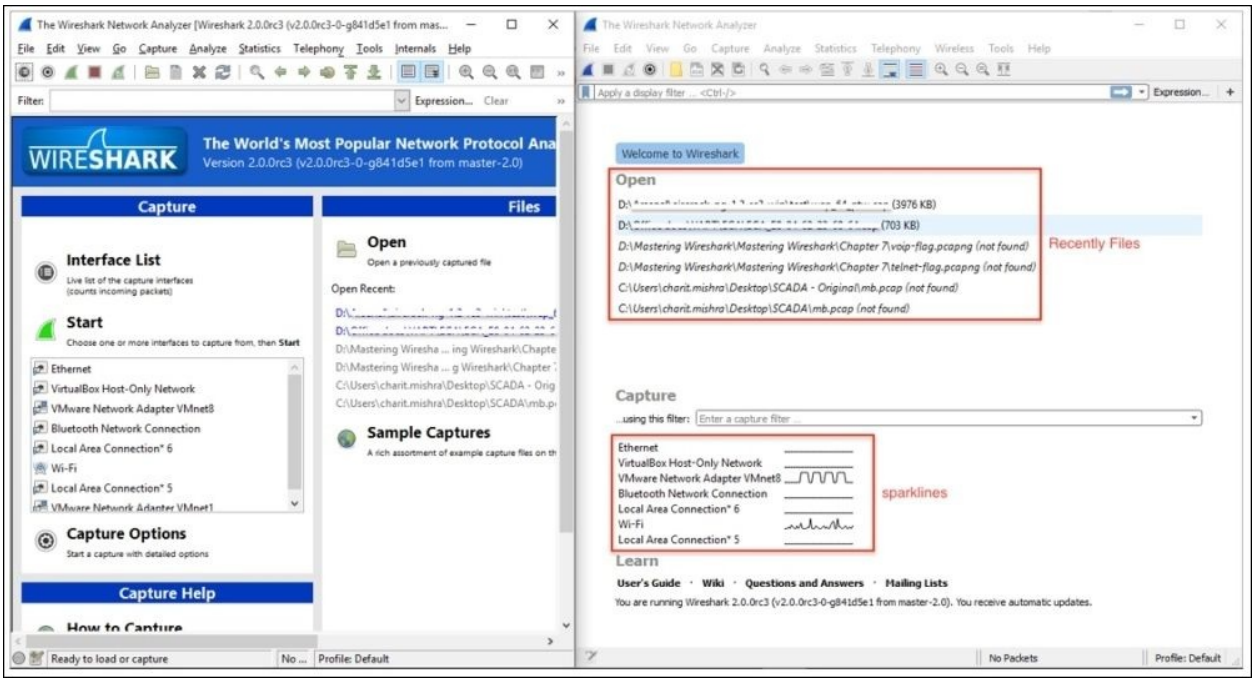


Figure 9.1: The GTK and QT frameworks

You must be wondering how you can get your machine installed with the latest version of Wireshark. It's really easy; you just have to visit http://wireshark.org, and then go to the download page. There, you will find the latest release. Download the one appropriate to your operating system. During installation, there is one important question that you will be asked, that is, whether you want to install the legacy version along with the newer release or you just want to install the newer version (note that only Windows users have this privilege; Mac and Linux users can just install the latest version of the application).

There is one more component that you will see being installed on your machine: USBpcap. I have dedicated a separate section in this chapter for this particular topic. For the sake of basic introduction, USBpcap facilitates users to capture data that moves back and forth from your machine's USB port. The tool has been available for Linux users for quite a long time, but luckily, Windows users can also utilize this now.

For starters, let's have a look at the main screen , which has a completely different feel from the previous version. Refer to the following screenshot to get a look:

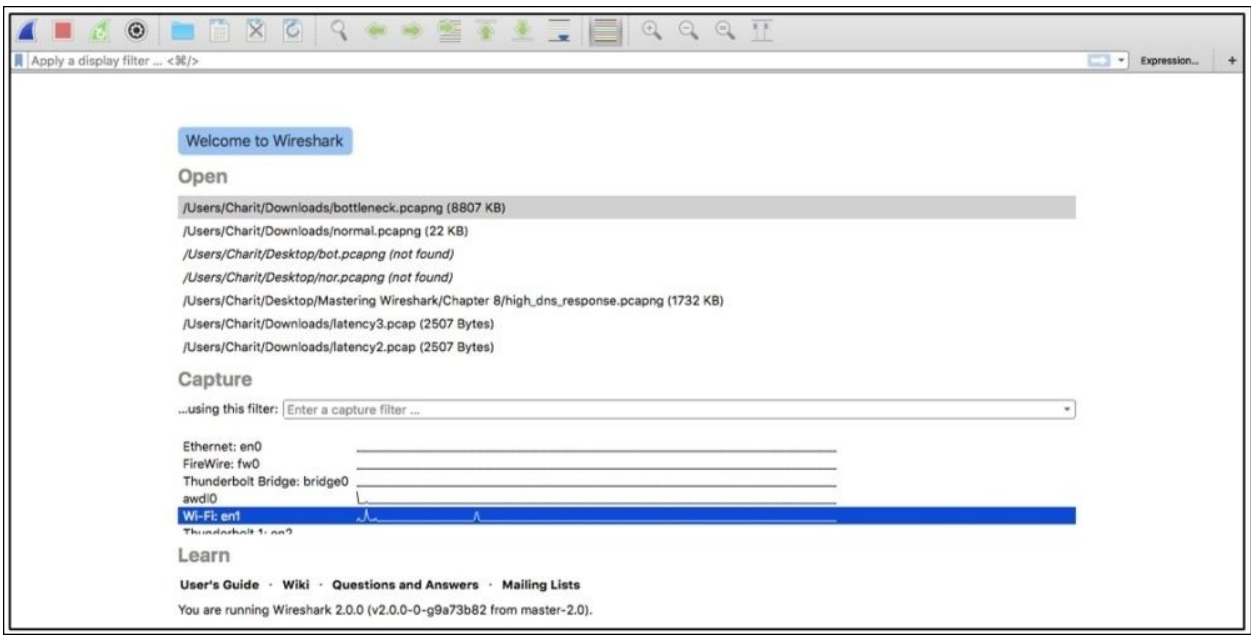


Figure 9.2: The main screen of Wireshark v2

I hope you feel the same way I do about the new, exciting look. Everything in this version looks so properly arranged and cleaner. Even a novice user who has no experience at all in protocol analysis can get a great head start just because this has now become a simple and attractive interface.

Just observe the toolbar area, for instance. In this version, it seems like the developers have filtered out the unwanted and less commonly used tools, which eventually makes the interface quite comfortable for the eyes. In this new

version, we have quick access directly to a basic toolset, such as the start and stop capture buttons, the interface customization button, a button to save/open/close the current capture file, some navigational tools, and the auto scroll and coloring activate/deactivate button.

Just below the toolbar area, we have our good old friend, the **Display Filter** toolset, which is redesigned with great efforts. On the leftmost side of display filter text box, you will see a bookmark kind of icon (in blue—top-left corner) that will show you the default and manually created filter expressions. Refer to the following screenshot that shows an illustration:

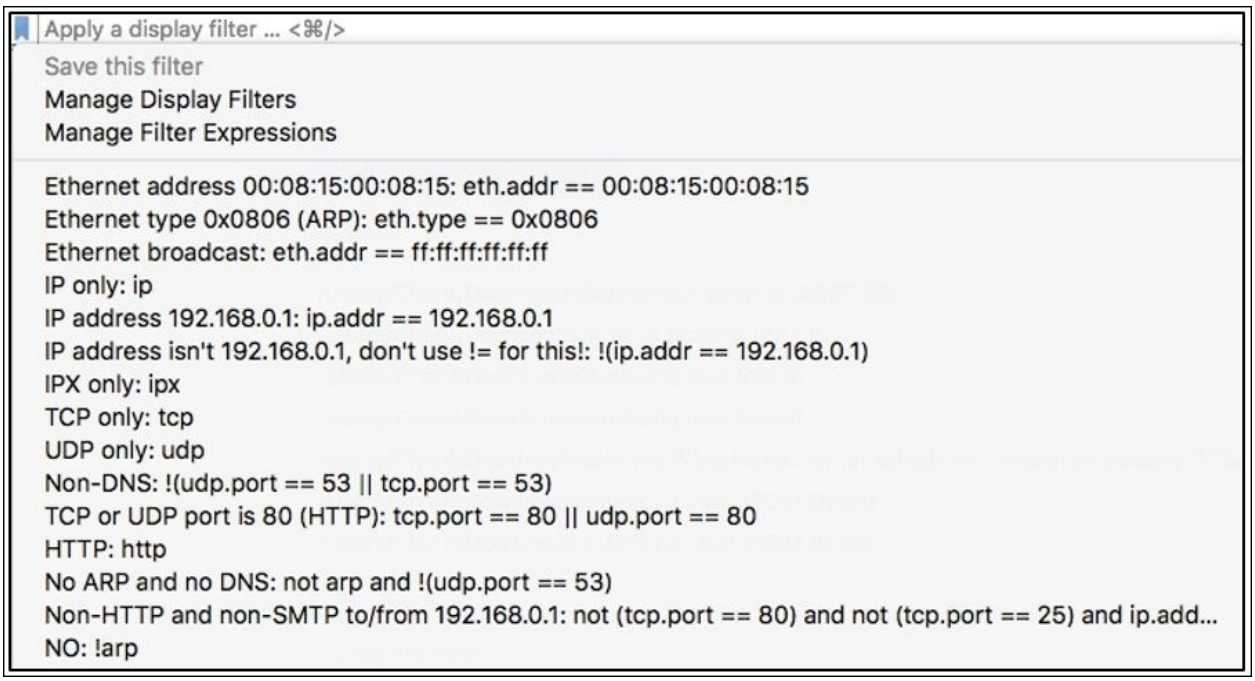


Figure 9.3: The Display Filter toolbar

As you can see, all the filters are listed, which you might have created, or are default ones. So now, it's a matter of just a click if you want to activate any one of them, instead of getting a pop-up window from where you choose and apply the filter, like in the older version. This definitely speeds up the process of analyzing and makes the life of IT professionals easier.

