

C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Learning OMNeT++

Make realistic and insightful network simulations with OMNeT++

Thomas Chamberlain

[PACKT]
PUBLISHING

www.allitebooks.com

Learning OMNeT++

Make realistic and insightful network simulations with
OMNeT++

Thomas Chamberlain

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

Learning OMNeT++

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: September 2013

Production Reference: 1110913

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84969-714-9

www.packtpub.com

Credits

Author

Thomas Chamberlain

Project Coordinator

Sherin Padayatty

Reviewers

Thomas M. Chen

P. Victor Paul

Proofreader

Faye Coulman

Acquisition Editor

Vinay Argekar

Indexer

Hemangini Bari

Commissioning Editor

Shreerang Deshpande

Production Coordinator

Shantanu Zagade

Technical Editors

Dennis John

Gaurav Thingalaya

Cover Work

Shantanu Zagade

Copy Editors

Laxmi Subramanian

Adithi Shetty

Kirti Pai

Cover Image

Conidon Miranda

About the Author

Thomas Chamberlain grew up in the 90s in London. From a young age, he was passionate about programming and computers, which led him to pursue a degree in Computer Science, where he focused on intelligent, self-adaptive networks.

Since then, Tom has worked for a leading aerospace, defense, and advanced technology company as a Software and Systems Engineer.

I would like to thank my friends and family, who have always supported me, and I would like to dedicate this book to my wife, Andrea.

About the Reviewers

Thomas M. Chen received his BS and MS degrees in Electrical Engineering from the Massachusetts Institute of Technology, and a PhD in Electrical Engineering from the University of California, Berkeley. After graduation, he worked on high-speed networks research at GTE Laboratories (now Verizon) in Waltham, Massachusetts. He contributed to the national ATM standards and the Industry consortium ATM Forum. In 1997, he joined the Department of Electrical Engineering at Southern Methodist University, Dallas, Texas, as an Associate Professor. At SMU, he led the research in network security, network protocols, and traffic control. He joined Swansea University, Wales, UK, in 2008 as a professor of networking.

He is the co-author of *ATM Switching Systems*, Artech House (1995) and the co-editor of *Broadband Mobile Multimedia: Techniques and Applications*, Auerbach Publications (2008) and *Mathematical Foundations for Signal Processing, Communications, and Networking*, CRC Press (2012). He was formerly the Editor-in-chief of IEEE Communications Surveys, IEEE Communications Magazine, and IEEE Network. He was also formerly the Technical Editor for IEEE Press books and the Associate Editor for ACM Transactions on Internet Technology. He has served as a member-at-large in the IEEE Communications Society Board of Governors and as a Treasurer for the IEEE Computer Society's Security and Privacy group. He currently serves as the Associate Editor for the International Journal of Security and Networks, Journal on Security and Communication Networks, and International Journal of Digital Crime and Forensics. He received the IEEE Communications Society's Fred Ellersick Best Paper award in 1996.

P. Victor Paul is a research scholar pursuing a PhD in the field of Computer Science, Pondicherry University, Pondicherry, India. He has completed his B.Tech. in Information Technology (2007) from SMVEC with a university rank and M.Tech. in Network and Internet Engineering (2011) from Pondicherry University, Pondicherry, India, with a University gold medal. He is a recipient of the INSPIRE fellowship from the Department of Science and Technology, Govt. of India. He has around six years of professional experience in Industry and Research. Currently, he is working in the fields of Wireless Communications, Evolutionary Computing, and Distributed Systems. He is one of the researchers proficient in using and optimizing the OMNeT++ simulation tool and has developed a number of projects in the same. He has published around 20 research papers in various forums such as conferences and journals at national and international levels.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: Getting Started with OMNeT++	7
What this book will cover	7
What is OMNeT++?	8
The need for simulation	8
Examples of simulation in the industry	8
What you will learn	9
Summary	10
Chapter 2: Installing OMNeT++	11
Downloading OMNeT++ source code	11
Windows	12
Linux	12
Prerequisites for compiling OMNeT++ from the source	13
Installing OMNeT++ on Windows	14
Installing OMNeT++ on Linux	15
Compiling and installing on Windows	16
Compiling and installing on Linux	16
Running OMNeT++ IDE for the first time	17
Running OMNeT++ on Linux	18
Running OMNeT++ on Windows	19
Downloading INET	19
Importing INET to prepare for the next chapter	20
Summary	21

Chapter 3: OMNeT++ Simulations	23
Beginner to expert, low to high level	23
Components that make up OMNet++ simulations	23
Network	23
Module	24
Compound Module	26
Channels	27
The NED language	27
OMNeT++ configuration file	31
Simulation frameworks	34
Google Earth demo	35
INET in detail	37
Example INET simulations	38
Summary	42
Chapter 4: Creating and Running a Simulation	43
The OMNeT++ IDE	43
Creating a new project	46
Creating an empty project	46
Importing a project	48
Creating an example project	48
Switching to another workspace	54
Hello World network simulation	55
What this simulation will do	55
Defining your network	56
Creating multiple scenarios	67
Controlling the flow of your simulation	69
Summary	71
Chapter 5: Learning from Your Simulations	73
Gathering useful information	73
Visualizing gathered data	78
Vector charts	78
Scalar charts	79
Analysis of the Tictoc example project	79
Generating capture packet data	83
Summary	84
Index	85

Preface

I've always enjoyed blogging about anything and everything; things I love and things I want to share with others. I first started getting really into blogging while I was in my first year of university. I would write posts about computer security and my general ideas and thoughts on things surrounding the topic. Whenever I was particularly passionate about a topic that I was either studying or researching, I would blog about it. That's really how this book came about; from being at the university and then blogging about it. The reason why I was using OMNeT++ in the first place was because of the degree I was reading in Computer Science and the need to simulate networks for research. In my final year of graduation, I proposed to do my own dissertation instead of choosing from the set list that my batch had been given. It was a dissertation that was completely drenched in my passion for networks and security and was titled *Optimizing Self-Aware Networks to Defend Against Distributed Denial of Service Attack*. There was no way I could get around the epically long title! The reason why I wanted to defend against DDoS attacks was because of how popular they were. This popularity not only led me to realize how simple such an attack is, but also how difficult they are to defend against. In my research, I realized that I needed to study two main crippling barriers in creating a DDoS attack: cost and time. The main problem was that I needed to create massive wide area networks in order to observe the mechanisms of a life-sized DDoS attack. DDoS attacks were very popular in 2011 and are very popular even today. I found that the only good answer to this problem was simulation. I set out to try and create a lifelike model of a DDoS attack using simulation and not just something "like" a DDoS attack. I wanted something that truly was a DDoS attack, but just on a "non-physical" network, mostly because I couldn't afford the real thing and I wouldn't have had the time to set it up in real life.

After spending the next five minutes on Google, I came across OMNeT++. It boasted of everything that I was looking for and needed. OMNeT++ made network simulation fun for me. It made it easy to edit my simulations and learn from the networks I had created. I'm a very visual learner, so the visual side of OMNeT++ made the simulations completely real for me. I loved what I had found from my five-minute Google search. I was really impressed. So, as with everything similar I come across, I blogged about it. Shortly thereafter, I received an e-mail from Reshma Raman, an Author Acquisition Executive from Packt Publishing. The conversation went something like this: "Would you like to be an author for a mini book on OMNeT++?" "Sure," I replied. It felt like an invitation to extend my love of blogging on a grander scale, and I also knew that I'd learn a lot more about OMNeT++ in the process. You can never say you know everything, so I never do.

What this book covers

Chapter 1, Getting Started with OMNeT++, talks about what this book will cover, what you will learn from the book, and also the importance and uses of simulation.

Chapter 2, Installing OMNeT++, walks you through the steps of how you can get and install OMNeT++. It also includes how to get optional add-ons and what they do.

Chapter 3, OMNeT++ Simulations, goes into detail about the components that make up OMNeT++ simulations and how a simulation can be configured into numerous scenarios.

Chapter 4, Creating and Running a Simulation, walks you through the steps of how you can create your own network simulation, which includes creating the network topology and creating multiple scenarios for your simulation. It then explains how to run and control the flow of your simulation.

Chapter 5, Learning from Your Simulations, shows you how to gather data from your network simulation and then how to create visualizations to represent the data collected.

What you need for this book

For this book, you need a Windows operating system, such as Windows XP or later versions of Ubuntu Linux 12.04 or 13.04. Other Linux distributions will work, but this book uses Ubuntu.

Prior knowledge on C++ programming will aid a reader of this book, but it is not essential.

Who this book is for

This book is ideal for anyone wishing to quickly get to grips on network simulation, whether you're an expert in networks or just a beginner. It is perfect for all network engineers and administrators who wish to emulate networks using OMNet++ as a preparation for building the actual network.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "You may need to change 4.2.2 to match the version you have downloaded. This also assumes that the tarball was downloaded into your Downloads folder."


A block of code is set as follows:


```
standardHost: StandardHost {
  @display("p=59,215");
  IPForward = true; //Turn IP forwarding on
  numTcpApps = 5; //Set the number of TCP Apps on this node to 5
}
```

Any command-line input or output is written as follows:

```
sudo apt-get update && sudo apt-get install build-essential
gcc g++ bison flex perl
tcl-dev tk-dev blt libxml2-dev
xlibg-dev openjdk-6-jre doxygen graphviz
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "The benefit of using the **Edit Parameters** form is that you can see all the parameters that belong to the submodule you are observing and their default values".

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via 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 on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, 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 all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

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 books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, 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 on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright 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

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with OMNeT++

This book is intended for a whole range of people, from network engineers who want to create reliable networks to budding simulation enthusiasts. I know I would have benefited from a book like this when I was in my final year of University. That's when I realized I needed to simulate networks to solve the problems I had. This book would have been useful for me, because once I realized I wanted to simulate a network, I also realized that I had no idea how to do it. Once I discovered OMNeT++, I also found the learning curve for using it to be steep to start with, and I really wanted the network simulations that I would soon create to be up and running as quickly as possible. I wish for this book to be useful, interesting and also fun.

What this book will cover

This book will show you how you can get OMNeT++ up and running on your Windows or Linux operating system. This book will then take you through the components that make up an OMNeT++ network simulation. The components include models written in the **NED (Network Description)** language, initialization files, C++ source files, arrays, queues, and then configuring and running a simulation. This book will show you how these components make up a simulation using different examples, which can all be found online. At the end of the book, I will be focusing on a method to debug your network simulation using a particular type of data visualization known as a sequence chart, and what the visualization means.

What is OMNeT++?

OMNeT++ stands for Objective Modular Network Testbed in C++. It's a component-based simulation library written in C++ designed to simulate communication networks. OMNeT++ is not a network simulator but a framework to allow you to create your own network simulations.

The need for simulation

Understanding the need for simulation is a big factor in deciding if this book is for you. Have a look at this table of real network versus simulated network comparison.

A real network	A network simulation
The cost of all the hardware, servers, switches and so on has to be borne.	The cost of a single standalone machine with OMNeT++ installed (which is free).
It takes a lot of time to set up big specialist networks used for business or academia.	It takes time to learn how to create simulations, though once you know how it's done, it's much easier to create new ones.
Making changes to a pre-existing network takes planning, and if a change is made in error, it may cause the network to fail.	Making changes to a simulated network of a real pre-existing network doesn't pose any risk. The outcome of the simulation can be analyzed to determine how the real network will be affected.
You get the real thing, so what you observe from the real network is actually happening.	If there is a bug in the simulation software, it could cause the simulation to act incorrectly.

As you can see, there are benefits of using both real networks and network simulations when creating and testing your network. The point I want to convey though, is that network simulations can make network design cheaper and less costly.

Examples of simulation in the industry

After looking into different industries, we can see that there is obviously a massive need for simulation where the aim is to solve real-world problems from how a ticketing system should work in a hospital to what to do when a natural disaster strikes. Simulation allows us to forecast potential problems without having to first live through those problems.

Different uses of simulation in the industry are as follows:

- **Manufacturing:** The following are the uses under manufacturing:
 - To show how labor management will work, such as worker efficiency, and how rotas and various other factors will affect production
 - To show what happens when a component fails on a production line
- **Crowd Management:** The following are the uses under crowd management:
 - To show the length of queues at theme parks and how that will affect business
 - To show how people will get themselves seated at an event in a stadium
- **Airports:** The following are the uses for airports:
 - Show the effects of flight delays on air-traffic control
 - Show how many bags can be processed at any one time on a baggage handling system, and what happens when it fails
- **Weather Forecasting:** The following are the uses under weather forecasting:
 - To predict forthcoming weather
 - To predict the effect of climate change on the weather

That's just to outline a few, but hopefully you can see how and where simulation is useful.

Simulating your network will allow you to test the network against myriads of network attacks, and test all the constraints of the network without damaging it in real life.

What you will learn

After reading this book you will know the following things:

- How to get a free copy of OMNeT++
- How to compile and install OMNeT++ on Windows and Linux
- What makes up an OMNeT++ network simulation
- How to create network topologies with NED

- How to create your own network simulations using the OMNeT++ IDE
- How to use pre-existing libraries in order to make robust and realistic network simulations without reinventing the wheel

Learning how to create and run network simulations is definitely a big goal of the book. Another goal of this book is to teach you how you can learn from the simulations you create. That's why this book will also show you how to set up your simulations, and to collect data of the events that occur during the runtime of the simulation. Once you have collected data from the simulation, you will learn how to debug your network by using the Data Visualization tools that come with OMNeT++. Then you will be able to grasp what you learned from debugging the simulated network and apply it to the actual network you would like to create.

Summary

You should now know that this book is intended for people who want to get network simulations up and running with OMNeT++ as soon as possible. You'll know by now, roughly, what OMNeT++ is, the need for simulation, and therefore OMNeT++. You'll also know what you can expect to learn from this book.

2

Installing OMNeT++

It's important that you have OMNeT++ correctly installed onto your Windows or Linux operating system. OMNeT++ has some prerequisites and optional add-ons that we will also be installing to ensure a fully working OMNeT++ environment. The Linux operating system I am using throughout this book is Ubuntu 12.10. The Windows version I will be using is Windows 7. It's also important that you read the licenses for each tool and package you install so that you are aware about what you are allowed to do with those tools and packages.

Downloading OMNeT++ source code

OMNeT++ is not available on the community website as an executable program; only its source code is available. This means that you need to compile the source code in order to run OMNeT++. If you are not a confident Linux user, I suggest that you use Windows as it's much easier to install on Windows.

First, go to the OMNeT++ download page at the following URL: <http://www.omnetpp.org/omnetpp>. Once there, you need to click on **OMNeT++ Releases**. Now you will see two download selections:

1. OMNeT++ x.x.x win32 (source + IDE + MINGW, zip)
2. OMNeT++ x.x.x (source + IDE, tgz)

If you are using Windows, click on option 1; you can click on option 2 if you are using Linux. Throughout this chapter, you will only need to read the content for the operating system you are using.

Windows

Once you have downloaded the ZIP file, you need to unzip it to a folder of your choice. To unzip it, you need to right click on the zipped file and then click on **Extract All**. I recommend that you can easily find it on the root folder of your local drive.

Linux

Once you have downloaded the tarball, you need to extract it either using the file manager you have installed (this is typically done by right-clicking on the tarball and then clicking on **Extract Here**, or double-clicking on the file and selecting where to extract it) or entering the following code into your terminal:

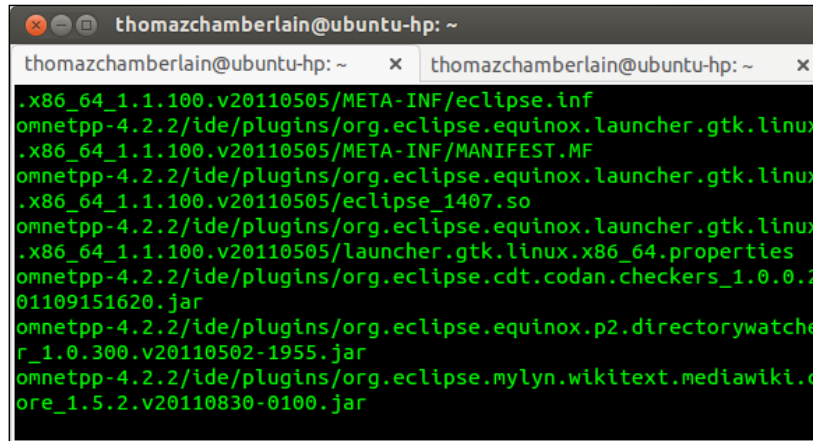
```
tar xvzf ~/Downloads/omnetpp-4.2.2-src.tgz -C ~/
```

You may need to change 4.2.2 to match the version you have downloaded. This also assumes that the tarball was downloaded into your Downloads folder. What this command does is extract the tarball into your home directory. The parameter `xvzf` used in the command stands for the following:

- `x`: Extract
- `v`: Verbose (so that you can see what is happening while the command is running)
- `z`: Uncompress
- `f`: File to extract followed by the tarball that you downloaded

`-C` is then used to declare the directory that we want to extract the files into. If you wish to change the directory that the tarball is extracted into, you need to change the command to match what you want. For more information about using the `tar` command, enter `man tar` into your terminal to bring up the manual for the `tar` command.

While the command is running, you should see something like the following in your terminal:

A terminal window titled 'thomazchamberlain@ubuntu-hp: ~' showing a list of files being extracted. The files are listed in green text on a black background. The paths include various Eclipse IDE plugins and launchers, such as 'eclipse.inf', 'eclipse.equinox.launcher.gtk.linux', 'MANIFEST.MF', 'eclipse_1407.so', 'launcher.gtk.linux.x86_64.properties', 'cdt.codan.checkers_1.0.0.201109151620.jar', 'p2.directorywatcher_1.0.300.v20110502-1955.jar', and 'mylyn.wikitext.mediawiki.core_1.5.2.v20110830-0100.jar'.