On the other end of the **Display Filter** toolbar, we have a few old tools that have

been remodeled in a fresh look, along with some functionality improvements; refer to the following screenshot for an illustration:

Figure 9.4: The Display Filter toolset

To apply any display filter now, you just need to click on the arrow, and the dropdown next to it will give you access to frequently used filter expressions (history of last-used expressions). Then, you have the **Expression** button, which will help you access the dialog where you can get access to all possible filter expressions categorized on the basis of protocols. Next, on the rightmost side of the display filter textbox, you have the + sign; by clicking on this, you can create a filter button. Let me help you in creating one for yourself in the newer version to get started.

For example, I want to create a button to see only the ARP packets, so I will type `arp` in the display filter area and click on the + sign at the end of the toolbar. Then, you need to specify the name of the button you want:

Figure 9.5: Adding a custom display filter expression button

This will add a physical button next to the + sign. This technique will prove worthy and very effective when you have long display filter expressions, which you might need often. So, instead of typing the whole expression again, you can just activate them with a single click. As a result, you will see something like what is shown in the following screenshot. Now, you are just a single click away from applying **arp** as the display filter:

Figure 9.6: The display filter button created

Next, below the display filter toolbar, you can see the recently used files; just double-click on any file you want to open.

After the **Open** file section, we have the capture filter toolbar, and I don't think you need any explanation regarding what it is for and how you are going to use it for your perusal.

Now comes the major change that you will witness on the main screen, that is, the interface's name followed by an interactive graph. The graphs you will see are actually live, meaning you will see the fluctuations, that is, the lines going up and down. The miniature graph followed by the interface name represents the amount of traffic moving back and forth from the interfaces you have. The proper terminology for these miniature graphs is sparklines. In the older legacy version, we had the live statistics in numerical form.

Now, if you decide to capture traffic from a particular interface, just double-click on the graph area, and Wireshark will do the rest for you.

The intelligent scroll bar

This is one of the features launched in the latest release, and you might have already noticed some colored sections/lines in the scroll bar area. If not, then go back to any of the capture files you have, slowly scroll up and down, and observe the coloring pattern in the scroll bar area. Any guesses what difference it would make in the analysis process? Let's understand this with an example.

I will use a previously captured file for demonstration purpose, which has HTTP and HTTPS packets along with some retransmission and duplicate frames. There is no difference that you can figure out at first glance, but as soon as you start scrolling, the coloring pattern will be shown in the scroll bar area. This pattern is based on the coloring rules that you have in your application. For example, as per the default coloring rules, duplicate and retransmission packets are usually seen with a black background and a red foreground, and HTTP packets are shown with a green background and a black foreground. Now, let's verify this in the application itself. Refer to the following figure for the same:

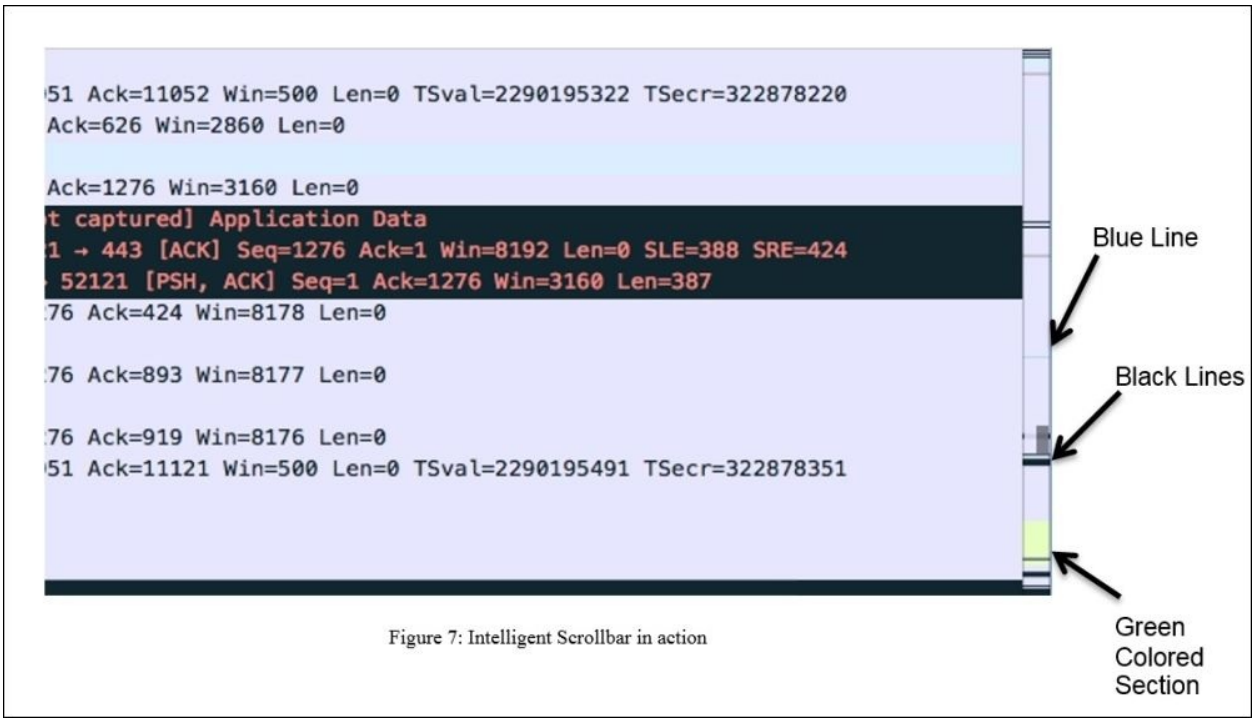


Figure 9.7: The intelligent scroll bar in action

The way packets in the list pane are shown in different colors is similar to the way the scroll bar represents the different sections of your list pane.

In the same way that the blue line indicates the selected packet, the black lines denote the duplicate ACKs and retransmissions, and the green-colored section indicates that at the bottom of the capture file, we have some HTTP packets listed. By just observing the coloring pattern in the scroll bar area, we can figure out what sort of packets we have ahead, and most importantly, navigating to a certain section of packets you are looking for is now much easier and faster.

We already discussed customizing the coloring rules in previous chapters; let's take one more example of the same capture file, and this time, I want to customize the HTTP packet coloring rule. We will change the green background color to yellow. Let's see what difference it would make in the scroll bar area in the following screenshot:

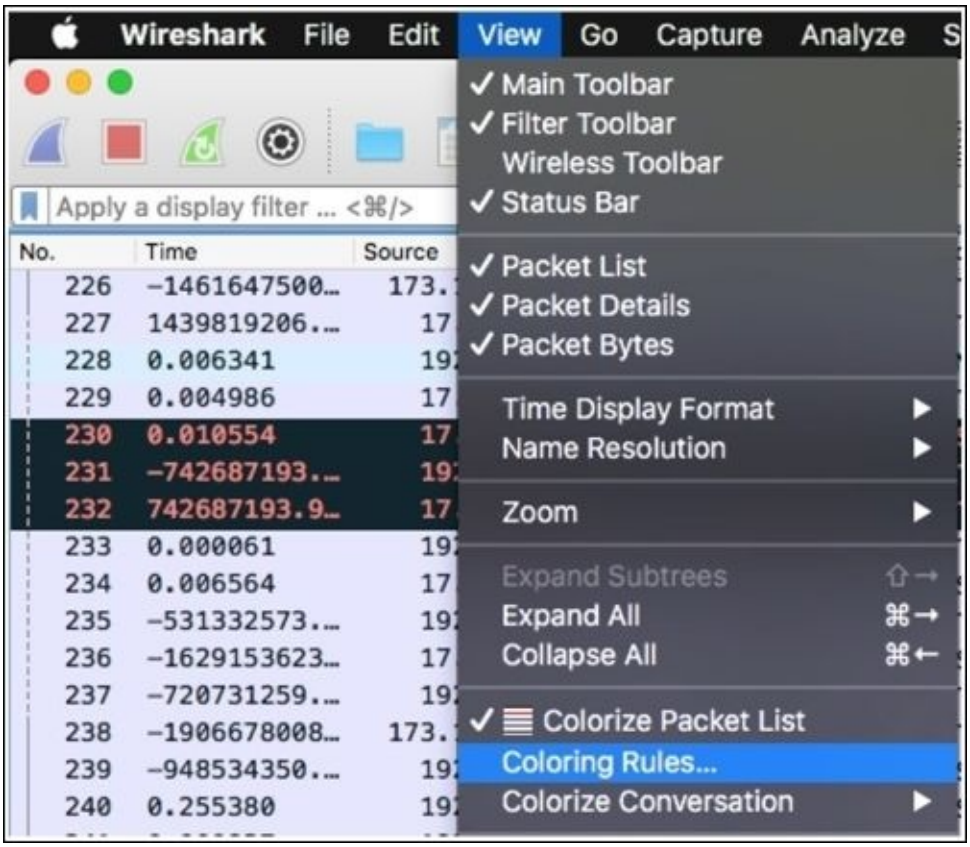


Figure 9.8: Accessing the coloring rules dialog

To access the coloring rules, you need to click on **View** from the menu bar and then choose **Coloring Rules** at the bottommost corner, which will show you the dialog where all coloring rules will be listed. Try changing the HTTP coloring rule to yellow. Once this has been done, close the dialog and reopen the capture file in order to see the change.

Now, try scrolling the same file, and I hope you will see the difference in the coloring pattern in the scroll bar and your list pane too, where all HTTP packets are colored with a yellow background. Refer to the following screenshot:

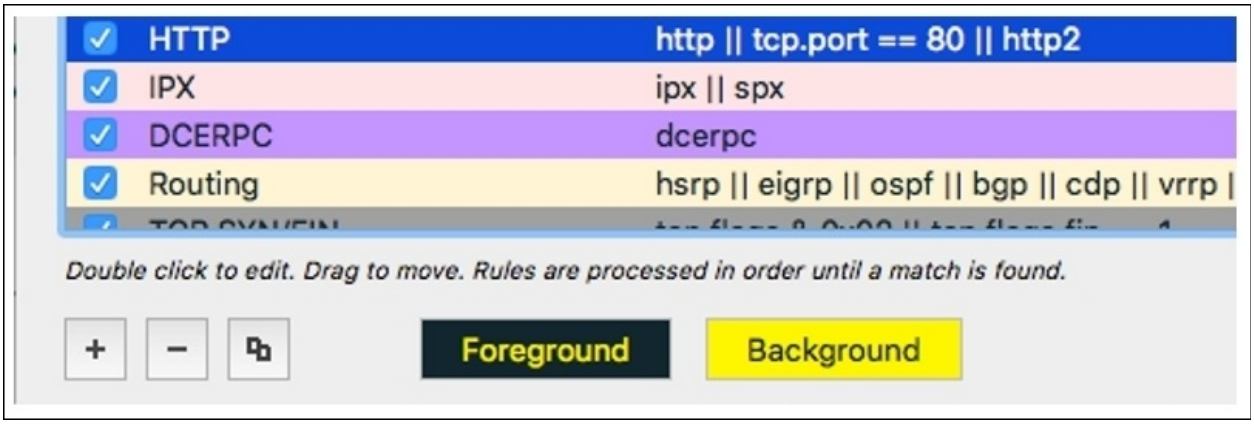


Figure 9.9: The HTTP coloring rule

Now, let's compare what difference it made when we tried scrolling up and down in the list pane after the new coloring rule was applied. Refer to the following screenshot to go through the illustration:

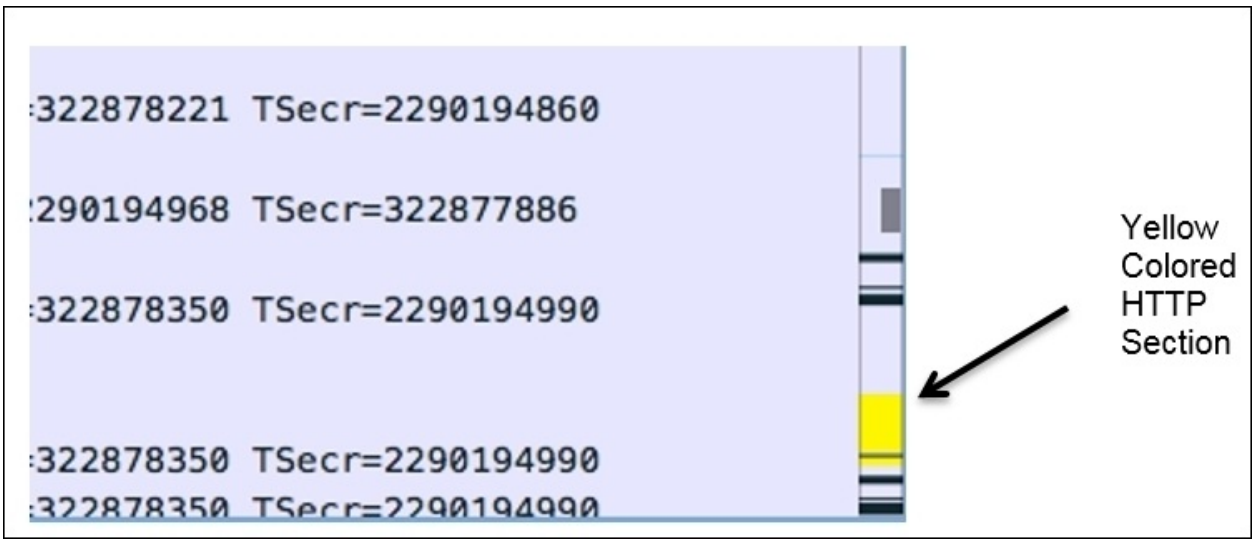


Figure 9.10 Effect of the HTTP coloring rule can be seen in the scroll bar

A good amount of cleanup has been done from the toolbar area where, for example, the **coloring rules** toolset has been removed, and now you can access it from the **view** menu. The **+** and **–** symbols at the bottom of the coloring rules window can facilitate you with the configuration of the rules.

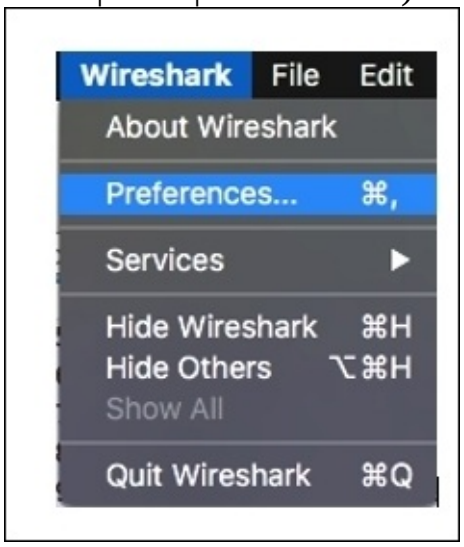
Translation

I think this amazing and pretty cool feature is not able to gain limelight, so I want you to know that Wireshark offers you to change the language to any other language of your choice, for example, Spanish, Japanese, Chinese (Mandarin actually), Polish, French, and so on, and this feature has been there their since version 1.99.

Giving the privilege to users to change the default language of the application to their native language is all about personalizing user experience while working with the application. If users feel more connected and comfortable with the application, then they will definitelybecome more productive.

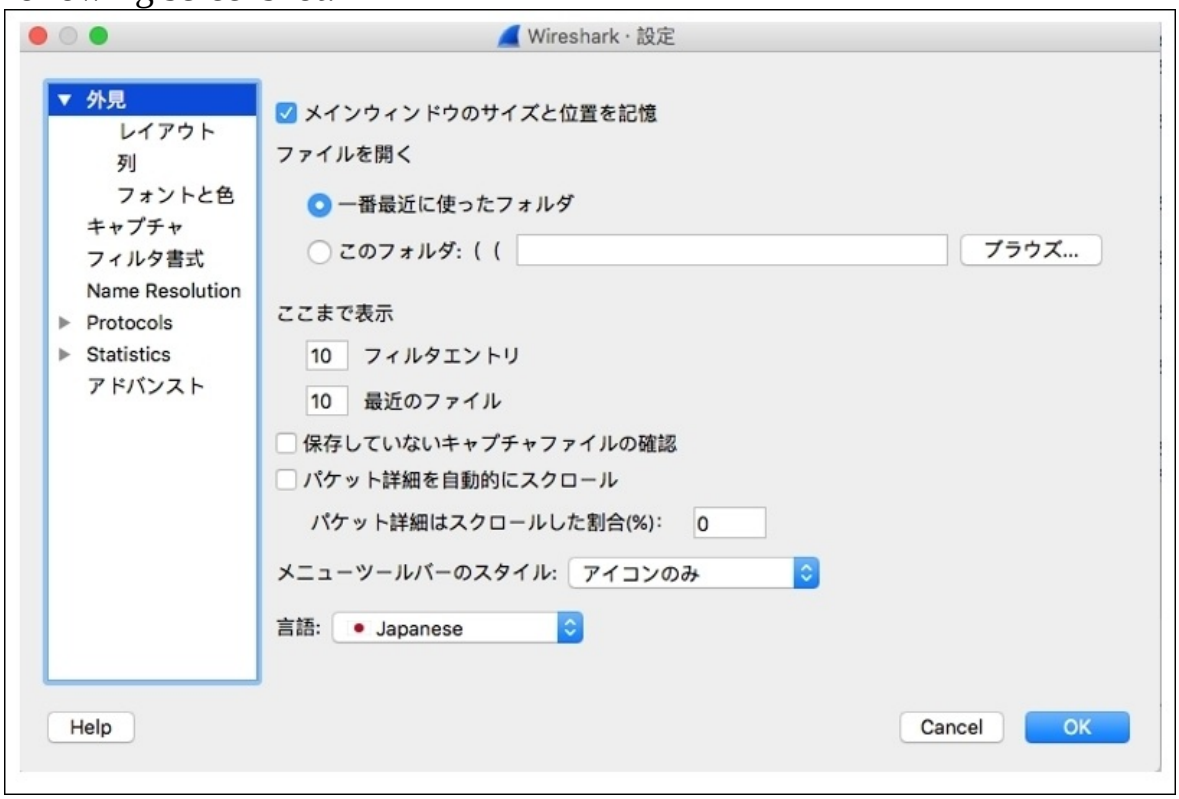
Let's see, with the help of an example, how we can change our system's default language to Japanese (launched with version 2.0). Follow the given steps to achieve the same:

1. Navigate to **Wireshark** | **Preferences** (Windows users need to navigate to **View** | **Edit** | **Preferences**):



2. Now, choose **Japanese** from the drop-down list at the bottom, and click on **OK**:

3. Now, you probably will see everything in Japanese, as shown in the following screenshot:



4. To revert it back to System Default, follow the same steps.

The most amazing thing about this is that you can also become part of the change; this means that if you want to help Wireshark's team in adding your native language, then you can get in touch with them.

From the help menu, you can list all the keyboard shortcuts, which can be used to make things work faster than usual. Even to make graphs, now you have a shortcut available.

Graph improvements

This is something that you will be really pleased to know about. Yes, Wireshark has made quite significant changes that will make your analytical tasks more comfortable. To understand the difference, the best option will be to go through an example.