```
thomazchamberlain@ubuntu-hp: ~
thomazchamberlain@ubuntu-hp: ~ x thomazchamberlain@ubuntu-hp: ~ x
.x86_64_1.1.100.v20110505/META-INF/eclipse.inf
omnetpp-4.2.2/ide/plugins/org.eclipse.equinox.launcher.gtk.linux
.x86_64_1.1.100.v20110505/META-INF/MANIFEST.MF
omnetpp-4.2.2/ide/plugins/org.eclipse.equinox.launcher.gtk.linux
.x86_64_1.1.100.v20110505/eclipse_1407.so
omnetpp-4.2.2/ide/plugins/org.eclipse.equinox.launcher.gtk.linux
.x86_64_1.1.100.v20110505/launcher.gtk.linux.x86_64.properties
omnetpp-4.2.2/ide/plugins/org.eclipse.cdt.codan.checkers_1.0.0.2
01109151620.jar
omnetpp-4.2.2/ide/plugins/org.eclipse.equinox.p2.directorywatche
r_1.0.300.v20110502-1955.jar
omnetpp-4.2.2/ide/plugins/org.eclipse.mylyn.wikitext.mediawiki.c
ore_1.5.2.v20110830-0100.jar
```

You will know that the extraction is complete when the terminal stops outputting the files it's extracting.

You can double check if the extraction worked by locating the extracted folder which will be in your home directory or where you have otherwise specified.

Prerequisites for compiling OMNeT++ from the source

There are optional prerequisites that you can build OMNeT++ with, but on Linux you also need to make sure you have the correct mandatory prerequisites in order for OMNeT++ to build.

There are three optional packages which can be used to build OMNeT++. They are as follows:

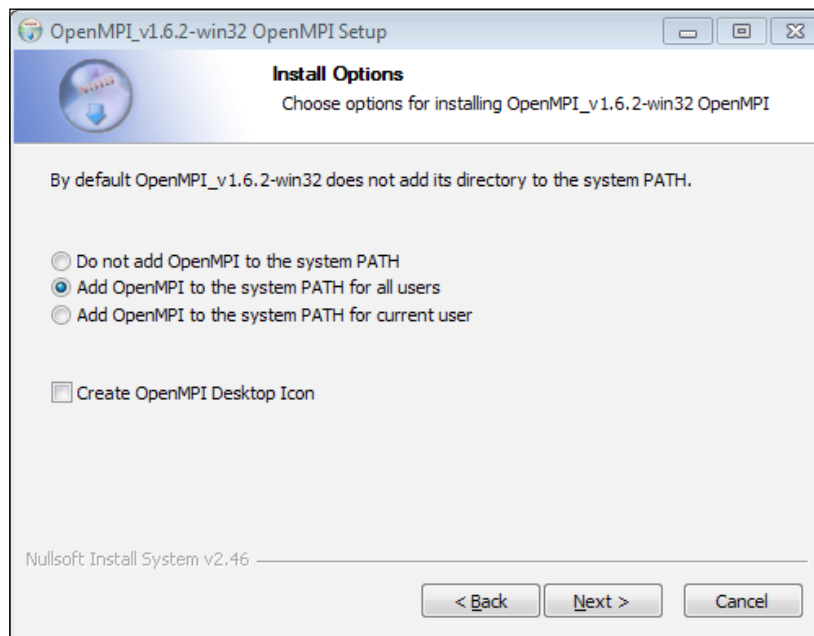
- **MPI:** This stands for message passing interface and is a message-passing library. This means that it is used for parallel programming that can be done across multiple machines with distributed memory. You may find this useful for OMNeT++ as it will allow you to speed up your simulations. Essentially, MPI will facilitate faster and more powerful simulations.
- **PCAP:** This stands for packet capture, and it allows packets that are travelling through the local network to be captured.

- **Akaroa:** Akaroa smartly collects data during the runtime of a simulation and stops the simulation once enough data has been collected to make a meaningful observation from that data. This tool is very useful for anyone who is trying to debug a network to make it more efficient and robust.

Installing OMNeT++ on Windows

All the prerequisites for installing OMNeT++ on Windows are actually included in the ZIP file that we have downloaded. This is why installing OMNeT++ on Windows is a lot easier than installing it on Linux. However, the optional packages are not included.

To get MPI for Windows, you first need Visual Studio Express for Windows desktop which can be found at <http://www.microsoft.com/visualstudio/eng/downloads>. Once you have installed Visual Studio, you just need to run the MPI installer provided by Open MPI that can be found at <http://www.open-mpi.org/software/mpi/v1.6/>. During the installation, make sure that the option to add MPI to the path is selected in the following manner:



You need to edit the `configure.user` file found in your OMNeT++ folder in order to tell OMNeT++ where to find your MPI. If you read the `configure.user` file, you will find the instructions and examples on how to do this.

The PCAP implementation for Windows is called **WinPcap** and can be downloaded from <http://www.winpcap.org/install/default.htm>. When installing WinPcap, I recommend that you select the option to automatically start the WinPcap driver at boot time.

Unfortunately, Akaroa is not Windows-friendly and this book will not cover how to install it on the Windows operating system.

Installing OMNeT++ on Linux

I will show you how to install the MPI implementation for Linux, Open MPI. It provides a full implementation of the message-passing interface. To install, open your terminal and enter:

```
sudo apt-get install openmpi-bin libopenmpi-dev
```

You can also install Open MPI by using the synaptic package manager that is found on most of the popular Linux distributions.

The PCAP implementation for Linux is **libpcap** and can be installed from the terminal by entering the following command:

```
sudo apt-get install libpcap-dev
```

You can also install libpcap-dev by using the synaptic package manager.

Akaroa can be downloaded from http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.shtml. You must first register to download the source code for Akaroa, as you need to compile it. Once you have downloaded the Akaroa tarball, you need to first extract it and then, using your terminal, move it into the extracted directory and enter the following command:

```
./configure  
make  
sudo make install
```


If there are any errors in this installation, please consult the documentation that is provided in the Akaroa folder which you just downloaded.

You must have some packages before you can compile OMNeT++; these can be downloaded and installed by entering the following command into your terminal:

```
sudo apt-get update && sudo apt-get install build-essential  
gcc g++ bison flex perl  
tcl-dev tk-dev blt libxml2-dev  
xlib1g-dev openjdk-6-jre doxygen graphviz
```

Once you have installed these packages, you can learn more about them if you are curious by using the `man` command. Not every package will have a main page that you can view, it just depends whether one has been written or not. For example, to find out more about `flex`, you would enter the following command into your terminal:

```
man flex
```



Downloading the example code

You can download the example code files for all Packt books that you have purchased from your account at <http://www.packtpub.com>. If you purchase this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Compiling and installing on Windows

The process of compiling and installing on Windows is very self-contained. To compile and install, just follow the given steps:

1. Enter the OMNeT++ directory that you unzipped earlier.
2. Run the file called `Mingwenv.cmd`.
3. When the terminal appears, enter the following command:

```
./configure  
make
```

Compiling and installing on Linux

To compile OMNeT++ on Linux, you must open up a terminal and move it to the extracted OMNeT++ folder that you downloaded earlier. Now enter the following command into your terminal:

```
./configure  
make
```

On Ubuntu 12.10, I received the following error message when I tried to run the `make` command:

```
abspath.cc: In function 'std::string toAbsolutePath(const char*):  
abspath.cc:63:38: error: 'getcwd' was not declared in this scope
```

Thanks to the support of the OMNeT++ community online, I realized that the fix to the problem was editing `abspath.cc` that is found in `src/utils/` of your extracted the OMNeT++ folder. Using the text editor of your choice, open `abspath.cc` and then add the following code onto line 21:

```
#include <unistd.h>
```

Save and close `abspath.cc` and rerun the `make` command. Once the last command has finished running, OMNeT++ will be installed. Making sure that you're still in the same directory in your terminal, you can now create a menu item from your terminal by typing the following command:

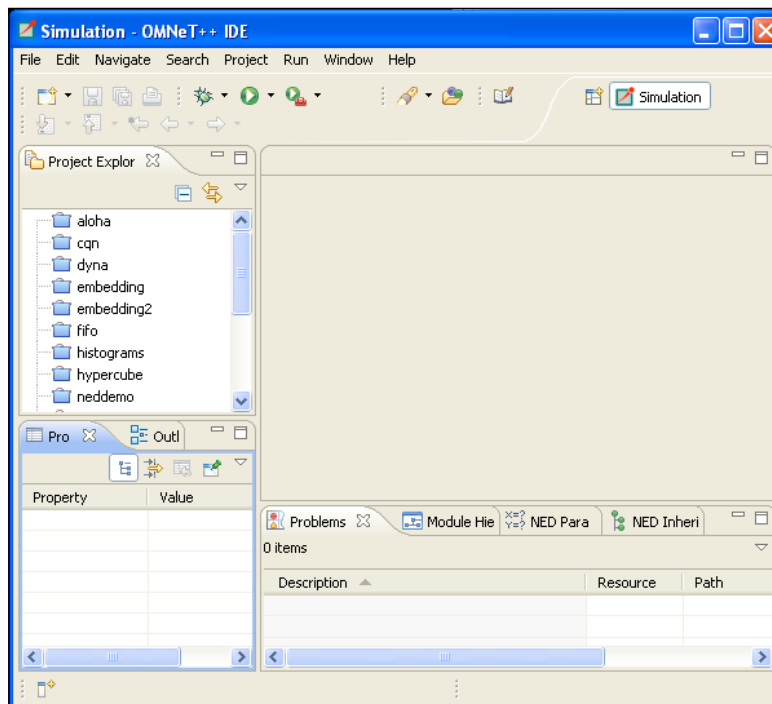
```
make install-menu-item
```

If you want to also create a desktop icon for OMNeT++, enter the following command into your terminal:

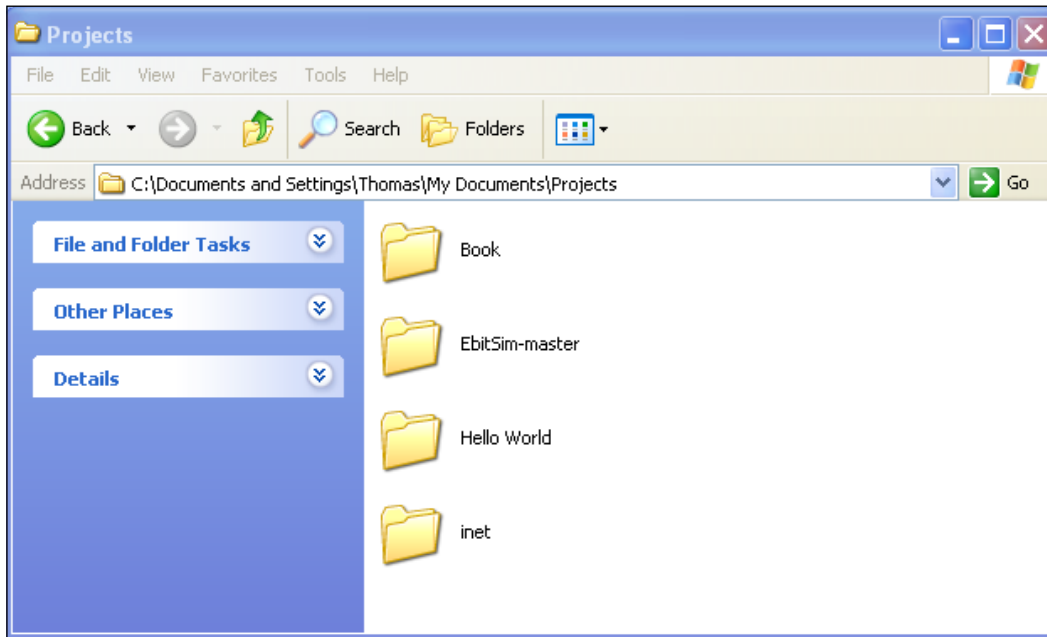
```
make install-desktop-icon
```

Running OMNeT++ IDE for the first time

Now that OMNeT++ is installed, it is as easy to run as any other piece of software on your computer. OMNeT++ can now be run using either your mouse or terminal.



Once you run OMNeT++, you will be asked to select the default workspace. A workspace is a logical collection of projects. For example, a workspace called peer to peer networks may contain only peer-to-peer network simulations. On my installation, I've created my own workspace folder which I've called Projects. My workspace looks like the following screenshot:



Running OMNeT++ on Linux

To run OMNeT++ on Linux, you can type the following command into your terminal:

```
omnetpp
```

If you have created a menu item, you can run OMNeT++ by clicking on **OMNeT++** from your application menu. The icon on my Ubuntu 12.10 looks like the following:



Running OMNeT++ on Windows

In the folder `ide` inside your `OMNeT++` folder, you will see an application called **omnetpp**; this is just a short form for OMNeT++. I recommend creating a shortcut for `omnetpp` on your desktop.

You can also run `omnetpp` by running `Mingwenv.cmd` as you did earlier and enter the command:

```
omnetpp
```

If you see the following splash screen, it means the OMNeT++ IDE is running correctly:



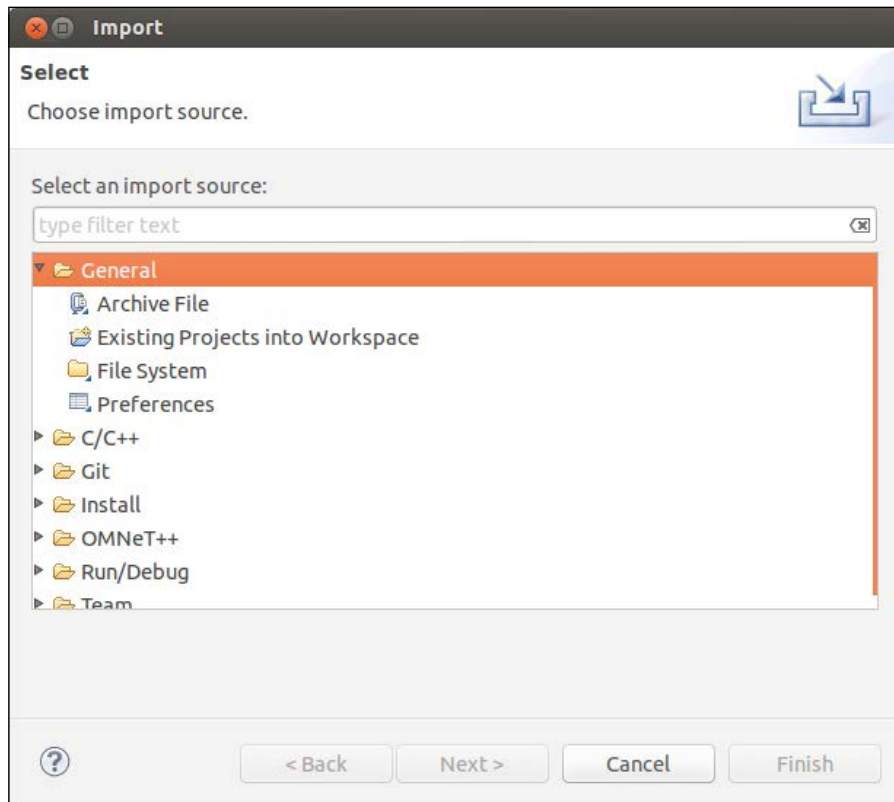
Downloading INET

INET can be downloaded from the download section at <http://inet.omnetpp.org>. The downloaded file is a tarball. Once it is downloaded, you can open it and choose where to extract to remember how this was done earlier. Also, choose a place to extract so that you remember.

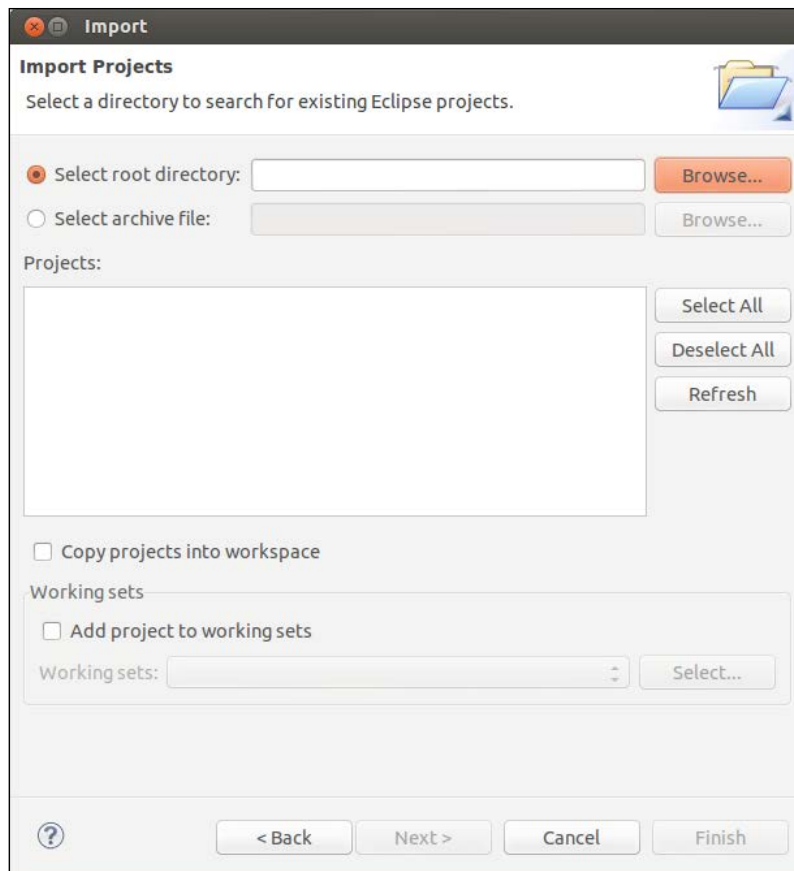
Importing INET to prepare for the next chapter

The following are the steps to prepare for the next chapter:

1. In OMNeT++, go to **Select File | Import**.



2. Click on **General** and then click on **Existing Projects into Workspace**. Now, click on **Browse** and select the **INET** folder that you just extracted. Make sure you tick the **Copy projects into workspace** checkbox.



3. Now click on **Finish**.
4. Now with **INET** selected under **Project Explorer**, press *Ctrl + B*. This will build INET so that it can be used in your network simulations.

Summary

Now you know how to get a free copy of OMNeT++, build the source code, compile it, and also build it with optional add-ons. You have also learned how to launch OMNeT++, obtain INET to import into OMNeT++, and then build it so that it's ready for you to use.

3

OMNeT++ Simulations

Let's now look at what exactly is an OMNeT++ network simulation and what it looks like. This chapter will give you a quick overview of an OMNeT++ simulation with the help of an example to aid your learning. OMNeT++ is open source, meaning that all of the source code that makes up OMNeT++ has been made freely available to the public, thereby allowing people to modify that code and then distribute it to others.

Beginner to expert, low to high level

If you are not a coder and you really don't need to learn how to code, well then, you don't need to! OMNeT++ not only lets you get right down to the C++ code of the network simulations, but it also lets you use higher building blocks. In this way, it caters to all types of users.

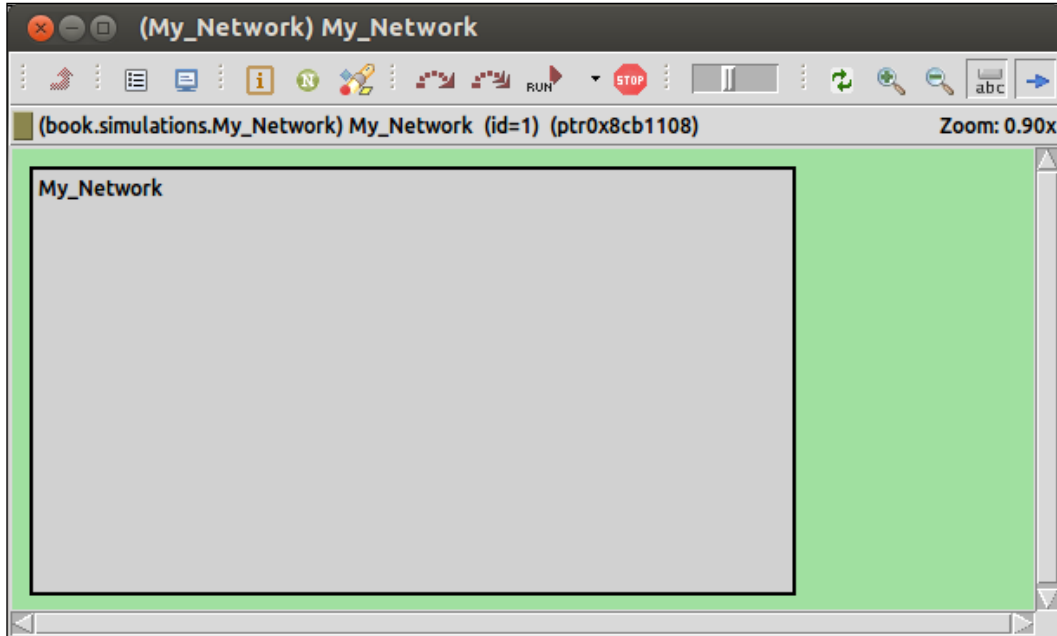
Components that make up OMNet++ simulations

Let us understand each component that makes up the simulation of OMNeT++.

Network

In OMNeT++, a **Network** is an object that defines the network and holds modules, submodules, and compound modules. The Network object allows those different types of modules to talk to each other via channels.

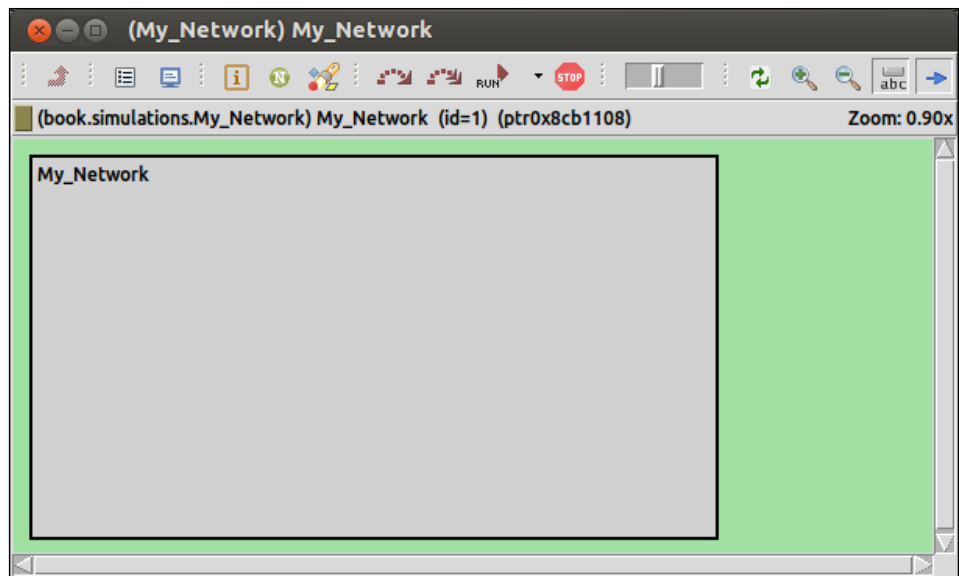
When you run your network simulation, the Network object will look like the following:



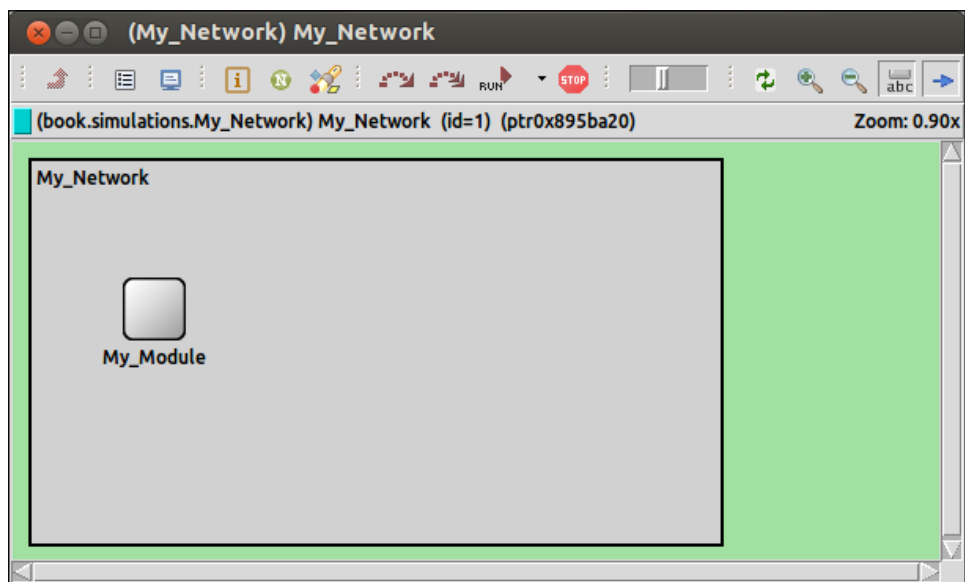
I have called this Network object **My_Network**.

Module

A **Module** object in OMNeT++ is a component that sits inside a Network object and is able to send messages to other Module objects. In your simulation, this could be a router, a webserver, a standalone machine, or any other component that you can think of, which is capable of making the communication across a network. When we add a Module object to our **My_Network** Network object, it looks like the following:

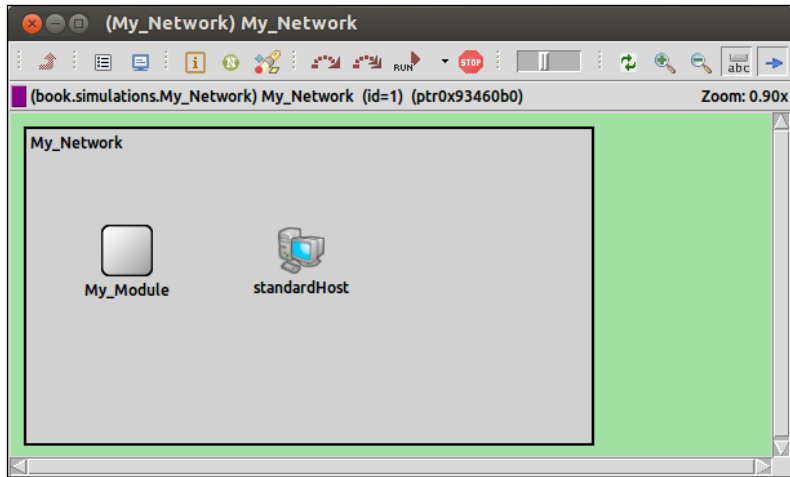


I have named the module **My_Module** and you can see that it sits inside the **My_Network** Network object, as shown in the following screenshot:

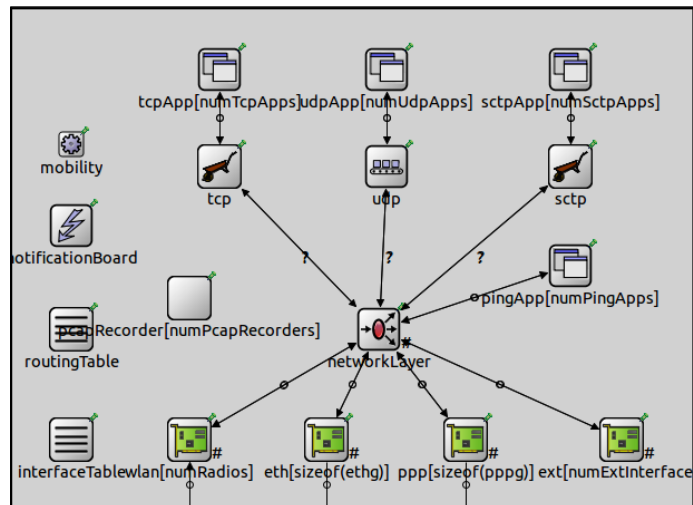


Compound Module

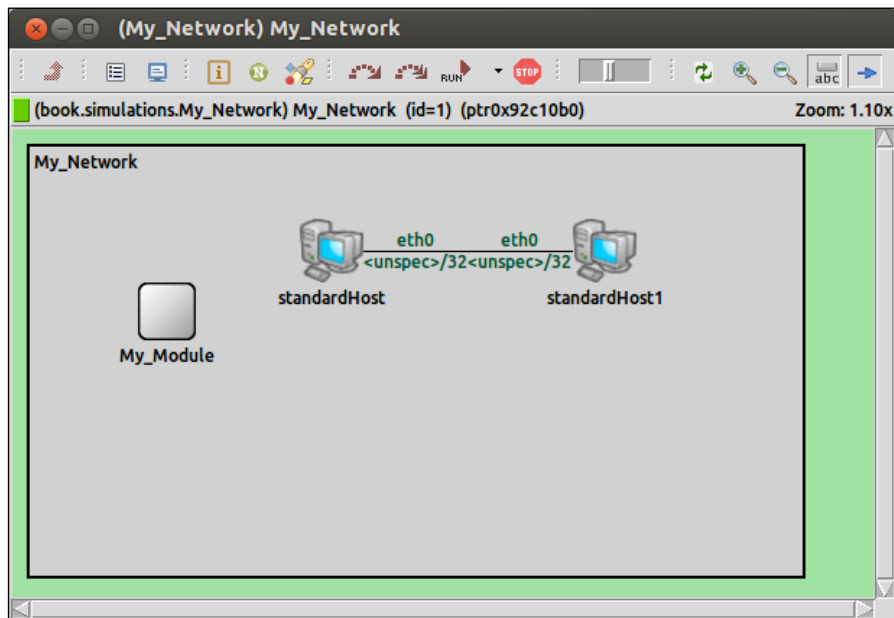
A **Compound Module** object is made up of multiple Module objects. The INET simulation framework for OMNeT++ is a library that contains loads of prebuilt Module and Compound Module objects. The following is what a StandardHost Compound Module looks like inside our **My_Network** Network object:



INET provides a neat icon for the StandardHost Compound Module object. If we have a look at what makes up this object, we can see a collection of Module and Compound Module objects that together make up a typical or "standard" host that you would find on a real-life network. The following is what the StandardHost Compound Module object looks like under the trunk:



Note the modules such as TCP and UDP; these prebuilt components are all ready for use and at your disposal! We can also see that the StandardHost Compound Module has a module inside it titled **pingApp**. This is an instantiation of the iPingApp Module Interface and it allows StandardHost to be used as a ping application and thereby send pings to other nodes on the network as shown in the following screenshot:



Channels

Channels are objects that are used to connect one module of the Compound Module object to the other. Channels allow these modules to send messages to each other and connect the gate of one module to the gate of another. This could be a StandardHost talking to another StandardHost via an Ethernet cable.

The NED language

NED is a network description language that is used to create the topology of networks in OMNeT++. The OMNeT++ IDE lets you create your topology either graphically or using NED. If you choose to create your network topologies using the graphical editor, the coinciding NED source code will also be generated for you. The choice to use the graphical editor over the NED language is purely yours.

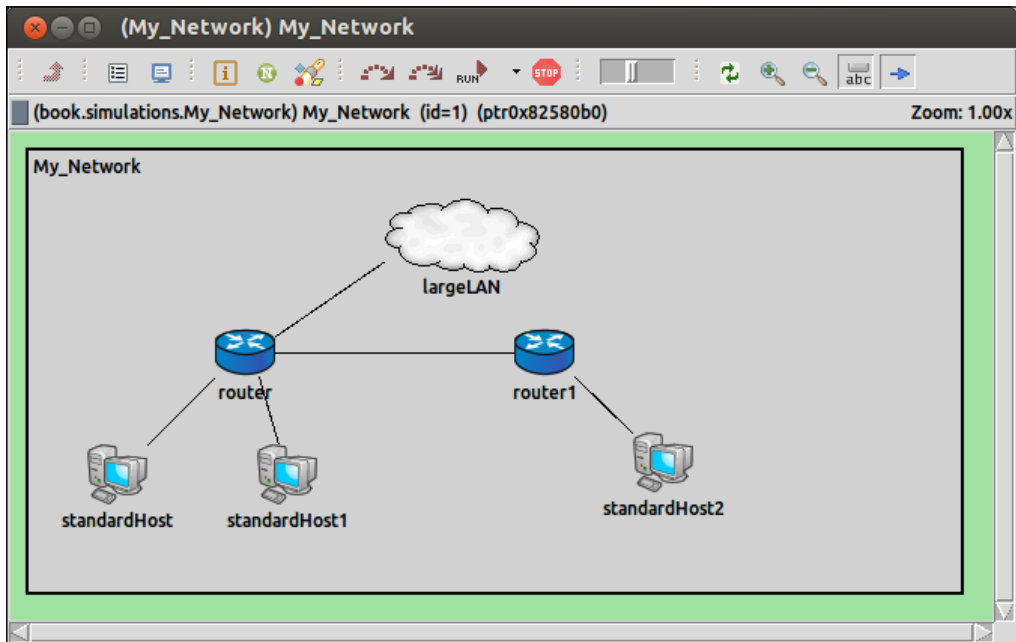
Let's look at some basic NED to get started. We will start by looking at how to define the most basic component – the Network component:

```
package book.simulations; /* Package is a mechanism to organize
                           the various classes and files. My
                           simulation project inside of OMNeT++ is
                           called "Book" and this NED file is found
                           in the "simulations" folder of the
                           project. */

networkMy_Network // This is simply defining a Network component
                  called
{
    "My_Network"
}
```

The text after the // and in between /* */ are the comments that explain what the syntax means. Graphically, this NED produces just an empty network called My_Network.

Let's consider this network topology. We have three standard hosts and a large **Local Area Network (LAN)** connected via two routers, as shown in the following screenshot:



Let's now look at the corresponding NED code that belongs to this topology:

```
package book.simulations;