We will try to create an IO graph in order to witness the changes that the new version has. I am using a capture file from the previous chapter, which has mixed packet types and mostly contains VoIP traffic. The sole purpose of this exercise is to see how graphs can be of better assistance in version 2 of Wireshark. Follow these steps to create an IO graph in Wireshark version 2.0:

1. Capture the normal traffic from your network or open any previously captured trace file that you have.
2. Click on **IO Graph** under **Statistics**. Once you do that, you will be directly presented with a graph without any further hassle:

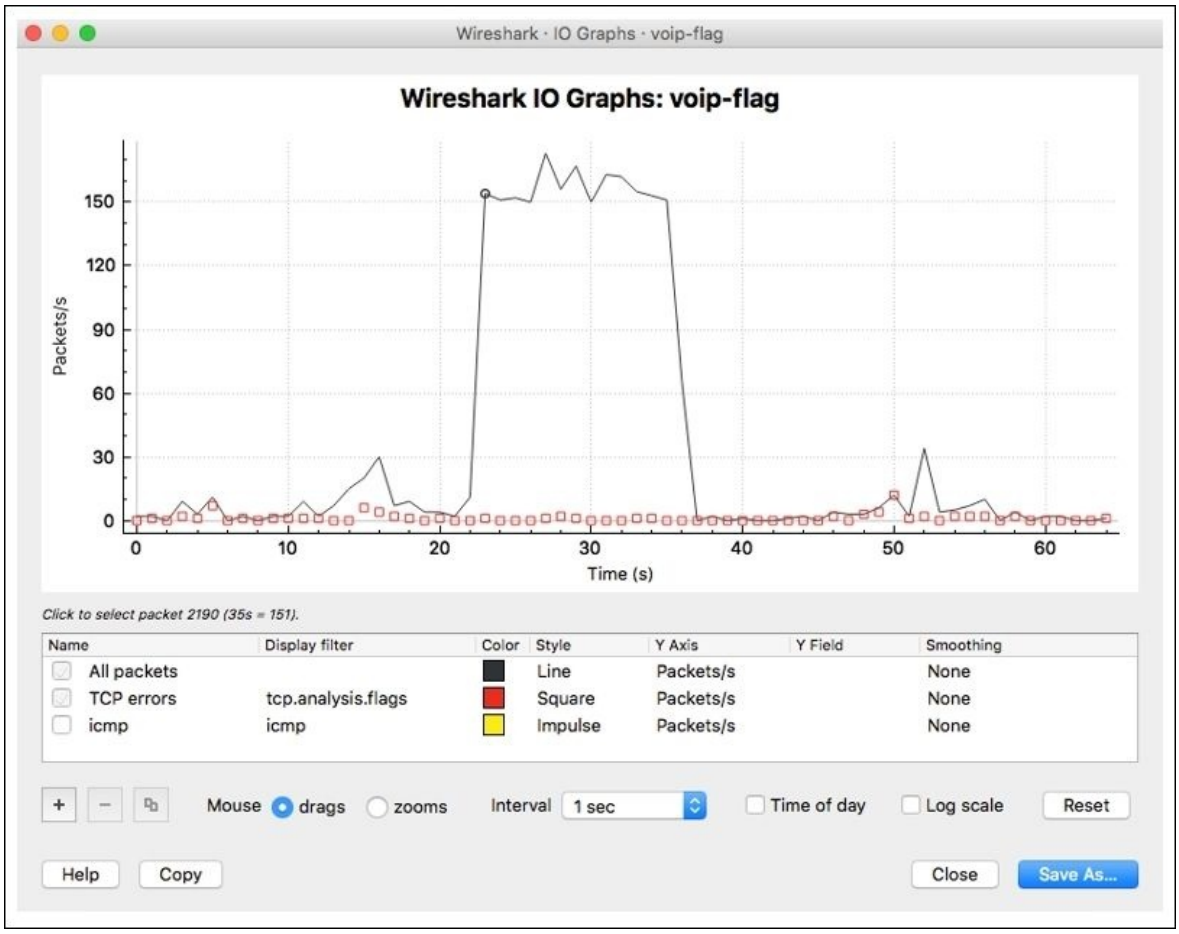


Figure 9.11: The IO graph

3. Now, if you want to modify and configure the graph, then you can use various configurable options given at the bottom of the dialog.
4. For instance, if I want to add any filter to the graph, I can click on the + symbol at the bottom and a new line will be shown, as in the following screenshot:

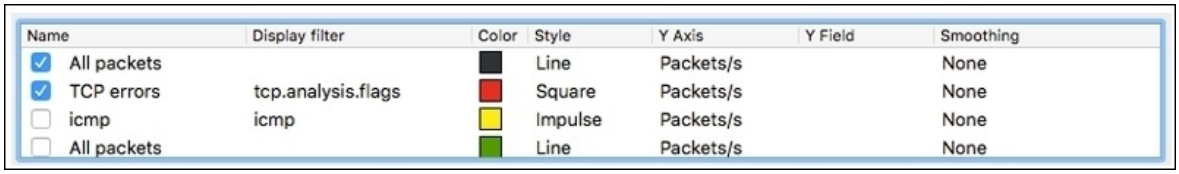


Figure 9.12: Adding a filter to a graph

5. Now, I want to see the traffic pattern for the ARP packets along with other traffic-related details. So, I would write `arp` as a filter expression in the display filter column and **ARP packets** in the name column. If you want to customize the look and feel too, you are most welcome to do so.

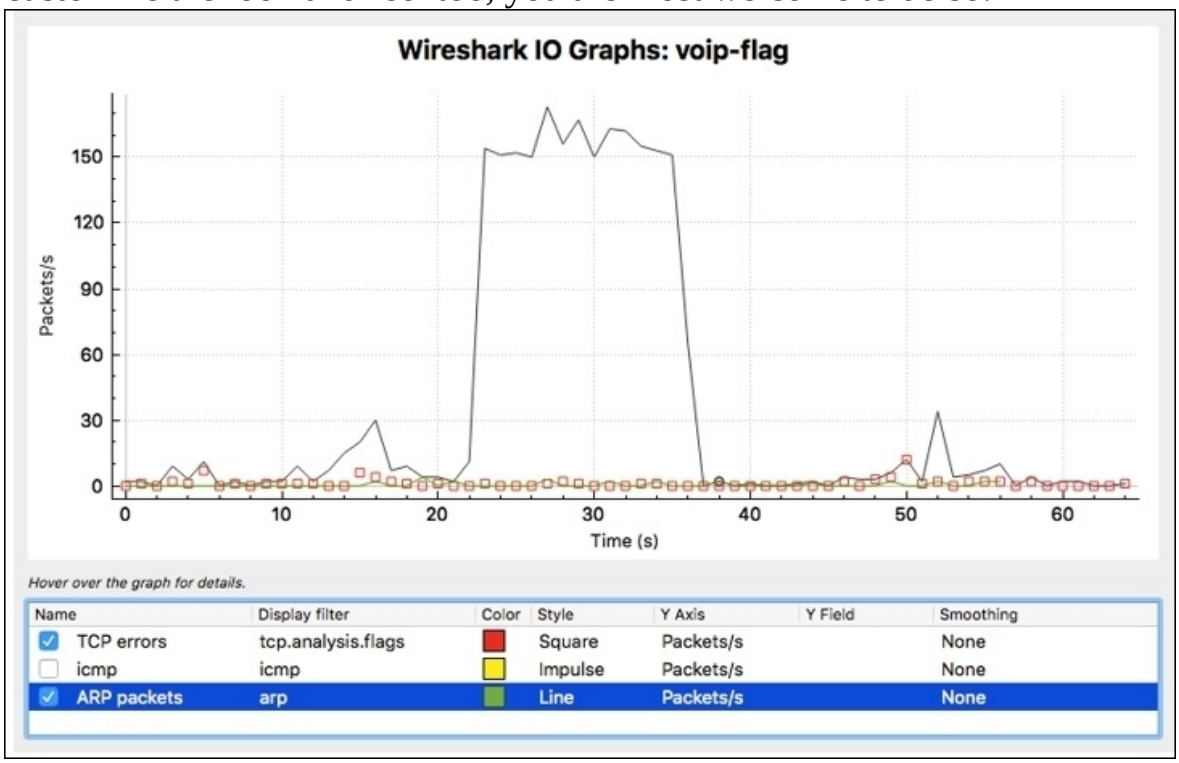


Figure 9.13: The ARP filter added in the IO graph

6. As you can see, our newly created filter is in effect, and we can observe the frequency of ARP packets appearing in our graph as well.

Using graphs is now much more convenient, as you are no longer required to pass any statistical information to the graph. Just choose whichever graph you want, and then the default version of the graph will be presented to you without any questions asked. Now, if you feel like changing the graph as per your need, then just use the toolset given at the end of the graph to custom configure it.

Now, after we have made an IO graph, you will see how clean it looks; there are lots of features that have been introduced. Using the default graph, most of the time you will be able to figure out the ups and downs in your trace file. The

legends are shown at the bottom most in a separate section, along with other configurable options like changing colors, hiding or enabling a filter, and much more.

Additional features can be listed and explored in the graphs; all you need to do is right-click on the graph area. The graph can now be moved along with the **x** and **y** axis by just clicking and dragging. Adding new arguments to the graph couldn't be any easier than this. As you can see, so many new amazing features are waiting for you to discover them.

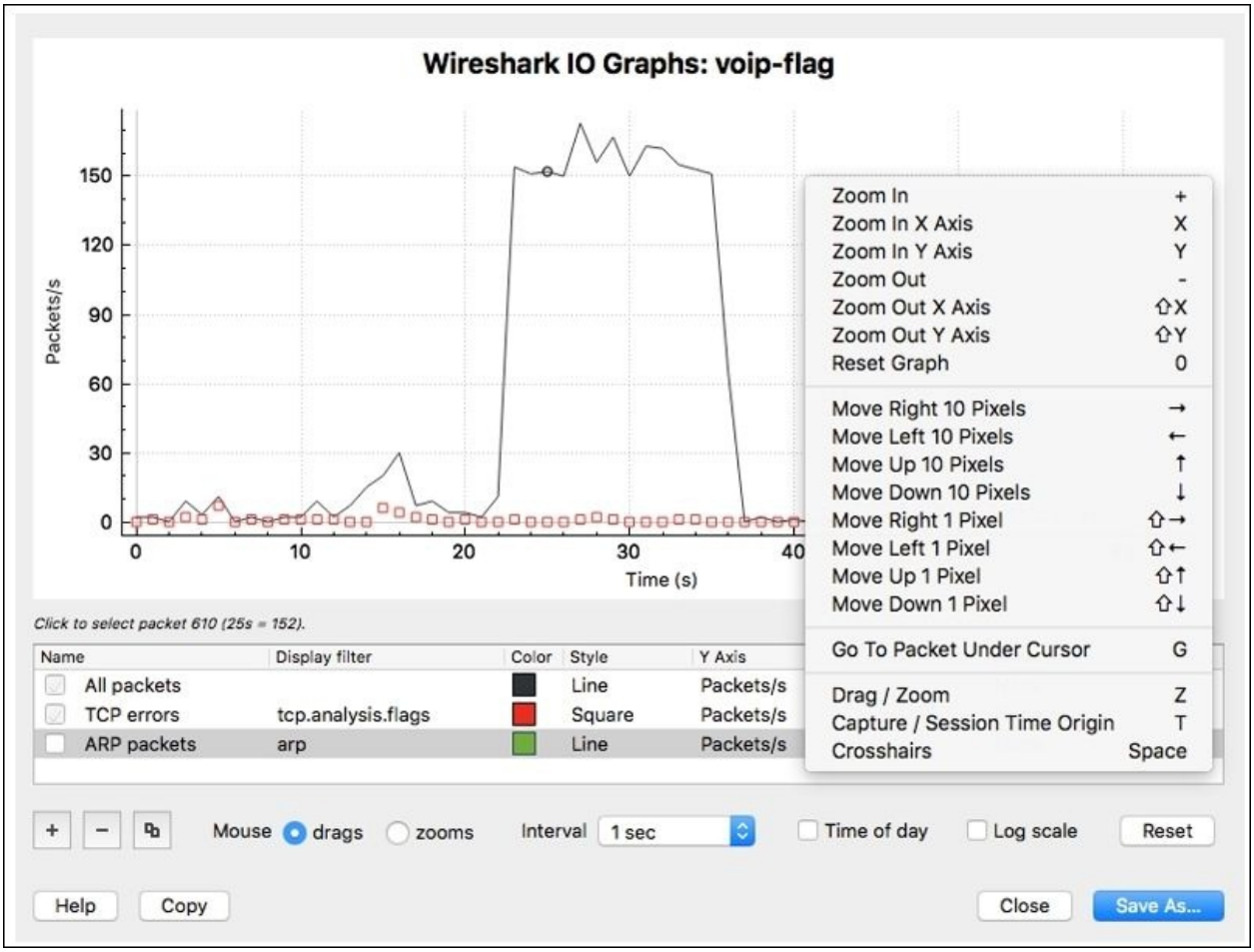


Figure 9.14: The right-click options list

Opening two graphs is now possible; and maybe someday, you will feel like comparing the traffic patterns in two trace files that you have. For example, I

want to compare the normal VoIP traffic pattern and the malicious traffic pattern. Then, we can use two graphs to figure out the difference graphically, and it's really effective. Refer to the following screenshots:

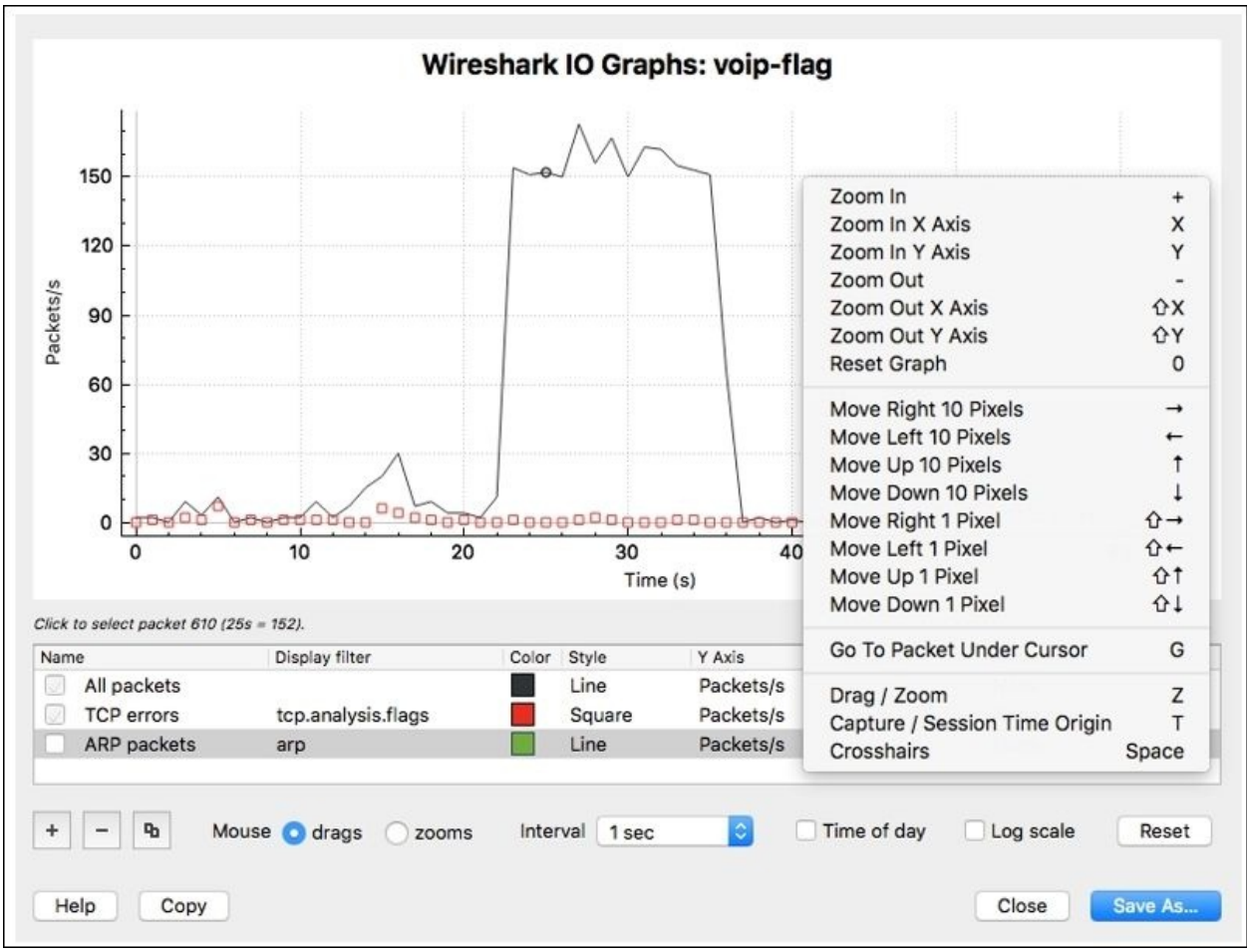


Figure 9.15: Comparing two graphs at a single instance

Similarly, you can create a flow graph that can be of great assistance while analyzing the TCP flow and to know how `SYN` and `ACK` coordinate with each other. I would highly recommend that you create the flow graph in the newer version of Wireshark.

To switch between the graphs, you have the drop-down list sitting at the bottom-left corner of the graph window, which can assist you in doing so, and you are no longer required to go the window in the background to switch between graphs.

Another useful feature that can be taken advantage of when you are trying to create reports for your client or maybe for your own reference purpose is to export the graphs in PDF formats. You might have done this before; if not, then let's do this together here. Follow the given steps to do so:

1. You need to click on the **Save as** icon at the bottom-right corner in the graph dialog window. Now, choose the location where you want to save the PDFs and click on **Save**.
2. Once this has been done, you can export the PDF to anywhere you want to. Refer to the following screenshot:

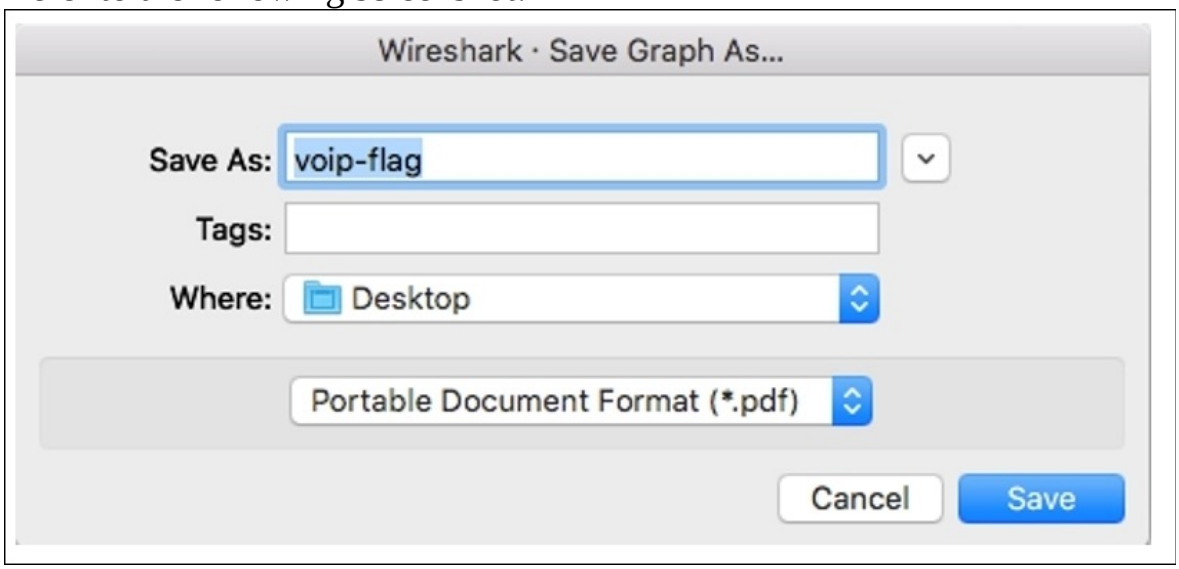


Figure 9.16: Exporting graphs to PDF format

Now, whenever you want to import it into your report, just add it like an image and the graph from the PDF you exported will be added to your document. Doing this is really this easy:

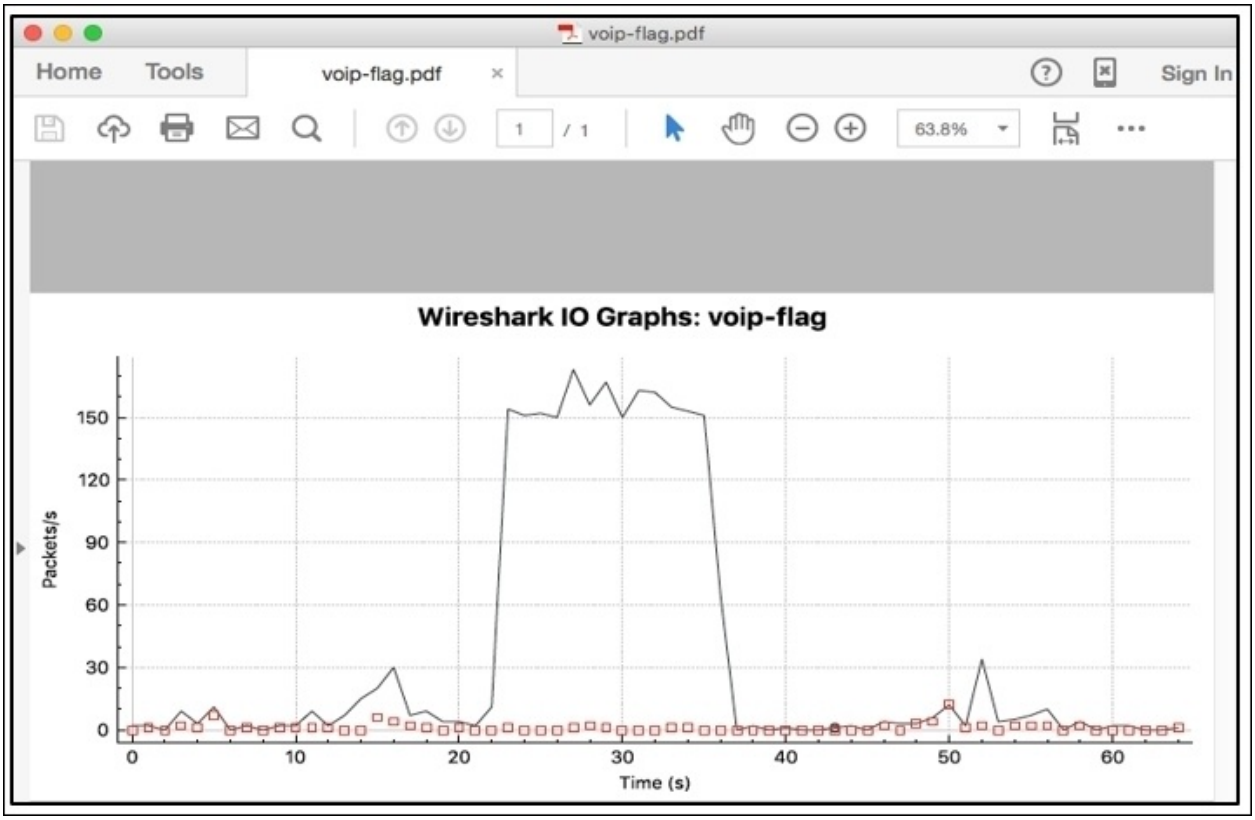


Figure 9.17: The graph exported as PDF

TCP streams

This is one of the features that you might have used very often so far, and I suppose the story will be same for all IT professionals using Wireshark as a utility. The gist of the tool definitely will remain the same in the next version, which is going to come in the future; however, there are some new things that I would like to emphasize. To view the TCP stream window, the process remains the same as usual. Right-click on the list pane area and choose **Follow** by hovering your mouse over it, which will the present available different streams. Then, click on **TCP Stream** options. Refer to the following screenshot to see these steps:

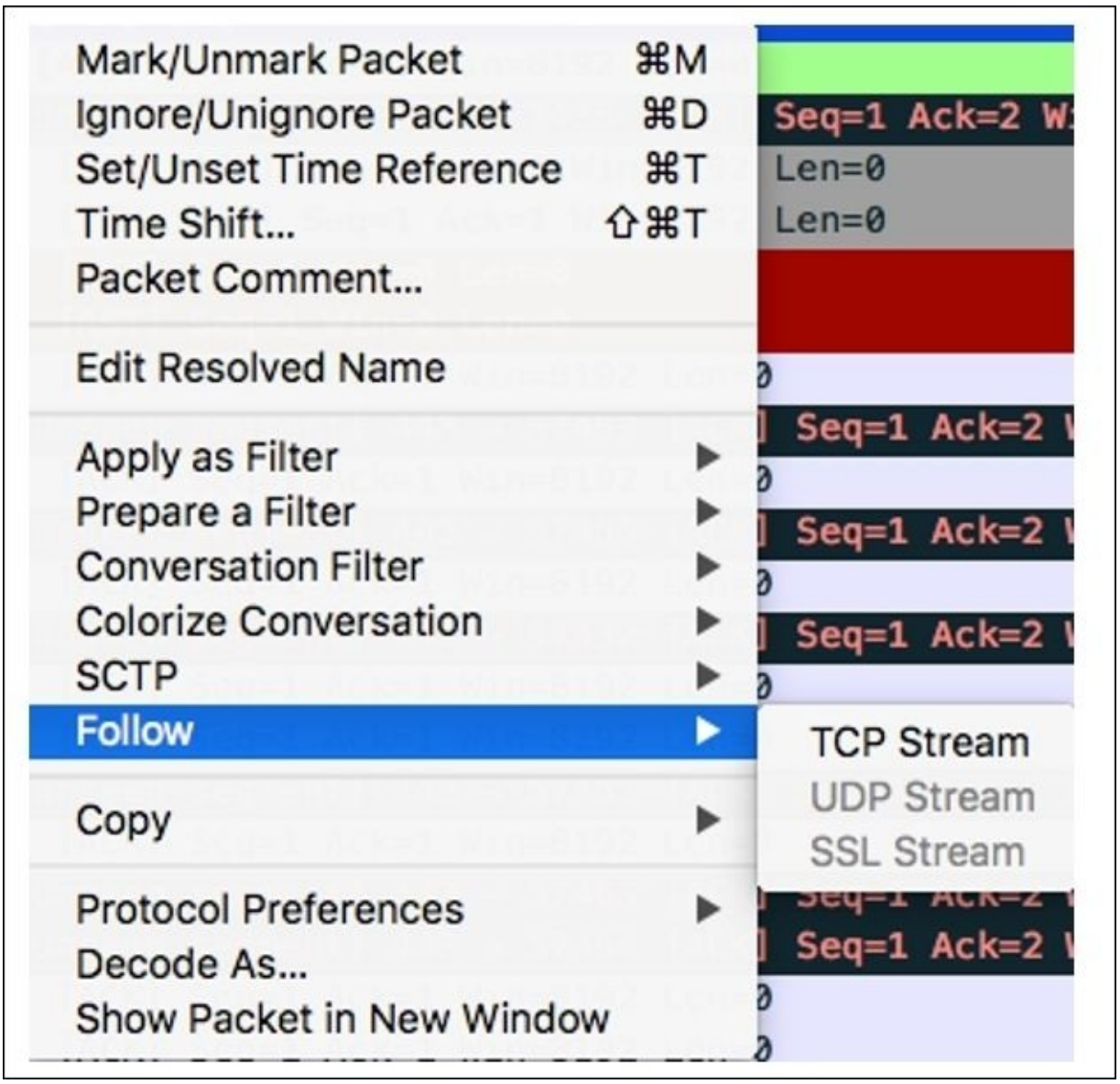


Figure 9.18: Follow TCP streams

Following this will present you with a usual-looking stream window similar to what we have seen in our previous chapters. However, we definitelyhave some new features to discuss, such as the flexibility of moving back and forth between the different TCP/UDP streams available, and the **find** utility that lets you search in the stream window for any text.

First, we will see how you can traverse in between the different streams

available in your trace file. Then, we will try to search some text through the follow streams window. Refer to the following **Stream** option screenshot that can be used to traverse between various TCP streams available:

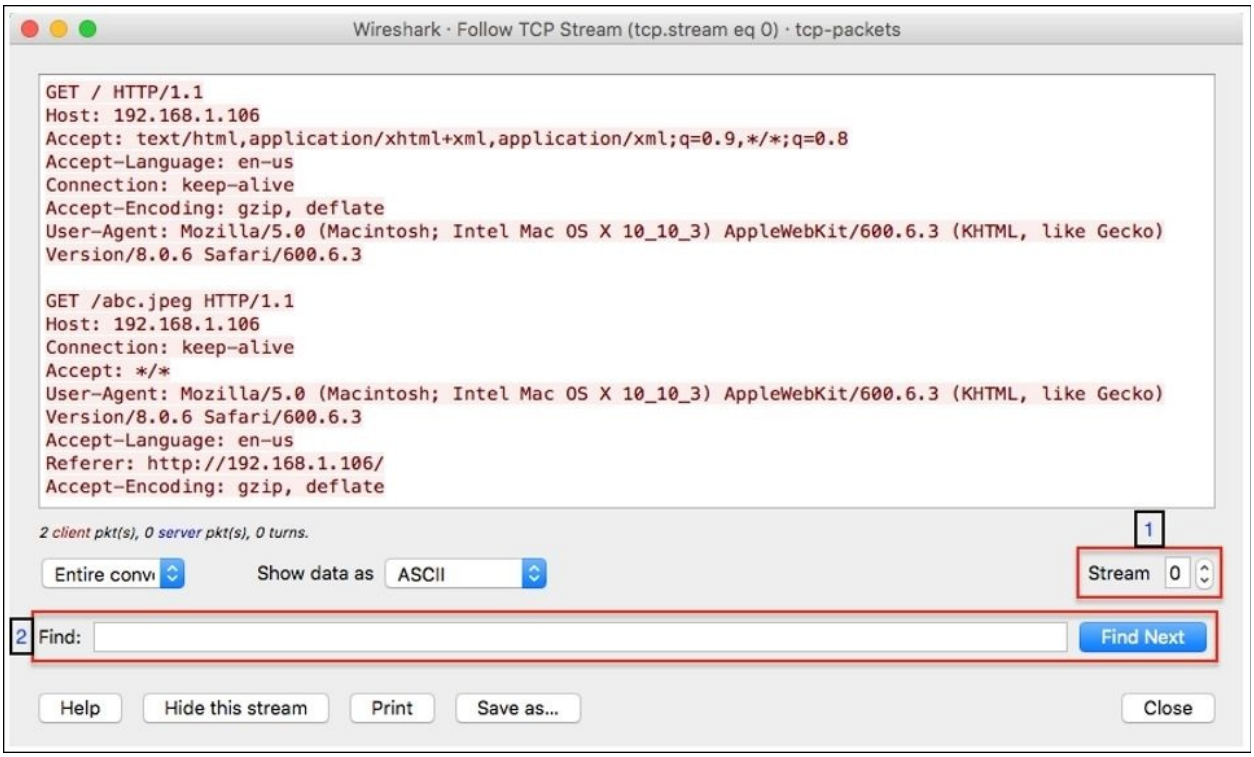


Figure 9.19: Follow the TCP Stream dialog

The stream option labeled (**1**) at the bottom-right corner of the preceding dialog gives you the flexibility to move back and forth between the different streams available. You have two choices here: you can specify the number of the stream you want to look at or you can traverse up or down by clicking on the up/down arrow followed by the textbox. So now, if you are looking for a different stream, you don't have to close and reopen the dialog, like we did while working with the earlier version of the application. Refer to the following screenshot:

Figure 20: The Stream option

The part labeled (**2**) gives you the facility to find any ASCII text inside the Follow stream dialog, which definitely gives an extra mile advantage for every person actively using this beautiful application. Most of the time, when we are using the stream dialog, it is for analytical purpose, and with these new features, our job becomes more easy and interesting. Refer to the following screenshots for reference regarding both the newly introduced options:

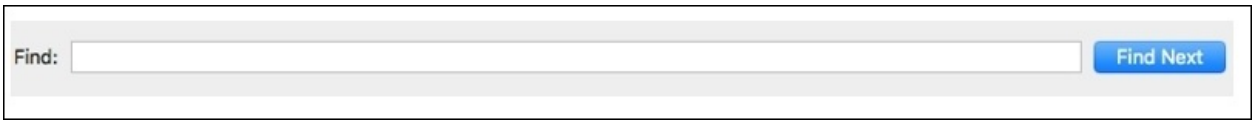


Figure 9.21: The Find utility in the Follow TCP stream dialog

For example, if you want to search for the text `abc` in the current stream, then just type the search string in the find textbox and press *Enter* or click on **Find Next**.