/*
Similar to JAVA syntax, the NED file needs to include modules that are
to be used to define the network topology. Since we are using INET,
many of the imports are from the INET library.
*/
import inet.applications.httpptools.HttpServer;
import inet.examples.ethernet.lans.LargeLAN;
import inet.examples.httpptools.socket.tenserverssocket
        .ethernetline;

import inet.examples.mobileipv6.fiberline;
import inet.nodes.ethernet.Eth100G;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
import ned.DatarateChannel;

network My_Network
{
@display("bgb=620,293");

/* @display is just used to describe the appearance of the topology,
such as locations of components and other appearance attributes. */

submodules: //Define submodules that belong in our Network object.
standardHost: StandardHost {
@display("p=59,215");
}
        standardHost1: StandardHost {
@display("p=172,215");
}
router: Router {
@display("p=144,134");
}
largeLAN: LargeLAN {
@display("p=288,56;is=1");
}
        router1: Router {
@display("p=343,134");
}
}
```



```
        standardHost2: StandardHost {
@display("p=422,207");
        }
connections:

/* Define what connections exist in our Network object. In the first
line below, we can see that "standardHost" uses an Ethernet connection
to connect to "router" using the "ethernetline" channel.*/

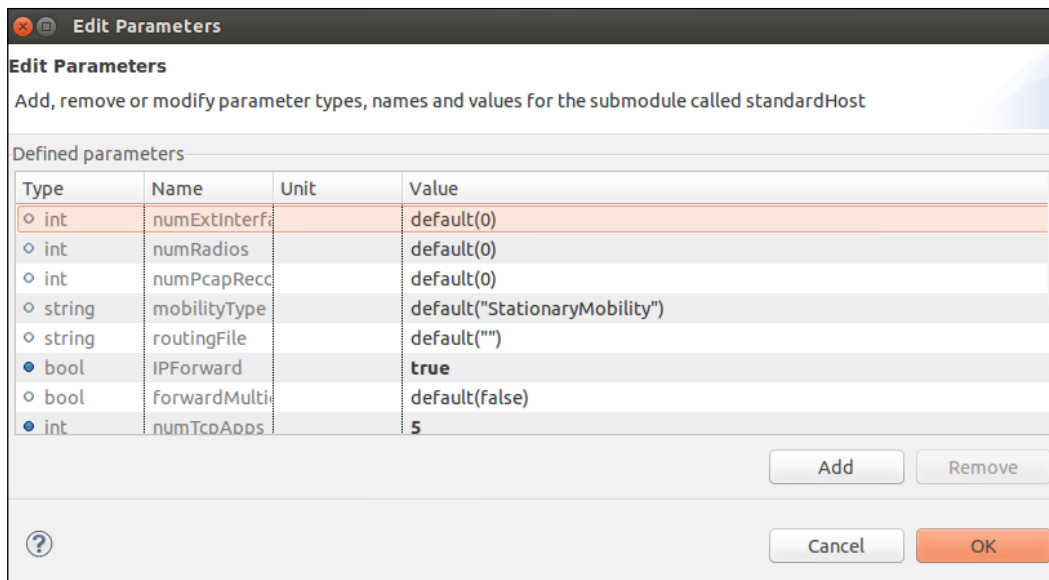
standardHost.ethg++ <-->ethernetline<-->router.ethg++;
router.pppg++ <-->DatarateChannel<--> standardHost1.pppg++;
router.ethg++ <-->fiberline<-->largeLAN.ethg;
router.ethg++ <-->fiberline<--> router1.ethg++;
router1.ethg++ <--> Eth100G <--> standardHost2.ethg++;
}
```

When getting to grips with OMNeT++, I found it very useful to create my network topologies using the graphical editor found inside the IDE. Once I added a new object to my topology, I would check the coinciding NED source code to see what changes had been made. I would recommend doing this too, because it is easier to see exactly what attributes are set for your submodules in the source code instead of having to navigate hidden menus in order to see the attributes.

Let's add some more details to the first submodule `standardHost :StandardHost` that was defined previously:

```
standardHost: StandardHost {
@display("p=59,215");
IPForward = true; //Turn IP forwarding on
numTcpApps = 5; //Set the number of TCP Apps on this node to 5
}
```

These added details are very easy to see in the NED source code. However, to set and view these attributes in the IDE, you must right-click on the **standardHost** object and select **Parameters...** Now you will see a form similar to the one shown in the following screenshot:



This is a much longer way to change settings, but I must admit that it feels easier. The benefit with NED is that you can see the parameters for submodules much more easily. The benefit of using the **Edit Parameters** form is that you can see all the parameters that belong to the submodule you are observing and their default values.

OMNeT++ configuration file

The configuration file is what defines how the simulation will run. In every simulation, this configuration is called `omnetpp.ini`. Without this file, the simulation will not run.

To demonstrate a configuration file, let's consider the following network described by this topology:

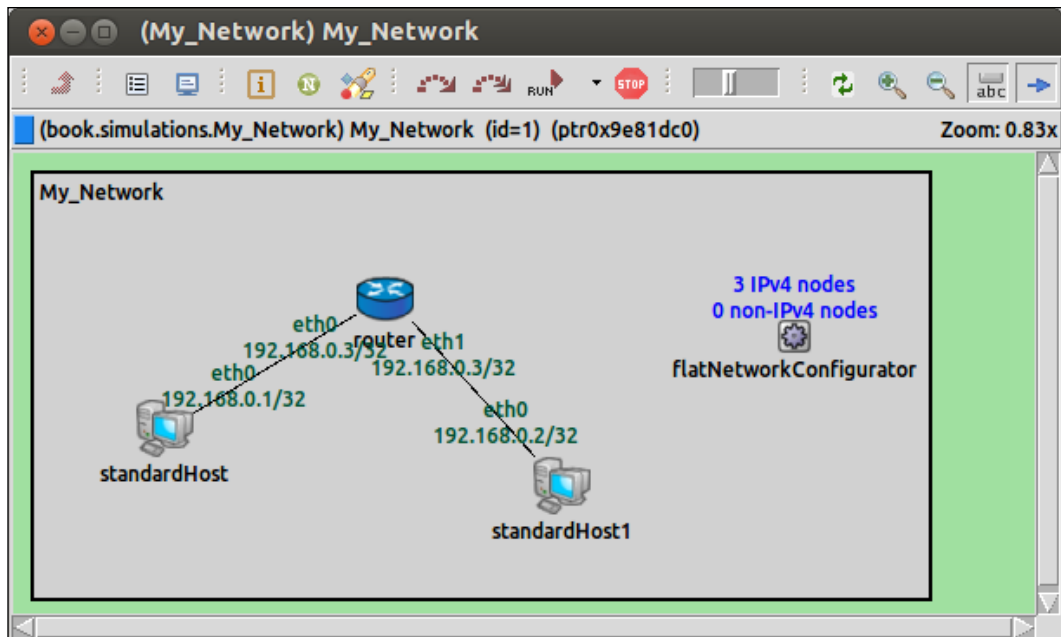
```
package book.simulations;

import inet.examples.httptools.socket.tenserverssocket
                                .ethernetline;
import inet.networklayer.autorouting.ipv4.FlatNetworkConfigurator;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;

network My_Network
```

```
{
  @display("bgb=620,293");
  submodules:
  standardHost: StandardHost {
    @display("p=90,176");
  }
  standardHost1: StandardHost {
    @display("p=365,215");
  }
  router: Router {
    @display("p=243,87");
  }
  flatNetworkConfigurator: FlatNetworkConfigurator {
    @display("p=527,112");
  }
  connections:
  standardHost.ethg++ <-->ethernetline<-->router.ethg++;
  router.ethg++ <-->ethernetline<--> standardHost1.ethg++;
}
```

The preceding code snippet creates the following network:



For this simulation to work, the following configuration file (`omnetpp.ini`) must be made as follows:

```
[General]
network = book.simulations.My_Network

#We will make standardHost a TCP Session Application in order for it
to #communicate
**.standardHost.numTcpApps = 1
**.standardHost.tcpApp[0].typename = "TCPSessionApp"
**.standardHost.tcpApp[0].connectAddress = "standardHost1"
**.standardHost.tcpApp[0].connectPort = 1000

#We will make standardHost1 a TCP Echo Application, this means that it
will send #an echo packet once it receives a packet.
**.standardHost1.numTcpApps = 1
**.standardHost1.tcpApp[0].typename = "TCPEchoApp"
**.standardHost1.tcpApp[0].localPort = 1000
**.standardHost1.tcpApp[0].echoFactor = 3.0

***.ppp[*].queueType = "DropTailQueue"
***.ppp[*].queue.frameCapacity = 10
***.eth[*].queueType = "DropTailQueue"
```

The first thing that we see in our configuration file is that the network to run is defined. The configuration file then goes on to describe how `standardHost` and `standardHost1` should behave. We connect `standardHost` to `standardHost1` with the following lines of code:

```
**.standardHost.tcpApp[0].connectAddress = "standardHost1"
**.standardHost.tcpApp[0].connectPort = 1000
```

The `**` at the start of the preceding lines mean that we don't have to use the following syntax instead:

```
My_Network.standardHost.tcpApp[0].connectAddress = "standardHost1"
My_Network.standardHost.tcpApp[0].connectPort = 1000
```

The `**` symbol is a wildcard, which works in this scenario as there is only one network it can pick from because I have only defined one.

Similar to the graphical editor for creating network topologies, there is also a form version for writing the configuration file of your network simulation.

You may prefer to use this instead of writing out the configuration file from scratch for the same reason as you may want to use the graphical editor to create the network topology for your network.

Simulation frameworks

There are many simulation framework packages for OMNeT++ that are freely available online. Whether you want to simulate peer-to-peer networks or simulate GPS navigation, there are simulation framework packages for almost everything you would want to do. You can also build on top of existing frameworks and create functionalities that don't exist. The following are some simulation framework packages that I think are worth mentioning:

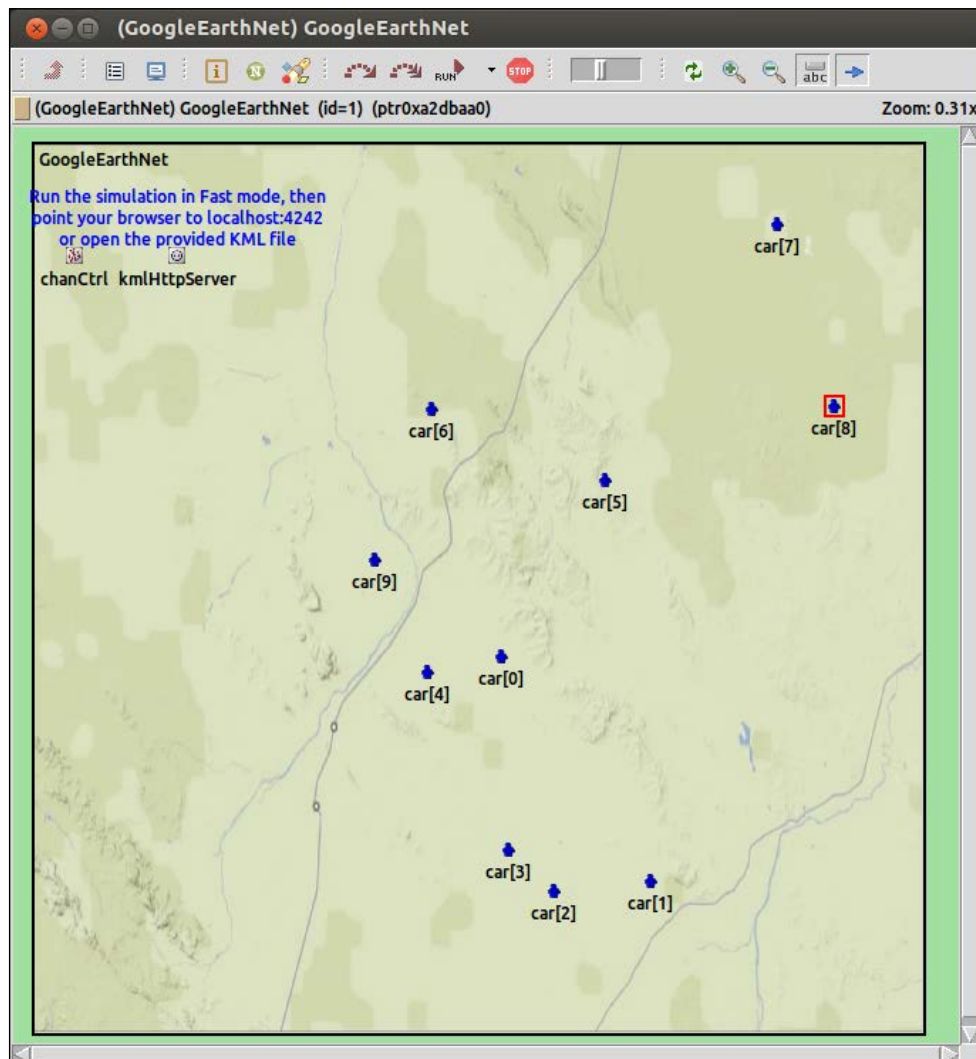
- **INET**: This is used throughout this book and allows simulations to include protocols such as the following:
 - IP
 - IPv6
 - TCP
 - UDP
- **HTTP Tools**: This allows you to simulate web browsers and web servers and uses StandardHost from INET.
- **EBitSim**: This allows you to create BitTorrent simulations. EBitSim makes use of the INET framework, and boasts of the capability of simulating 1000 nodes.
- **VoIPTool for INET**: This allows you to simulate the VoIP traffic and makes use of the INET framework.

A full list of frameworks can be found on the OMNeT++ website at <http://www.omnetpp.org/models/catalog>.

These framework packages can be used alongside each other to create extremely detailed and purposeful network simulations. There really is no need for you to reinvent the wheel with OMNeT++. With the power of online communities and the open source nature of OMNeT++, just about every type of network you will want to simulate can be built up from modules that other people have already made. In this way, OMNeT++ grows very organically and gets much better over time.

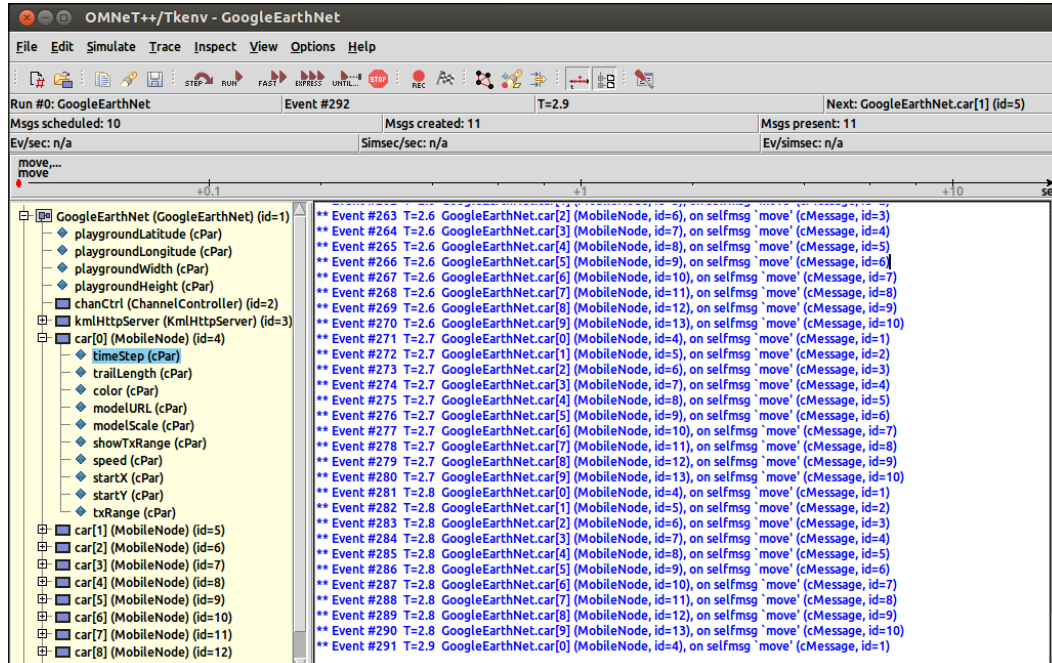
Google Earth demo

OMNeT++ 4.2.2 comes with a brilliant Google Earth demonstration. I'm including this to show what OMNeT++ is capable of and how many things it can be used for. The following is what the simulation produces when executed:

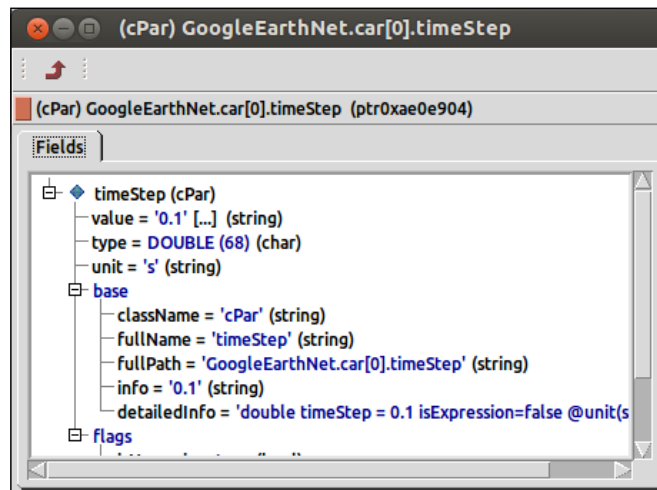


Not only do we see the cars moving around in the map, but we also see the output in another window that gives details on exactly what we are looking at in the simulation. All simulations will act in this manner.

The following is the other window that the simulation gives you:

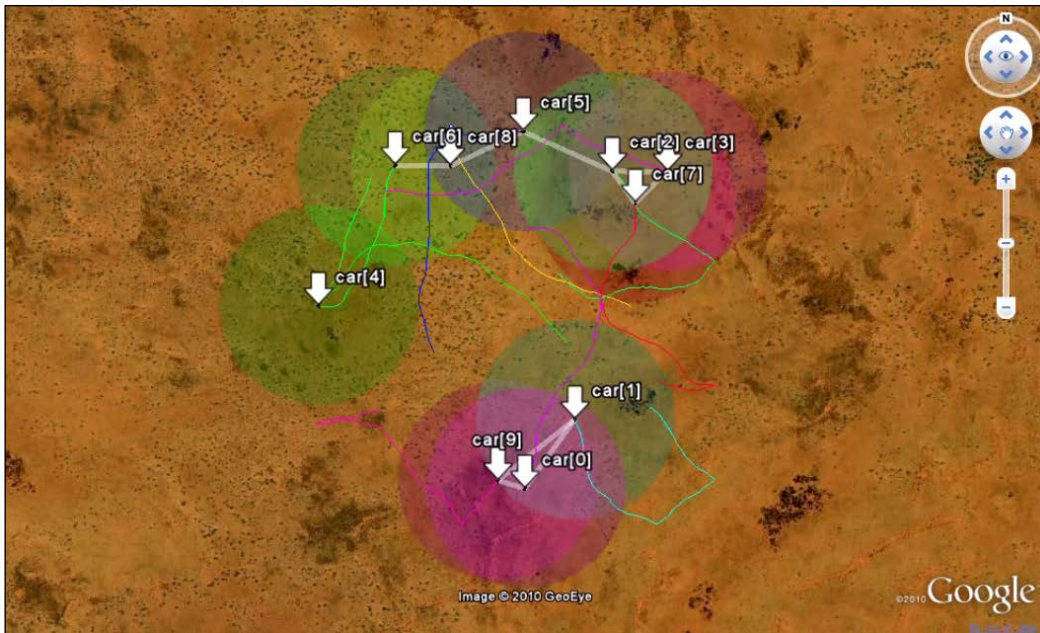


We can use this to control the speed of the simulation as well as read the output that the simulation may give. We can also hone in to specific components by double-clicking on them. If I double-click on `timeStep` under `car[0]`, I am presented with the following screen:



The special feature of this demo is its ability to also run on a Google Earth plugin that is available on most popular browsers. When we select **Fast** as the speed for the network, we can open our browser and go to `http://localhost:4242`.

This following is what we now see:



We see the cars moving across the map in real time. All the code can be found in the `samples/google-earth` folder in the OMNeT++ root directory.

INET in detail

The reason I am focusing on INET in this book is because many of the available simulation framework packages require INET to work. INET is a brilliant backbone to create your own frameworks from and is a great way of really getting to grips with OMNeT++!

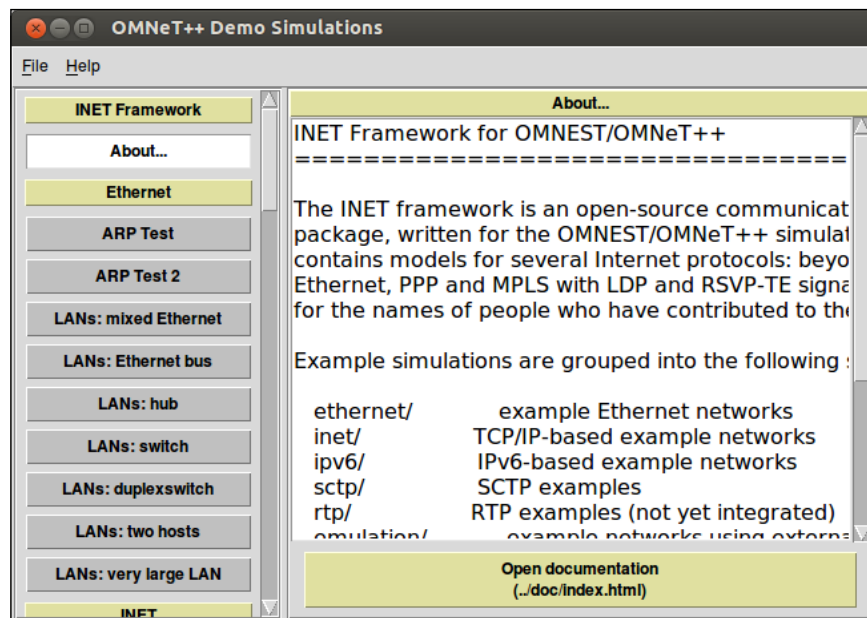
INET has been brilliantly documented and is available online for free at <http://inet.omnetpp.org/doc/INET/neddoc/index.html>.

The reference will give you a detailed look at the various components and capabilities included in INET.

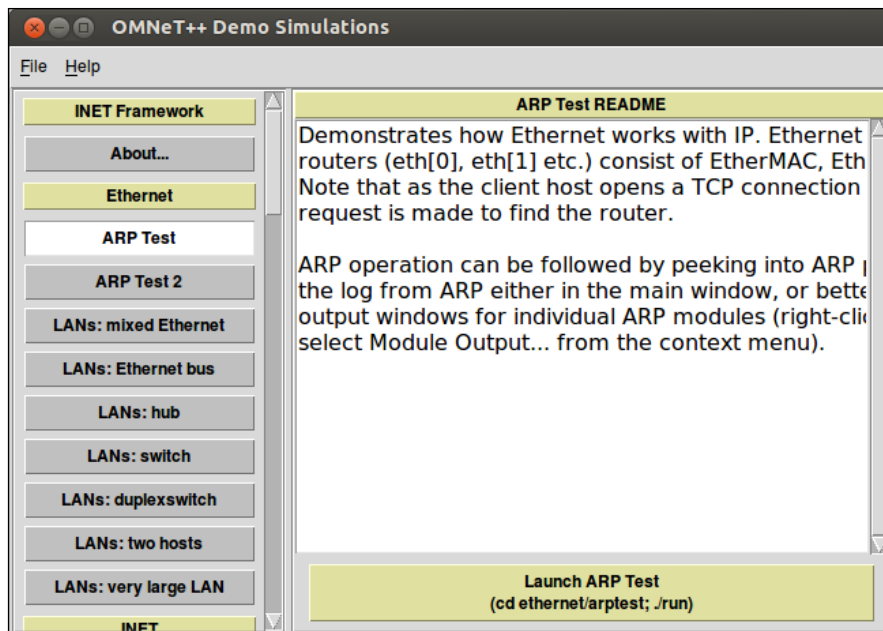
Example INET simulations

With INET built inside your OMNeT++ IDE, navigate to the `examples` folder in the **Project Explorer** pane. To run an example simulation in Linux, perform the following steps:

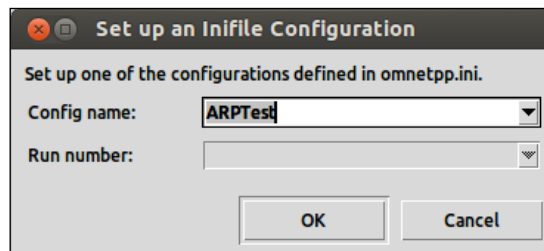
1. Locate the folder that INET has been built in and open the `examples` folder.
2. Now double-click on **run-demo** and select **Run in Terminal**.
3. Now you will see an interface that can be used to navigate and launch different simulation examples. Please do spend a bit of time getting familiar with these examples and remember that you can adapt these to create your own simulations. It will also be a good idea to read the **About...** page.



4. Let's now select an example by clicking on the left of the window. I'm going to select **ARP Test**. To launch the simulation, I will click on the **Launch ARP Test** button, as seen in the following screenshot:



5. When you see the following dialog box, just click on **OK** to proceed. What this dialog is asking is for you to select the configuration to use for this simulation that will be defined by the configuration `omnetpp.ini` file:



The configuration for the preceding simulation is as follows:

```
[General]
network = ARPTest
sim-time-limit = 500s
cpu-time-limit = 600s
total-stack = 2MiB
tkenv-plugin-path = ../../../../etc/plugins
#record-eventlog = true
#debug-on-errors = true

[ConfigARPTest]
# tcp apps
**.client.numTcpApps = 1
**.client.tcpApp[*].typename = "TCPSessionApp"
**.client.tcpApp[*].active = true
**.client.tcpApp[*].localAddress = ""
**.client.tcpApp[*].localPort = -1
**.client.tcpApp[*].connectAddress = "server"
**.client.tcpApp[*].connectPort = 1000
**.client.tcpApp[*].tOpen = 1.0s
**.client.tcpApp[*].tSend = 1.1s
**.client.tcpApp[*].sendBytes = 1MiB
**.client.tcpApp[*].sendScript = ""
**.client.tcpApp[*].tClose = 0

**.server.tcpApp[*].typename="TCPSinkApp"
**.server.numTcpApps = 1
**.server.tcpApp[*].typename = "TCPEchoApp"
**.server.tcpApp[0].localAddress = ""
**.server.tcpApp[0].localPort = 1000
**.server.tcpApp[0].echoFactor = 2.0
**.server.tcpApp[0].echoDelay = 0

# Ethernet NIC configuration
**.eth[*].mac.duplexMode = true

# Queues
**.ppp[*].queueType = "DropTailQueue"
**.ppp[*].queue.frameCapacity = 10
**.eth[*].queueType = "DropTailQueue"
**.eth[*].queue.dataQueue.frameCapacity = 10

# Ethernet switch
```

```

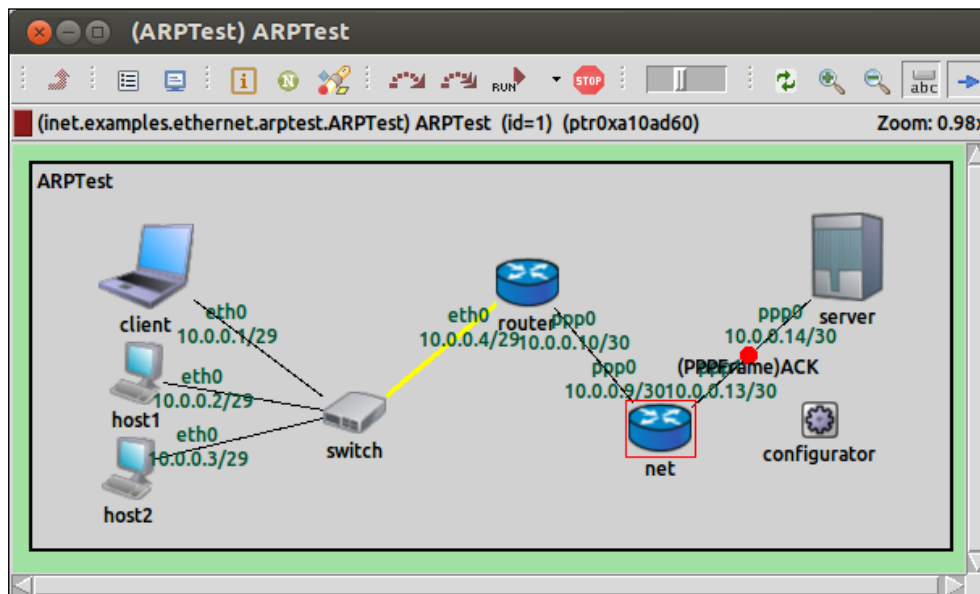
**.switch*.relayUnitType = "MACRelayUnitNP"
**.relayUnit.addressTableSize = 100
**.relayUnit.agingTime = 120s
**.relayUnit.bufferSize = 1MiB
**.relayUnit.highWatermark = 512KiB
**.relayUnit.pauseUnits = 300# pause for 300*512 bit (19200 byte) time
**.relayUnit.addressTableFile = ""
**.relayUnit.numCPUs = 2
**.relayUnit.processingTime = 2us

###.mac[*].txrate = 0 # autoconfig
**.mac[*].duplexMode = true

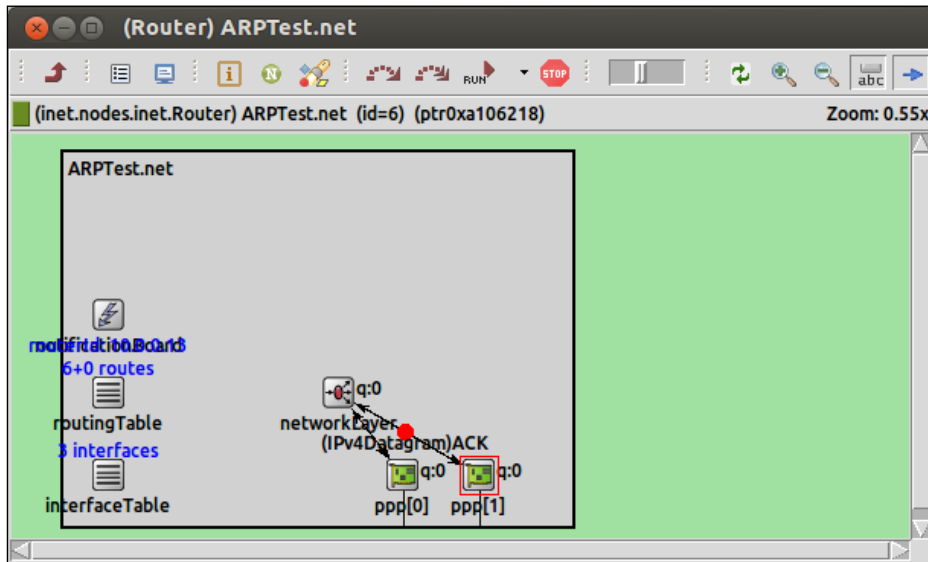
```

In the first few lines of the preceding code snippet, you can see [ConfigARPTTest]. If there were other configurations, you would be able to use the previous dialog box to select them. This allows you to set up many different scenarios for the network topology that you have created. [ConfigARPTTest] is one scenario for the network simulation. The second scenario could be called [ConfigARPTTest alternative], where different values are given to the parameters in the configuration.

You can now play with the example and explore the interface that it presents to give you a greater insight into the simulation. When you click on **RUN**, the simulation will start and will look like the following:



This gives us a very high-level look at what is going on. However, if we double-click on one of the objects in this simulation, it will take you deeper into what is happening. For example, if we click on the **net** router, we see the components that make up that compound module and also what those modules are currently doing, as shown in the following screenshot:



I recommend spending some time to modify the examples that come with INET and also the ones that can be found in the `samples` folder of your OMNeT++ folder. I find that the easiest way to learn is just to get involved, play around, explore the possibilities of what you can use all these functionalities for, and most importantly, have fun!

Summary

By now, you must have understood the OMNeT++ simulations and the components that make up the OMNeT++ environment.

4

Creating and Running a Simulation

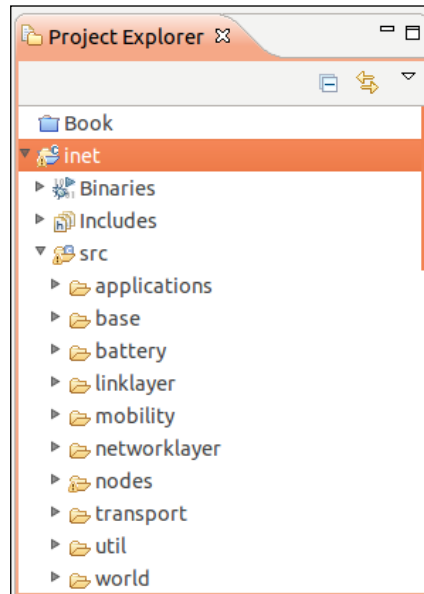
In this chapter, you will learn how to create your own network simulation and then run it using OMNeT++. After going through the previous chapters, you should have no problems understanding this chapter! You will learn the following topics:

- How to create the topology of your network using both the OMNeT++ IDE and the NED language
- How to create multiple scenarios for your simulation using the same network topology
- How to run and control the flow of your simulation

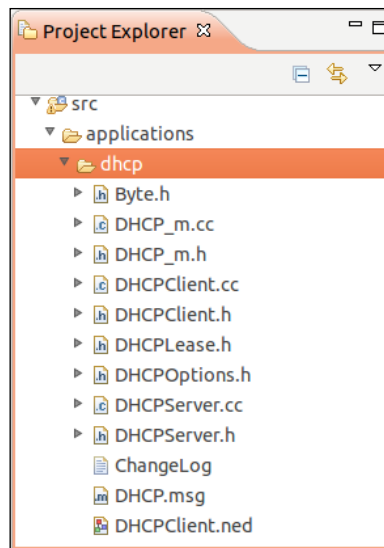
The OMNeT++ IDE

The OMNeT++ IDE provides you with the ideal environment to develop and run your OMNeT++ simulations. Based on the popular and extremely versatile Java-based Eclipse IDE platform, the OMNeT++ IDE is essentially a customized build of Eclipse. What's great about this development environment of OMNeT++ is that you're able to create network simulations without having to do any coding. However, if you need some extra functionality in your simulation, the OMNeT++ IDE lets you go as deep into the code as you want.

If we open up the OMNeT++ IDE, we should see the INET project that we previously set up. Using the **Project Explorer** on the left-hand side of the window, we can navigate through all the files and folders inside that project.



If you click on the triangle just on the left of the **src** folder, you will see a number of folders. These folders contain the source code that makes up the INET framework. This source code gives you the perfect template if you wish to write your own network protocols. The OMNeT++ IDE is also a fully-fledged development environment for C++ coding. Click on the triangle just on the left of the **applications** folder and then do the same for the **dhcp** folder. Now you'll see a listing of the source code that makes up an INET module for the Dynamic Host Configuration Protocol, as shown in the following screenshot:



To take a look at the source code for this module, just double-click on the **DHCPClient.cc** source file to open it. The OMNeT++ IDE provides really good code indentation and great syntax coloring which makes reading and writing code much easier.

The following is a snippet of the `DHCPClient.cc` file:

```
#include "DHCPClient.h"

#include "InterfaceTableAccess.h"
#include "IPv4InterfaceData.h"
#include "ModuleAccess.h"
#include "NotifierConsts.h"
#include "RoutingTableAccess.h"

Define_Module(DHCPClient);

DHCPClient::DHCPClient()
{
```



```
        timer_t1 = NULL;
        timer_t2 = NULL;
        timer_to = NULL;
        nb = NULL;
        ie = NULL;
        irt = NULL;
        lease = NULL;
    }

    DHCPClient::~DHCPClient()
    {
        cancelTimer_T1();
        cancelTimer_T2();
        cancelTimer_TO();
    }
}
```

The preceding code will make more sense as you continue through this chapter. Note how `DHCPClient` is parsed into a method called `Define_Module()`. It must be an OMNeT++ module!

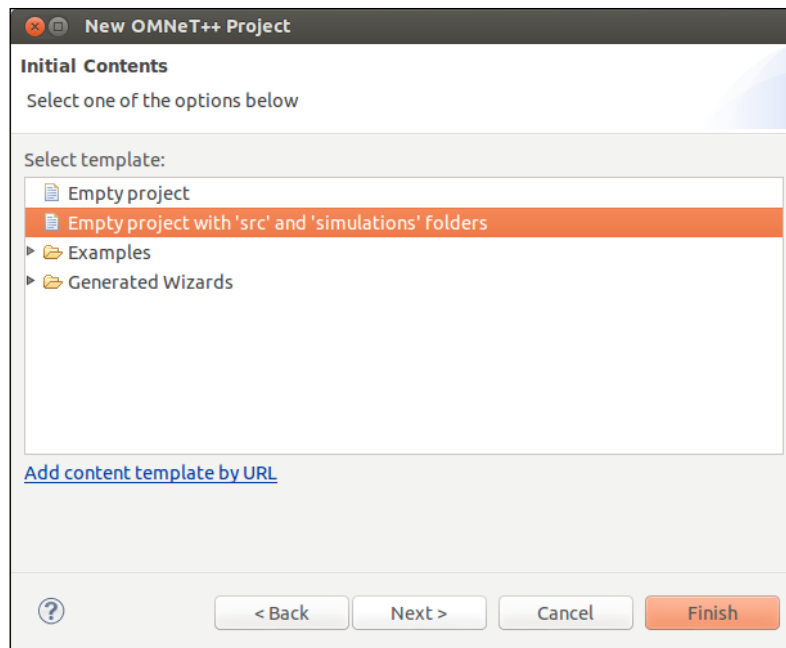
Creating a new project

There are many different ways you can create a new project. In this section, we will look at creating a completely empty but structured project, importing a project (for example, a project from a Git repository), and then we'll look at a very technical example project built in to OMNeT++ IDE that shows you how to program your own modules.

Creating an empty project

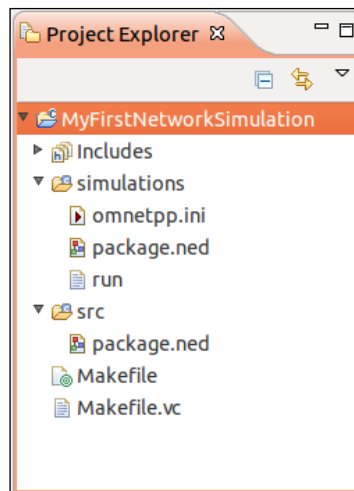
To create an empty project, perform the following steps:

1. Open the OMNeT++ IDE.
2. Navigate to **File | New | OMNeT++ Project...** or click the down-facing triangle on the **New** icon and select **OMNeT++ Project...**
3. Enter a name that you would like to give your new project and click on **Next >**.
4. Select **Empty project with 'src' and 'simulations' folders** as shown in the following screenshot:



5. Finally, click on **Finish** to have your project created.

The **Project Explorer** window in the OMNeT++ IDE will now have the following structure:



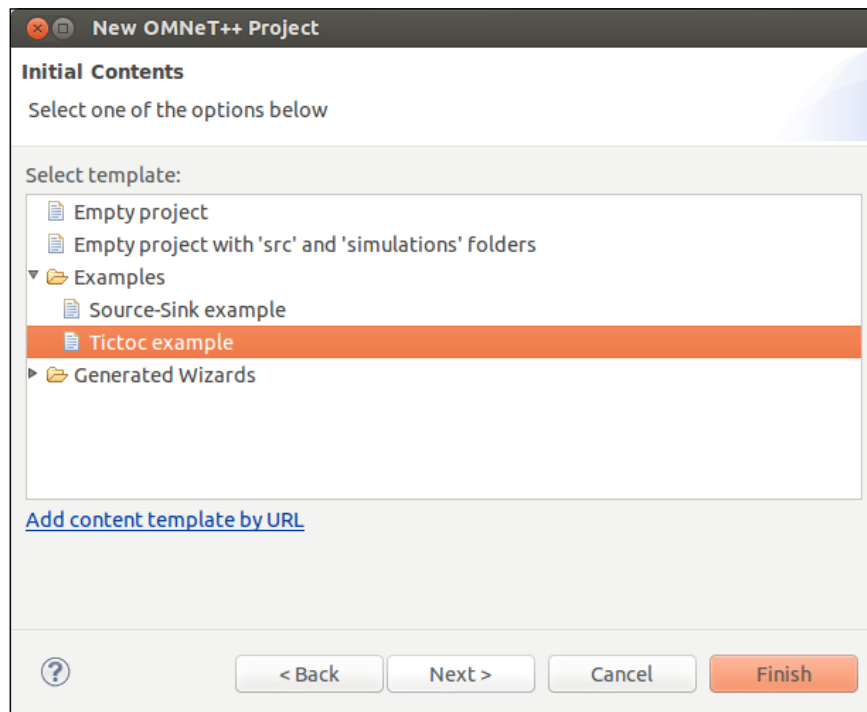
Importing a project

In *Chapter 2, Installing OMNeT++*, we imported INET into the OMNeT++ IDE so that it could be added as a library for your simulations to use. To import other projects, follow the same steps that you used to import the INET project.

Creating an example project

To create an empty project, perform the following steps:

1. Open the OMNeT++ IDE.
2. Navigate to **File | New | OMNeT++ Project...** or click the down-facing triangle on the **New** icon and select **OMNeT++ Project...**
3. Enter a name that you would like to give your new project and click on **Next >**.
4. Select the **Tictoc example** file in the **Examples** folder, as shown in the following screenshot:



5. Finally, click on **Finish** to create the **Tictoc example** project.

Let's first have a look at what we have understood already. In the newly created project, navigate to the **simulations** folder in the **Project Explorer** window of the OMNeT++ IDE and double-click on **Tictoc.ned** to open it.

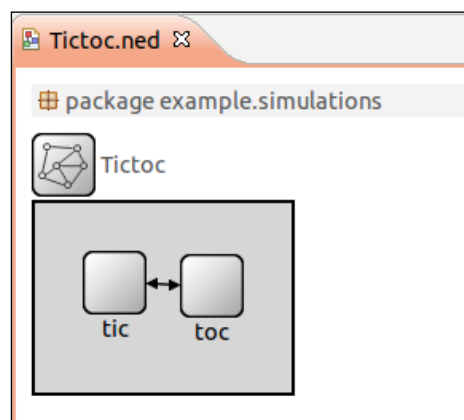
You should see something similar to what we have discussed previously in this book—this is the topology of a network called **Tictoc**. This network contains two modules, **tic** and **toc**. The source code that you can see by clicking on the **Source** tab next to the **Design** tab shows that this network topology is very basic:

```
package example.simulations;

import example.Txc;

//
// Two instances (tic and toc) of Txc connected.
//
network Tictoc
{
    submodules:
        tic: Txc;
        toc: Txc;
    connections:
        tic.out --> {delay = 100ms;} --> toc.in;
        tic.in <-- {delay = 100ms;} <-- toc.out;
}
```

We can see that **tic** and **toc** are submodules of **Txc**. The out gate of **tic** is connected to the in gate of **toc** and vice versa.



In the **src** folder of this project, double-click on **Txc.ned** and click on the **Source** tab. The following is a code snippet of the source:

```
package example;

//
// Immediately sends out any message it receives. It can optionally
// generate
// a message at the beginning of the simulation, to bootstrap the
// process.
//
simple Txc
{
    parameters:
        bool sendInitialMessage = default(false);
    gates:
        input in;
        output out;
}
```

This module contains one parameter, a Boolean value called `sendInitialMessage`, which has been given the default value `false`. The module contains one input gate called `in`, and one output gate called `out`. We can see these gates used in the `Tictoc` network topology as shown in the following line of code:

```
tic.out --> {delay = 100ms;} --> toc.in;
```

This example project, however, does not use a library like `INET`. In fact, this example has its own source code that defines the `Txc` module. In the **Project Explorer**, navigate to the **src** folder under the example project and double-click on the **Txc.cc** C++ source file. You will see the following source code:

```
#include "Txc.h"

namespace example {

Define_Module(Txc);

void Txc::initialize()
{
    if (par("sendInitialMessage").boolValue())
    {
        cMessage *msg = new cMessage("tictocMsg");
```

```

        send(msg, "out");
    }
}
void Txc::handleMessage(cMessage *msg)
{
    // just send back the message we received
    send(msg, "out");
}

}; // namespace

```

You can see that an OMNeT++ module is defined:

```
Define_Module(Txc);
```

This module has two methods: `initialize` and `handleMessage(cMessage *msg)`. The `initialize` method does a check on the `sendInitialMessage` parameter to see if it equals `true`. This parameter was defined in the `Txc` module's NED file as follows:

```
parameters:
    bool sendInitialMessage = default(false);
```

So we can see that by default, the value of `sendInitialMessage` is set to `false`. The `omnetpp.ini` configuration file can be used to set the value of `sendInitialMessage` to `true`. This is shown in the following code snippet:

```
[General]
network = Tictoc
cpu-time-limit = 60s
#debug-on-errors = true

**.tic.sendInitialMessage = true

```

This tells us that the submodule `tic` of the `Txc` module will send the initial message when the simulation is started.

In the `initialize` method, if the `sendInitialMessage` variable is equal to `true`, the following message is sent from the `out` gate:

```
if (par("sendInitialMessage").boolValue())
{
    cMessage *msg = new cMessage("tictocMsg");
    send(msg, "out");
}

```

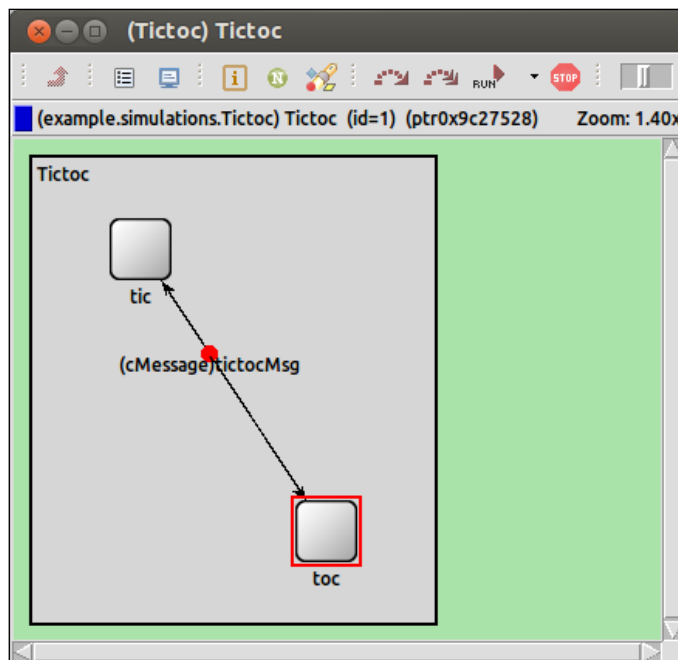
The network topology for this simulation tells us that the message is sent between the two submodules `tic` and `toc`. A message is sent from the `out` gate of `tic` to the `in` gate of `toc`. But what happens when `toc` receives this message? Like in every defined OMNeT++ module, there must be a `handleMessage` method. This method gets called within the module once it has received a message during the simulation runtime as follows:

```
void Txc::handleMessage(cMessage *msg)
{
    // just send back the message we received
    send(msg, "out");
}
```

This method takes the message that it received and sends it as a message from the module's `out` gate as the network topology tells us:

```
tic.out --> {delay = 100ms;} --> toc.in;
tic.in <-- {delay = 100ms;} <-- toc.out;
```

We can see that the `tic` and the `toc` node will constantly send a message back and forth as both the modules' `out` gates are connected to the other node's `in` gate, thus causing an infinite send and receive between the two nodes as shown in the following screenshot:



Let's now change the messages that go back and forth in order to say something else. Let's make `tic` say "Hello" and then `toc` reply with "Hello back!".

Let's first modify the `initialize()` method of `Txc.cc` to:

```
void Txc::initialize()
{
    if (par("sendInitialMessage").boolValue())
    {
        cMessage *msg = new cMessage("Hello");
        send(msg, "out");
    }
}
```

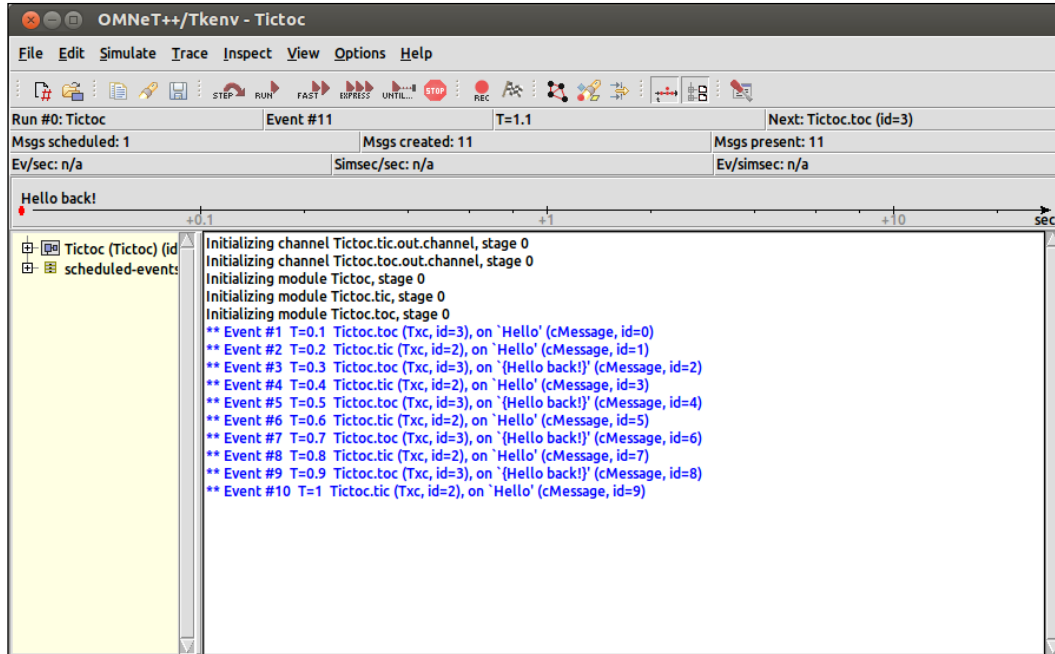
All I've done is changed the variable `msg` to store the string `Hello` instead of `tictocMsg`. That was simple enough, and now let's change the `handleMessage()` method to the following:

```
void Txc::handleMessage(cMessage *msg)
{
    if (strcmp("tic", getName()) == 0) {
        msg = new cMessage("Hello");
    } else {
        msg = new cMessage("Hello back!");
    }
    send(msg, "out");
}
```

The `if` statement checks to see if the message `id` value sent from a node is even or odd. If the `id` value is even, the message sent out is `Hello`, otherwise it's `Hello back!`. Let's look at a snippet from the output of the simulation log:

```
** Event #2  T=0.2  Tictoc.tic (Txc, id=2), on 'Hello' (cMessage, id=1)
** Event #3  T=0.3  Tictoc.toc (Txc, id=3), on '{Hello back!}' (cMessage,
id=2)
** Event #4  T=0.4  Tictoc.tic (Txc, id=2), on 'Hello' (cMessage, id=3)
** Event #5  T=0.5  Tictoc.toc (Txc, id=3), on '{Hello back!}' (cMessage,
id=4)
** Event #6  T=0.6  Tictoc.tic (Txc, id=2), on 'Hello' (cMessage, id=5)
** Event #7  T=0.7  Tictoc.toc (Txc, id=3), on '{Hello back!}' (cMessage,
id=6)
** Event #8  T=0.8  Tictoc.tic (Txc, id=2), on 'Hello' (cMessage, id=7)
** Event #9  T=0.9  Tictoc.toc (Txc, id=3), on '{Hello back!}' (cMessage,
id=8)
** Event #10 T=1    Tictoc.tic (Txc, id=2), on 'Hello' (cMessage, id=9)
```

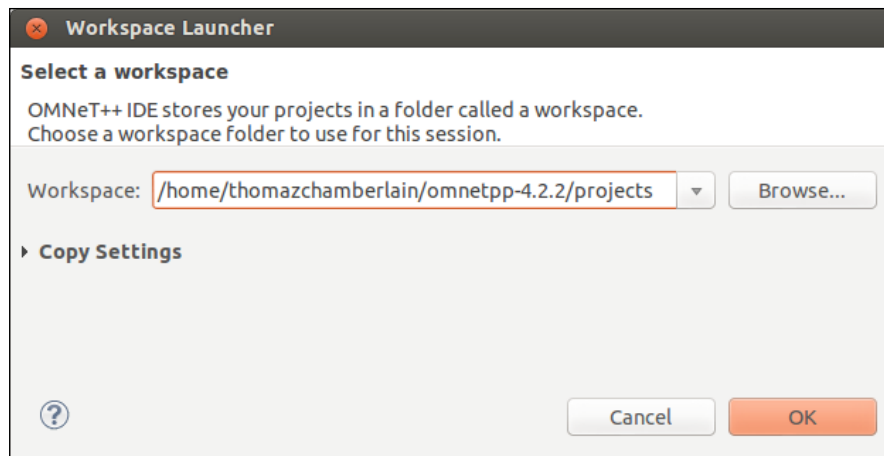

This shows us that the modification to the `Txc` node class was successful. This also shows us how we can use the simulation output log as a way to debug our network simulation. The simulation log appears when you start your network simulation environment. The following is what it looks like:



Switching to another workspace

When organizing your simulation projects, you may want to be extra neat and tidy and use different workspaces too. A **workspace** is a directory on your filesystem that is used to store project directories and all the files therein. So, you may want a workspace titled *Torrent Simulations*, for example, which you could use to hold all of your simulations that make use of torrent clients and P2P networks.

It's the directory that stores all of my current projects including INET, which is used as a library by other projects, in the same workspace. To create a new workspace to switch to, navigate to **File | Switch Workspace | Other inside of OMNeT++ IDE** as follows:



Click on the **Browse** button. Now you can browse your filesystem and create a new folder to select as your workspace. In Ubuntu, simply click on the **Create Folder** button, enter the workspace name, press *Enter*, and click on **OK** in the **Select Workspace Directory** window and then click on **OK** again in the **Workspace Launcher** window.

The OMNeT++ IDE will restart and you'll see what looks like a fresh installation.

Hello World network simulation

Now that you know how to create a new project inside OMNeT++, let's make a simulation from scratch. We will use the INET framework to save time and to get the simulation up and running as quickly as possible. To continue, you must first include INET as explained earlier.

What this simulation will do

This simulation will be simple, yet accurate. The simulation will contain five computers, all talking to a server via a router.

Defining your network

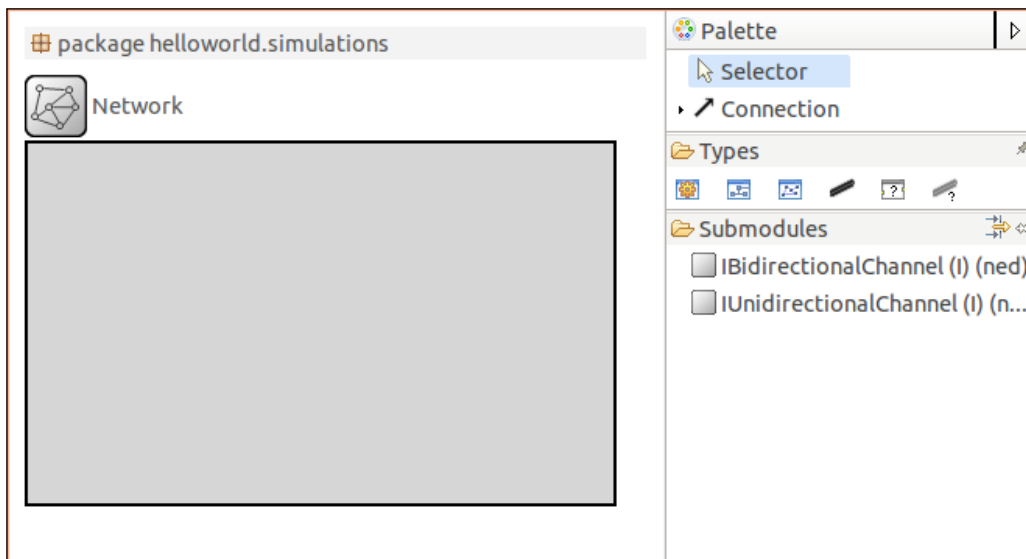
Start by creating an empty project with the `src` and `simulations` folders. I'm calling my simulation `HelloWorld`. With your new project open, the first step should be to create your network.

1. Navigate to the **simulations** folder and open **package.ned**. Now click on **Network** under **Types** in the **Palette** explorer and then click to the left on the opened **Design** tab. Alternatively, you could just start using the **Source** tab and add:

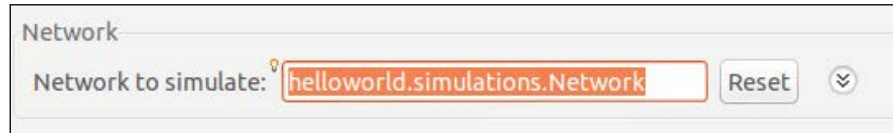
```
network Network
{
}

```

If the following is what you see in the **Design** tab, it means that you've done it correctly:



- Now, open the **omnetpp.ini** file under the **simulations** folder. The first thing to do is select the network to simulate. The network I just created is simply called **Network** and the package it belongs to is **helloworld.simulations**. With **omnetpp.ini** opened in the **Form** tab, select **General** from the left-hand side. Here you can select the network that you wish to simulate. Mine looks like the following:



- If you're not sure what to enter for **Network to simulate:**, just check the `package.ned` source and look at the first line. The first line of my `package.ned` source looks like the following:

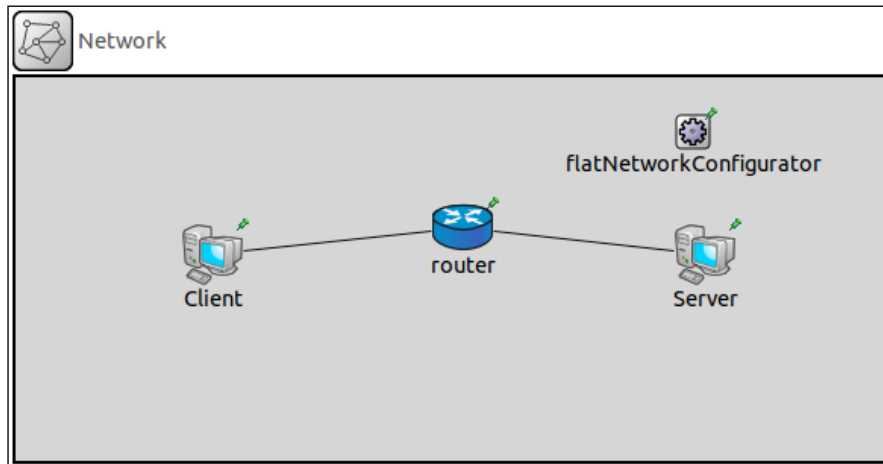
```
package helloworld.simulations;
```
- You can also clear the **Network to simulate:** textbox, tap *Ctrl* + Space bar and a list of available networks to select will appear.

Alternatively, you can do this by just using the **Source** tab instead of the **Form** view. As the network has already been selected, select the **Source** tab, and the following is what you will see:

```
[General]
network = helloworld.simulations.Network
```

- Let's now add the following four nodes to the network: a client, a server, a router, and a server. Going back to the Design view of your network topology, search for the following components from the **Palette** and add them to the network:
 - StandardHost x 2
 - A router
 - FlatNetworkConfigurator

The purpose of the `FlatNetworkConfigurator` submodule is to configure IPv4 addresses for the nodes to make up a "flat" network. The two `StandardHost` submodules will act as a client and server respectively. I've renamed the `StandardHost` submodules accordingly. The following is what your network topology should look like:



6. If you look at the source for this topology, you will see that I've decided to use Ethernet connections between these components. This suggests that the network is a LAN, as shown in the following code snippet:

```
package helloworld.simulations;

import inet.examples.httpptools.socket.tenserverssocket.
ethernetline;
import inet.networklayer.autorouting.ipv4.FlatNetworkConfigurator;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;

@license (LGPL);
//
// TODO documentation
//
network Network
{
    @display("bgb=575,253");
    submodules:
        Client: StandardHost {
```

```

        @display("p=130,117");
    }
    router: Router {
        @display("p=295,99");
    }
    Server: StandardHost {
        @display("p=455,117");
    }
    flatNetworkConfigurator: FlatNetworkConfigurator {
        @display("p=447,36");
    }
    connections:
        Client.ethg++ <--> router.ethg++;
        router.ethg++ <--> Server.ethg++;
}

```

7. Remember not to pay too much attention to the `@display()` tags as they just indicate where I have placed those submodules in the Network component. Note that the following license declaration is added in the code:

```
@license(LGPL);
```

Lesser General Public License (LGPL) means that the code is allowed to be copied and distributed, but changing it is not allowed. This license is added by default and can simply be removed or changed if you wish. I've kept it purely for the sake of this guide.

8. Now let's get the nodes of this network talking to each other. In order to do this, we must create configurations for the submodules in the `omnetpp.ini` file. This basic configuration will get the simulation running smoothly:

```

[General]
network = helloworld.simulations.Network

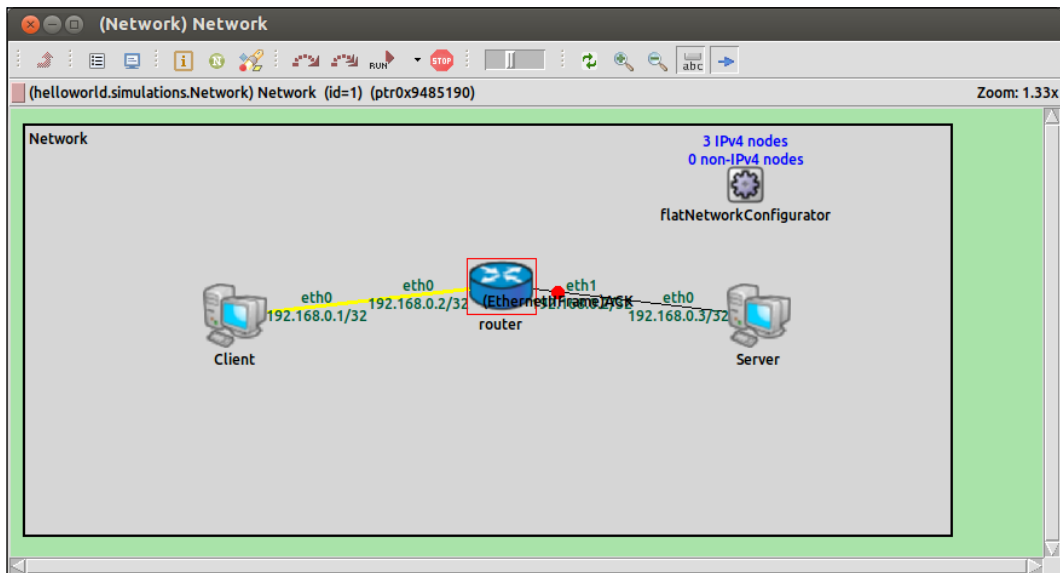
**.Client.numTcpApps = 1
**.Client.tcpApp[0].typename = "TCPBasicClientApp"
**.Client.tcpApp[0].connectAddress = "Server"
**.Client.tcpApp[0].connectPort = 80
**.Client.tcpApp[0].thinkTime = 0s
**.Client.tcpApp[0].idleInterval = 0s

**.Server.numTcpApps = 1
**.Server.tcpApp[0].typename = "TCPEchoApp"
**.Server.tcpApp[0].localPort = 80

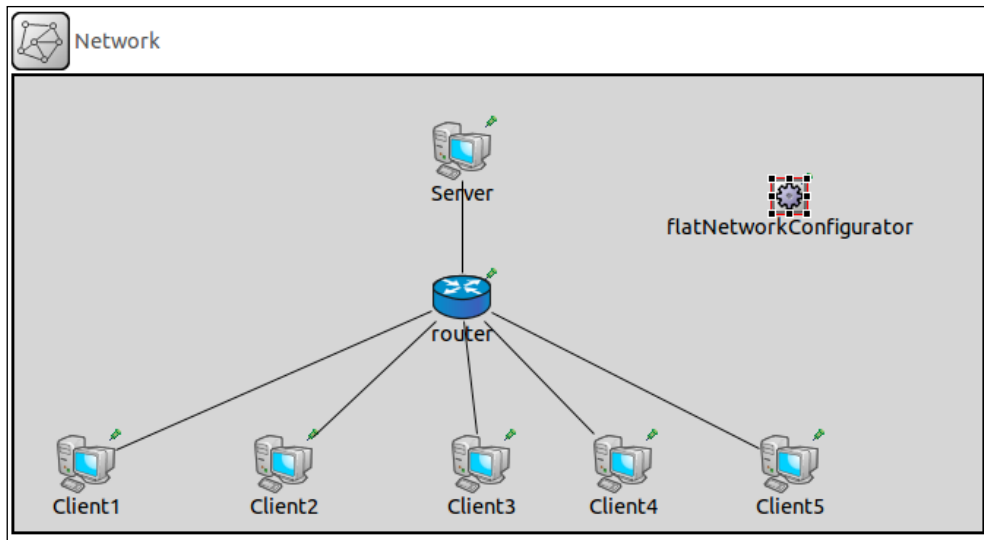
**.ppp[*].queueType = "DropTailQueue"
**.ppp[*].queue.frameCapacity = 10

```

9. This code is reasonably pragmatic, and you can see that the client is a basic TCP client application that talks to the server, which is a TCP echo application listening on port 80. The last two lines configure the network interface cards in the network.
10. You will now be able to run this simulation. You can use the same method that you learnt earlier. The following is what you will see when running the network:



11. You will see that everything is running as we configured it in the `omnetpp.ini` file. Let's now add four other client nodes to the Network topology. This is as easy as adding the first client node; simply add them from the **Palette** and name them accordingly. The Network topology should now look like the following:



The source for this new network topology is as follows:

```
package helloworld.simulations;

import inet.examples.httptools.socket.tenserverssocket.
ethernetline;
import inet.networklayer.autorouting.ipv4.FlatNetworkConfigurator;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;

@license (LGPL);
//
// TODO documentation
//
network Network
{
    @display("bgb=662,311");
    submodules:
```



```
Client1: StandardHost {
    @display("p=49,265");
}
router: Router {
    @display("p=306,151");
}
Server: StandardHost {
    @display("p=306,51");
}
flatNetworkConfigurator: FlatNetworkConfigurator {
    @display("p=530,82");
}
Client2: StandardHost {
    @display("p=184,265");
}
Client3: StandardHost {
    @display("p=319,265");
}
Client4: StandardHost {
    @display("p=416,265");
}
Client5: StandardHost {
    @display("p=530,265");
}
connections:
Client1.ethg++ <--> ethernetline <--> router.ethg++;
Client2.ethg++ <--> ethernetline <--> router.ethg++;
Client3.ethg++ <--> ethernetline <--> router.ethg++;
Client4.ethg++ <--> ethernetline <--> router.ethg++;
Client5.ethg++ <--> ethernetline <--> router.ethg++;
router.ethg++ <--> ethernetline <--> Server.ethg++;
}
```

This will be a good time to introduce the method of using arrays to organize similar nodes better; for example, there are five clients that all share the same parameters and are of the same submodule type.

Let's first add a parameters section to our code and add an integer variable just below the instance where the network is first declared, as shown in the following code snippet:

```
network Network
{
    parameters:
        int n;
```

This variable will store the number of clients on the network. This is done by removing all five of the client definitions from the source code and replacing it with only one as follows:

```
Client[n]: StandardHost {
}
```

12. The code is now much tidier and easier to read. Tidy code will make larger projects much easier to manage. The last thing that needs to be changed is in the connections section; first delete all the lines in that section that start with `Client`. Now write the following in the connections section:

```
for i=0..n-1 {
    Client[i].ethg++ <--> ethernetline <--> router.ethg++;
}
```

The following is the updated Network topology source code:

```
package helloworld.simulations;

import inet.examples.httpptools.socket.tenserverssocket.
ethernetline;
import inet.networklayer.autorouting.ipv4.FlatNetworkConfigurator;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;