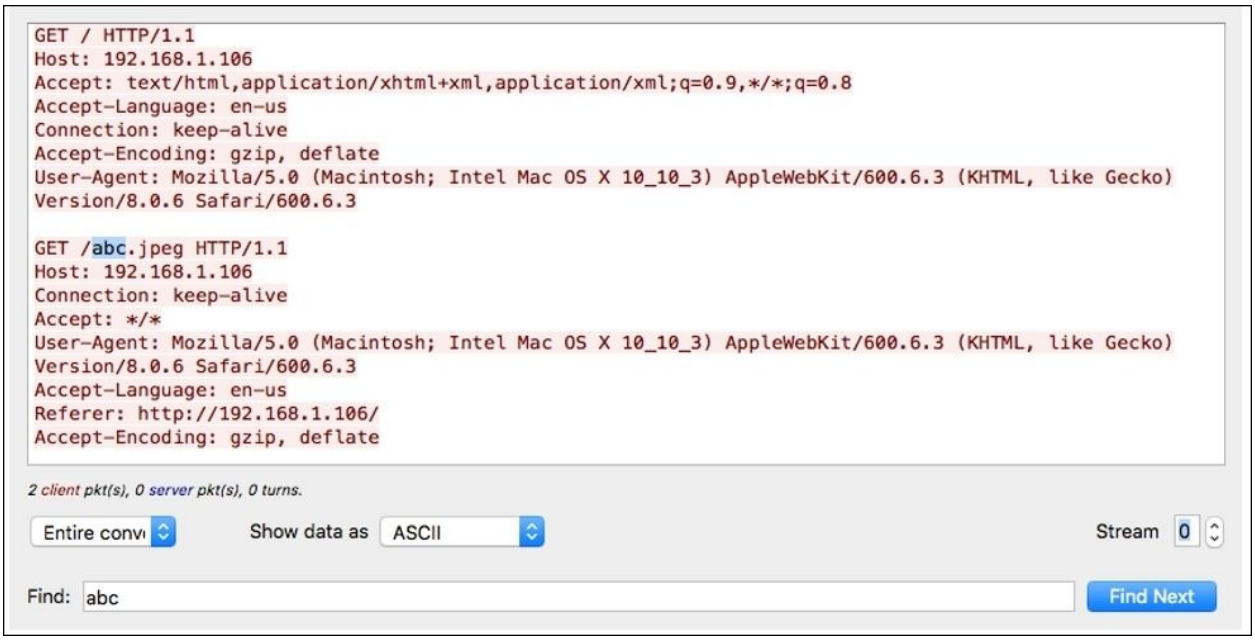


Figure 9.22: The Find utility in the Follow TCP stream dialog

USBPcap

USBPcap has been there from a long time with Linux and Mac users, but for Windows, this is the first time that users will be able to sniff the activity over USB interfaces. So, let's quickly walk through this latest feature and try to understand how to work with it with the help of an example. Follow the given steps to replicate the scenario:

1. After the successful installation of Wireshark on your Windows machine, it is highly recommended that you restart your machine because USBPcap might give you some trouble.
2. After your PC has restarted, open Command Prompt and change your current directory to the USBpcap installation directory that should be located at `C:\Program Files\USBPcap\`.
3. Now, perform a directory listing using the `dir` command to check whether `USBPcapCMD.exe` is present in the directory. Refer to the following screenshot that represents this step:

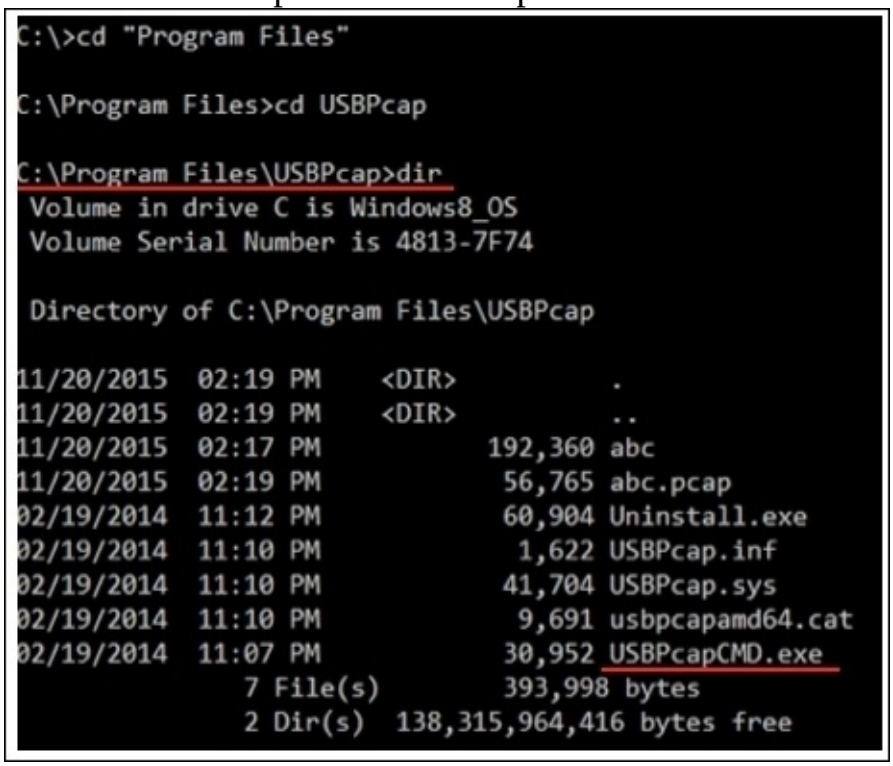
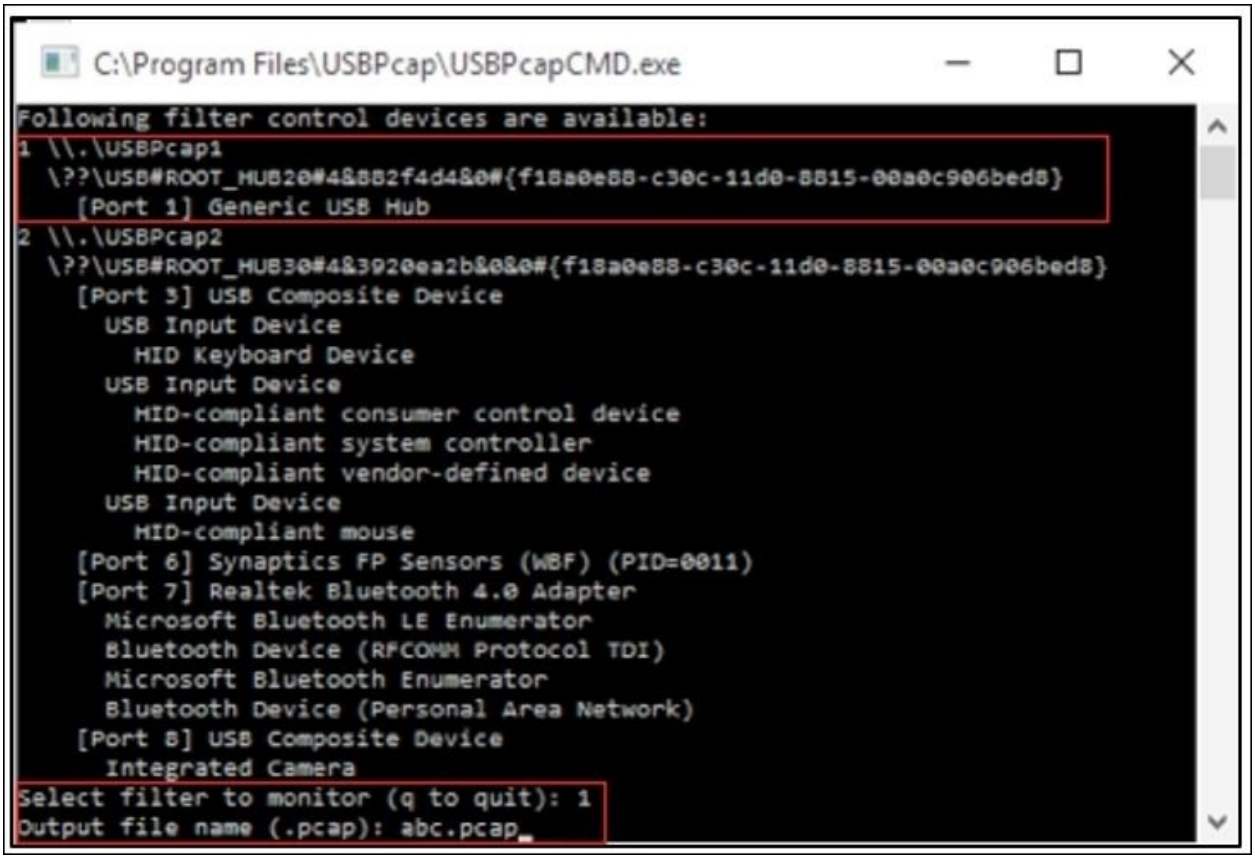


Figure 9.23: The USBPcap installation directory

- Type `USBPcapCMD.exe` in the Command Prompt to launch the sniffing application.
- As soon as it has been launched successfully, you will be asked to choose a root hub over which you want to sniff the traffic and the name of the trace file where you want to redirect the output. Refer to following screenshot that illustrates this:



- Now, as instructed, the application will initiate the sniffing process over root hub 1 and will dump any activity captured over the USB interfaces to the `abc.pcap` file.
- Now, try to copy something from your PC to the USB drive or vice versa. You probably won't be able to see any live activity over the Command Prompt, but in the background, it is actually running.
- Whenever you want to stop the sniffing process, you can press *Ctrl + C*.
- Now, it's time to open the abc.`pcap file` using Wireshark to see what we have in the trace file. Refer to the following screenshot that illustrates this:

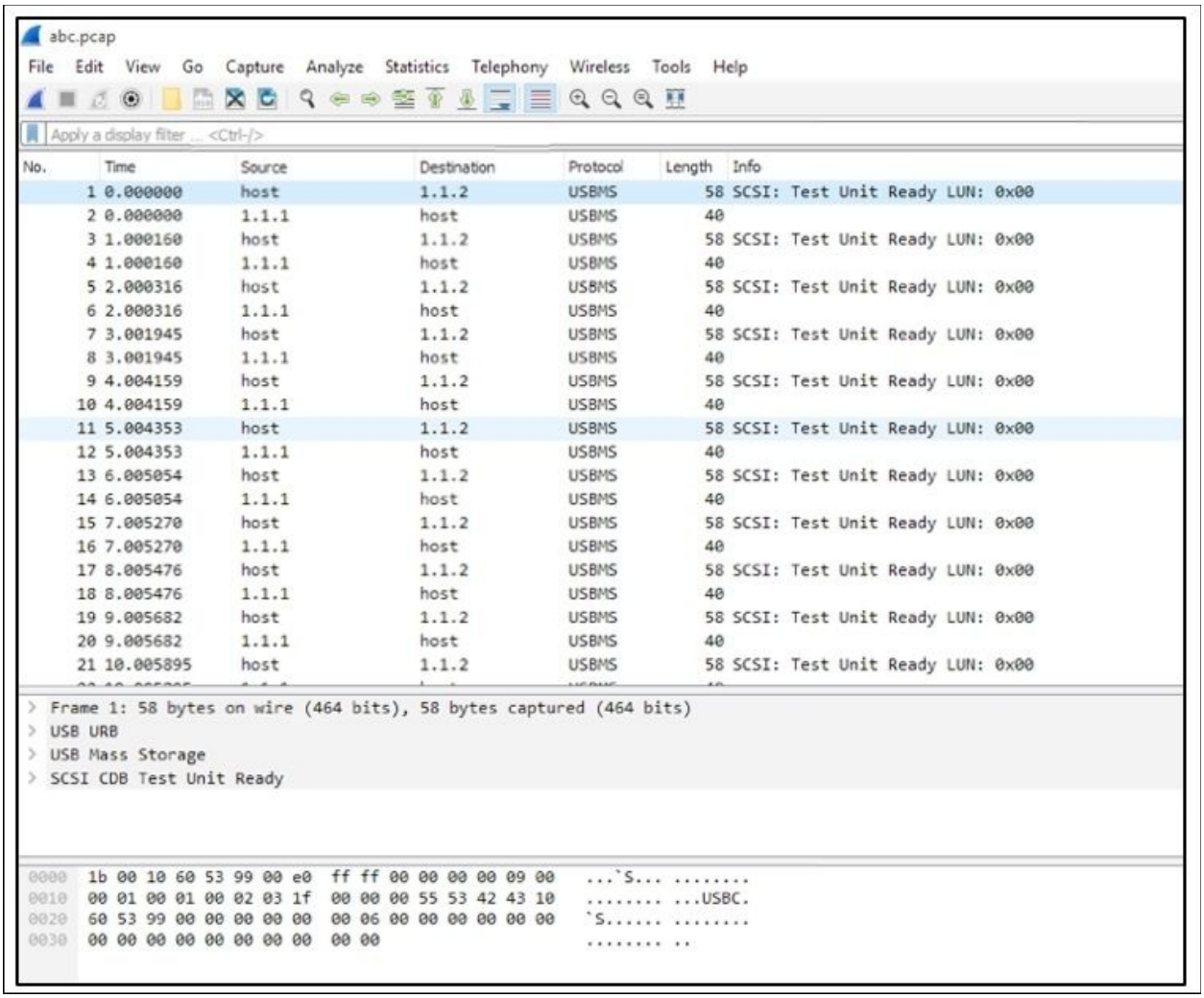


Figure 9.24: The abc.pcap trace file

As you can see, we have an activity, which got captured; it all looks similar to what we saw with network packets. We have all the familiar columns that list out various details such as time, source, destination, and so on. So we were able to successfully dump the activity over available USB interfaces without any technical hassle and I hope you will do some research to get a better understanding about USBPcap.

Summary

The newer version of Wireshark has adopted a new framework that gives us a new and totally amazing GUI. The older version was built upon the GTK framework, and since now we have the QT framework, from the perspective of a normal user, the differences are mostly concerned with its look and feel.

Scrolling is definitely one of the tools that we all have seen in all major applications, but hats off to the developers who came up with such a creative idea of showing the coloring pattern of your trace file inside the scroll bar while you are trying to look for something specific. It does give an extra advantage.

The Translation feature makes Wireshark more international and close to every user in terms of personalization. As many Wireshark users might not comfortable with the English language, now they have the facility to change the language to their native language, which would make the analytical process for a professional more effective.

Graphs are one of the features using which differences between normal and abnormal conditions can be figured out, and are used very often. Now, creating and customizing graphs is easier than ever, and the look and feel has drastically improved as well.

The following protocol-specific streams dialog is introduced with some of the new features that let you find an ASCII string, and itlets you move easily between the streams available too; you don't have to close and reopen the dialog to move to a different stream.

USBPcap has been there with us for quite a long time, and most Linux and Mac users are probably aware of this fact. The way your NIC card lets you listen over the wired/wireless channel is similar to the way the USBpcap option would let you listen over the USB ports that you have. This means that now, Wireshark can also trace the activities happening over a USB interface.

Practice questions

Q.1 Try to find out the major differences between the GTK and QT frameworks. And which one do you think is better?

Q.2 Try out the Translation feature by changing the system default language in Wireshark to any other language of your choice.

Q.3 Create a Flow graph using the newer version and the legacy version, and observe how many differences you can figure out between the graphs.

Q.4 Open any previous capture file you have, and try to figure out how many TCP streams there are in it.

Q.5 Figure out a way to remove the display filter button for the ARP protocol that we created earlier in this chapter.

Q.6 Try changing coloring rules for ARP packets, and check whether you can observe the difference in the intelligent scroll bar area.

Q.7 After installing the newer version of Wireshark on a Windows machine, try to launch USBPcap. Then, copy and paste from your PC to the sub device or vice versa (dump all the activities in the `test.pcap` file).

Q.8 Open the recently captured `test.pcap` trace file for the USB interface activity in Wireshark, and try to figure out what the packets listed in the list pane state. Specifically, try to analyze the values shown in the source and destination columns.

Bibliography

This course is a blend of different projects and texts all packaged up keeping your journey in mind. It includes the content from the following Packt products:

- *Wireshark Essentials - James H Baxter*
- *Network Analysis using Wireshark Cookbook - Yoram Orzach*
- *Mastering Wireshark - Charit Mishra*

Index

A

B

C

D

E

F

G

H

- features / Hypertext Transfer Protocol
- header / Hypertext Transfer Protocol
- Host field / Host
- Request Modifiers / Request Modifiers

I

J

L

M

N

- {n,}, modifier / How it works...
- {n}, modifier / How it works...

O

- about / Analyzing e-mail traffic and troubleshooting e-mail problems – POP, IMAP, and SMTP

P

    - track number of bytes in flight / Wireshark preferences
    - calculate conversation timestamps / Wireshark preferences
- protocols on Wikipedia
    - about / Protocols on Wikipedia
    - URL / Protocols on Wikipedia
- protocol_version / How to do it...
- provisional/informational codes / 1xx codes – provisional/informational
- proxy server / How it works...
- PSH / How it works…
- PSML / Saving data in various formats
- PSML (*.psml) / Saving data in various formats
- PSTN
    - about / How it works...
- PUT / HTTP methods

Q

- QOS data packet / WEP-open key
- qualifiers
  - type / How to use capture filters
  - direction / How to use capture filters
  - proto / How to use capture filters

R

to do it..., What are TCP retransmissions and what do they cause
    - to multiple destinations / Case 1 – retransmissions to many destinations
    - on single connection / Case 2 – retransmissions on a single connection
    - patterns / Case 3 – retransmission patterns
    - due to non-responsive application / Case 4 – retransmission due to a non-responsive application
    - due to delayed variations / Case 5 – retransmission due to delayed variations
- Retransmission Time Out (RTO) / How to do it..., How it works...
- RFC 2246, errors
    - close_modify / How to do it...
    - unexpected_message / How to do it...
    - bad_record_mac / How to do it...
    - decryption_failed / How to do it...
    - record_overflow / How to do it...
    - decompression_failure / How to do it...
    - handshake_failure / How to do it...
    - bad_certificate / How to do it...
    - unsupported_certificate / How to do it...
    - certificate_revoked / How to do it...
    - certificate_expired / How to do it...
    - certificate_unknown / How to do it...
    - illegal_parameter / How to do it...
    - unknown_ca / How to do it...
    - access_denied / How to do it...
    - decrypt_error / How to do it...
    - export_restriction / How to do it...
    - protocol_version / How to do it...
    - insufficient_security / How to do it...
    - internal_error / How to do it...
    - user_canceled / How to do it...
    - no_renegotiation / How to do it...
- Riverbed AirPcap adapter
    - about / AirPcap adapters
    - reference link / AirPcap adapters
- Riverbed Cascade

S

T

U

V

W

X

Y

Z