@license (LGPL);
//
// TODO documentation
//
network Network
{
    parameters:
        int n;
        @display("bgb=662,311");
    submodules:
        Client[n]: StandardHost {
        }
        router: Router {
            @display("p=306,151");
        }
        Server: StandardHost {
            @display("p=306,51");
        }
}
```

Creating and Running a Simulation

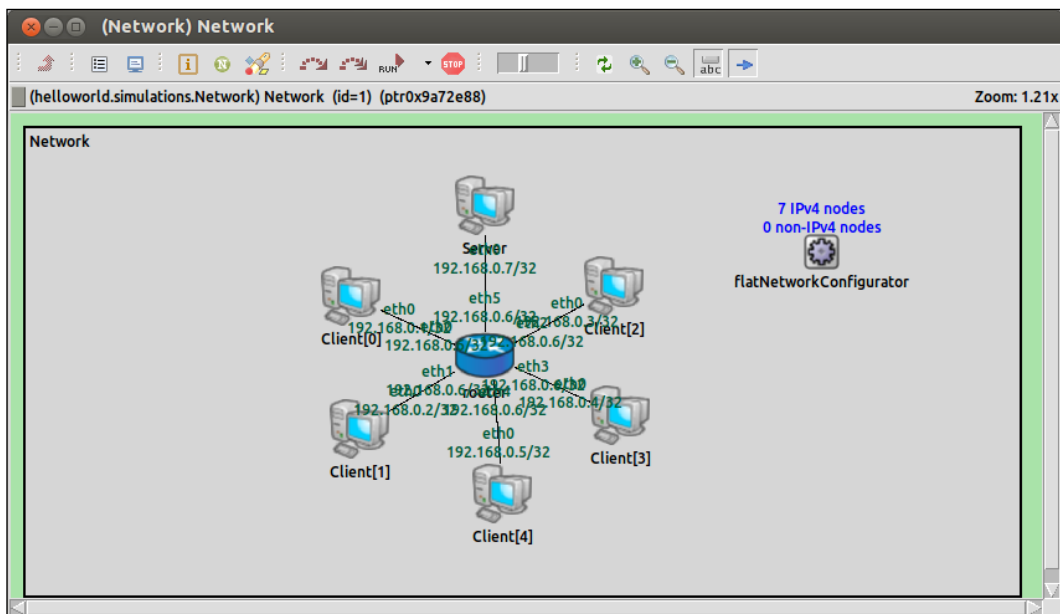
```
flatNetworkConfigurator: FlatNetworkConfigurator {
    @display("p=530,82");
}

connections:
for i=0..n-1 {
    Client[i].ethg++ <--> ethernetline <--> router.ethg++;
}
router.ethg++ <--> ethernetline <--> Server.ethg++;
}
```

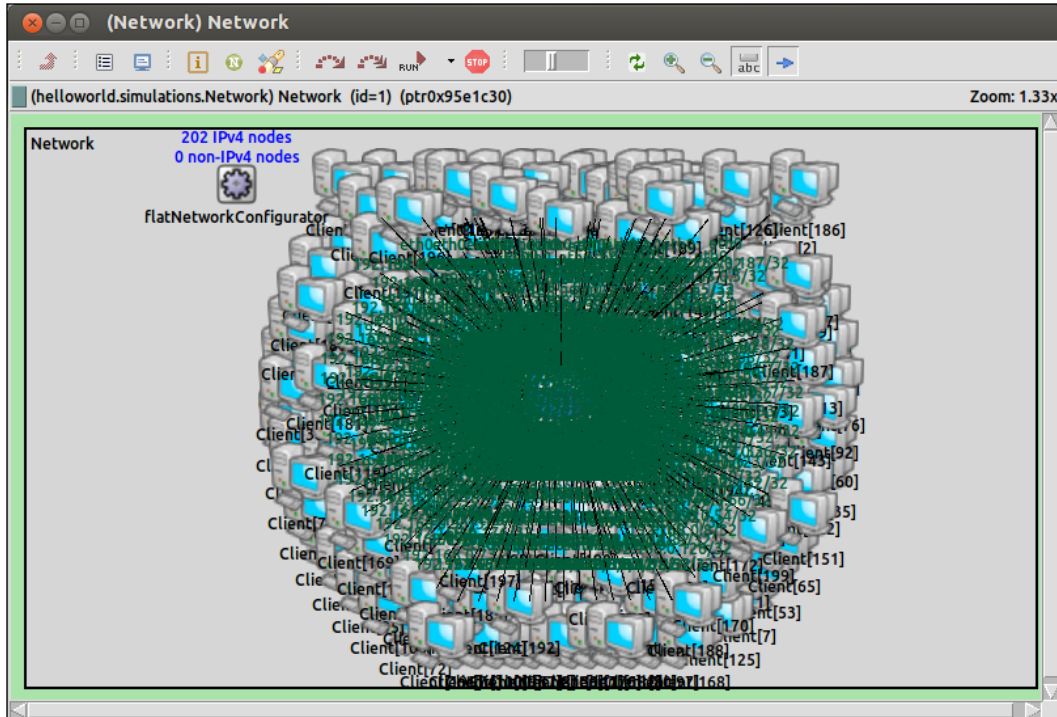
13. Now add the following line anywhere in the `omnetpp.ini` file:

```
*.n = 5
```

This sets the number of client nodes to 5. When you run the simulation again, the following is what you'll see:



14. You can now set any number of client nodes you wish and try out different values to see what's possible. The following is what the network looks like with the value of n set to 200:



This looks overwhelming; however, it still runs and the simulation's output log still gives the same insight into the simulation. Note that with this many nodes in the simulation, it will take much longer to run, depending on the processor speed you have available.

15. The following is the code of an INET example Network topology which also makes use of arrays:

```
package inet.examples.inet.nclients;

import inet.networklayer.autorouting.ipv4.Ipv4NetworkConfigurator;
import inet.nodes.inet.Router;
import inet.nodes.inet.StandardHost;
```

```
import ned.DatarateChannel;

network Nclients2
{
    parameters:
        int numRouters;
        int hostsPerRouter;
    types:
        channel ethernetline2 extends DatarateChannel
        {
            delay = 0.1us;
            datarate = 100Mbps;
        }
        channel gigabitline2 extends DatarateChannel
        {
            delay = 0.1us;
            datarate = 1Gbps;
        }
    submodules:
        configurator: IPv4NetworkConfigurator;
        r[numRouters]: Router;
        cli[numRouters*hostsPerRouter]: StandardHost {
            parameters:
                @display("i=device/laptop_vs");
        }
        srv: StandardHost {
            parameters:
                @display("i=device/server_1");
        }
    connections:
        for i=0..numRouters-1, for j=0..hostsPerRouter-1 {
            cli[i*hostsPerRouter+j].pppg++ <--> ethernetline2 <-->
r[i].pppg++;
        }
        for i=0..numRouters-2 {
            r[i].pppg++ <--> gigabitline2 <--> r[i+1].pppg++;
        }
        r[numRouters-1].pppg++ <--> ethernetline2 <--> srv.pppg++;
    }
}
```

The preceding code can be found at `inet/examples/inet/nclients/Nclients2.ned`.

Creating multiple scenarios

At the moment, the simulation that we have just created only uses one configuration; this is the one under `General` in the `omnetpp.ini` file. Let's add two scenarios that the simulation can use. Scenario 1 will have five client nodes, whereas scenario 2 will have ten client nodes. The following is what the updated `omnetpp.ini` file looks like:

```
[Config Scenario_One]

*.n = 5

[Config Scenario_Two]

*.n = 10

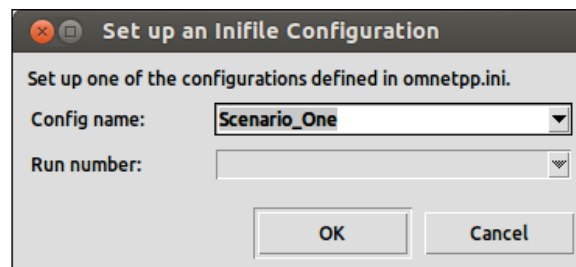
[General]
network = helloworld.simulations.Network

**.Client*.numTcpApps = 1
**.Client*.tcpApp[0].typename = "TCPBasicClientApp"
**.Client*.tcpApp[0].connectAddress = "Server"
**.Client*.tcpApp[0].connectPort = 80
**.Client*.tcpApp[0].thinkTime = 0s
**.Client*.tcpApp[0].idleInterval = 0s

**.Server.numTcpApps = 1
**.Server.tcpApp[0].typename = "TCPEchoApp"
**.Server.tcpApp[0].localPort = 80

**.ppp[*].queueType = "DropTailQueue"
**.ppp[*].queue.frameCapacity = 10
```

When you try to run your simulation now, you will be presented with a new dialog box asking you which configuration to select to run:



Click the **OK** button to continue to your simulation.

Let's now look at a more sophisticated example of using multiple scenarios:

```
[Config inet__inet]
description = "inet_TCP <---> inet_TCP"
# setting TCP stack implementation
**.srv*.tcpType = "TCP"
**.cli*.tcpType = "TCP"
**.srv.numPcapRecorders = 1
**.srv.pcapRecorder[0].pcapFile = "results/inet_srv.pcap"

[Config lwip__lwip]
description = "TCP_lwIP <---> TCP_lwIP"
# setting TCP stack implementation
**.srv*.tcpType = "TCP_lwIP"
**.cli*.tcpType = "TCP_lwIP"
**.srv.numPcapRecorders = 1
**.srv.pcapRecorder[0].pcapFile = "results/lwip_srv.pcap"

[Config lwip__inet]
description = "TCP_lwIP <---> inet_TCP"
# setting TCP stack implementation
**.srv*.tcpType = "TCP_lwIP"
**.cli*.tcpType = "TCP"

[Config inet__lwip]
description = "inet_TCP <---> TCP_lwIP"
# setting TCP stack implementation
**.srv*.tcpType = "TCP"
**.cli*.tcpType = "TCP_lwIP"

[General]
network = Nclients
#debug-on-errors = true
tkenv-plugin-path = ../../../../etc/plugins

sim-time-limit = 1000000s

# number of client computers
*.n = 4

# tcp apps
**.cli[*].numTcpApps = 1
**.cli[*].tcpApp[*].typename = "TelnetApp"
```

```

**.cli[*].tcpApp[0].localAddress = ""
**.cli[*].tcpApp[0].localPort = -1
**.cli[*].tcpApp[0].connectAddress = "srv"
**.cli[*].tcpApp[0].connectPort = 1000

**.cli[*].tcpApp[0].startTime = exponential(5s)
**.cli[*].tcpApp[0].numCommands = exponential(10)
**.cli[*].tcpApp[0].commandLength = exponential(10B)
**.cli[*].tcpApp[0].keyPressDelay = exponential(0.1s)
**.cli[*].tcpApp[0].commandOutputLength = exponential(40B)
**.cli[*].tcpApp[0].thinkTime = truncnormal(2s,3s)
**.cli[*].tcpApp[0].idleInterval = truncnormal(3600s,1200s)
**.cli[*].tcpApp[0].reconnectInterval = 30s
**.cli[*].tcpApp[0].dataTransferMode = "object"

**.srv.numTcpApps = 1
**.srv.tcpApp[*].typename = "TCPGenericSrvApp"
**.srv.tcpApp[0].localAddress = ""
**.srv.tcpApp[0].localPort = 1000
**.srv.tcpApp[0].replyDelay = 0

# NIC configuration
**.ppp[*].queueType = "DropTailQueue" # in routers
**.ppp[*].queue.frameCapacity = 10 # in routers

# turn on throughput stat
**.channel.throughput.result-recording-modes=+last

```

The preceding code can be found at `inet/examples/inet/nclients/omnetpp.ini`.

Controlling the flow of your simulation

There are different ways to control the flow of your simulation inside the simulation environment using the following toolbar:

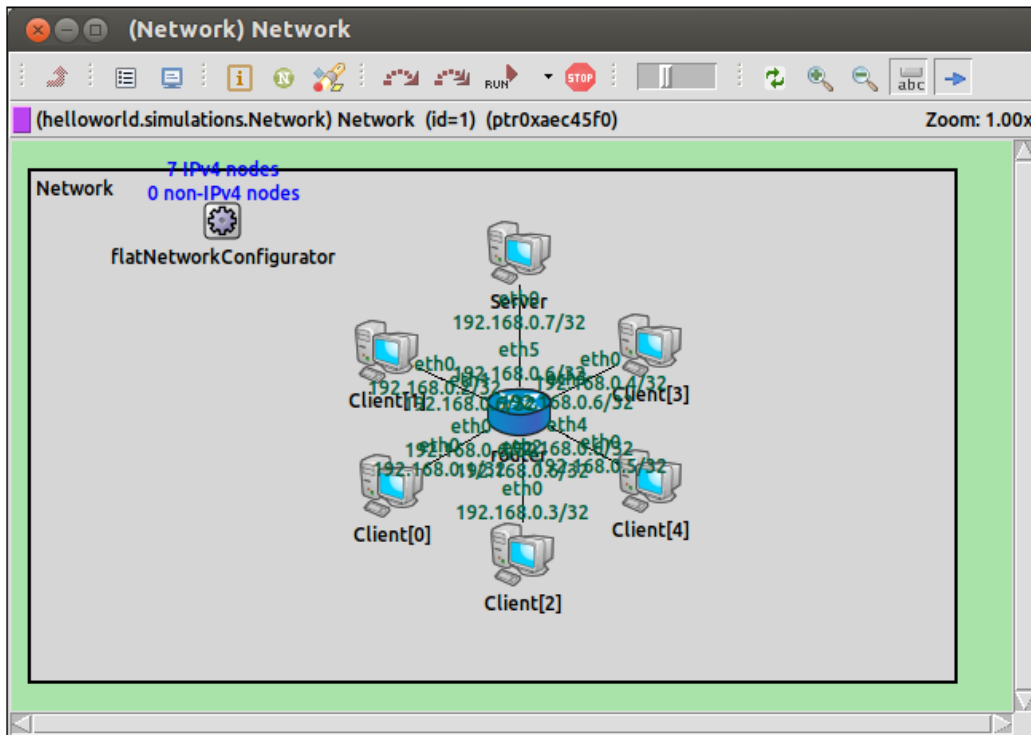


From left to right, the displayed buttons perform the following:

1. Run Until next event in the module.
2. Fast Run install the next event in the module.
3. Run.

4. Select Run, Fast Run, Express Run, or Run Until.
5. Stop.

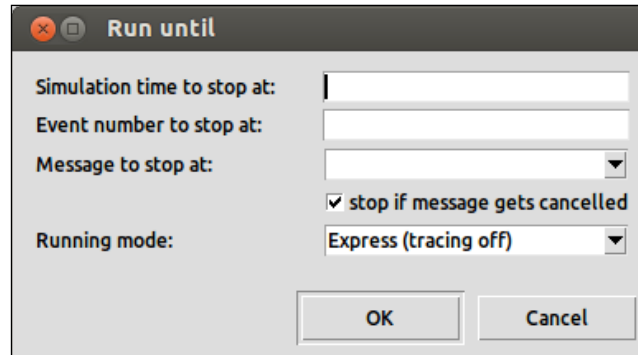
The following is a screenshot of a network simulation to remind you where this toolbar can be found:



The following table describes the flow types available when running a network simulation:

Flow Type	Description
Run	This runs the simulation normally, allowing you to visualize what is happening.
Fast Run	This runs the simulation much faster, doing 1000 events or more every other second.
Express Run	This runs the simulation at top speed, doing 100,000 events or more every other second.
Run Until	This allows you to run the simulation to a certain time or event in the simulation.

Selecting Run Until will display the following dialog box to tell the simulation when to stop:



Summary

We have now covered everything set out in this chapter. You should now know how to create OMNeT++ projects using the OMNeT++ IDE. This chapter has also shown you how to modify an example project to create your own network simulation. We have now covered how to create network topologies using the NED language and also graphically using the OMNeT++ IDE, as well as using arrays to allow for easy scalability of any network simulation. You should also now know how to configure multiple scenarios for your network simulation and then control its flow using the OMNeT++ IDE.

5

Learning from Your Simulations

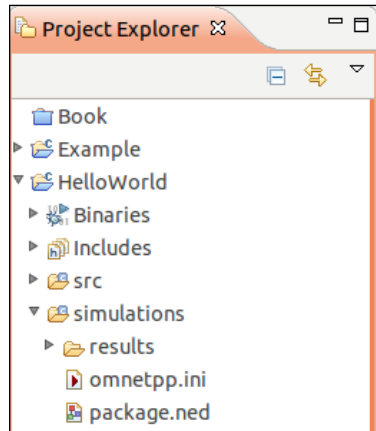
The main reason to simulate a network is to try and gain some insight into exactly what a network design will yield without having to first create that simulation in the real world. Creating and running your simulation isn't enough to achieve this; you must also gather data from your simulations and create meaningful visualizations. These visualizations may show a breaking point in your design, a particular pattern, inefficiency, or anything else that suggests bad designing.

Gathering useful information

Gathering data from your network simulation is not a difficult task to do using OMNeT++. OMNeT++ is able to gather vector and scalar data and also create realistic packet capture files. To enable your simulation to gather data, you must include the following extra line in the simulation's configuration `omnetpp.ini` file:

```
record-eventlog = true
```

Now when you run your simulation, you'll see a folder titled `Results` appear in the same folder where the `omnetpp.ini` folder is as shown in the following screenshot:

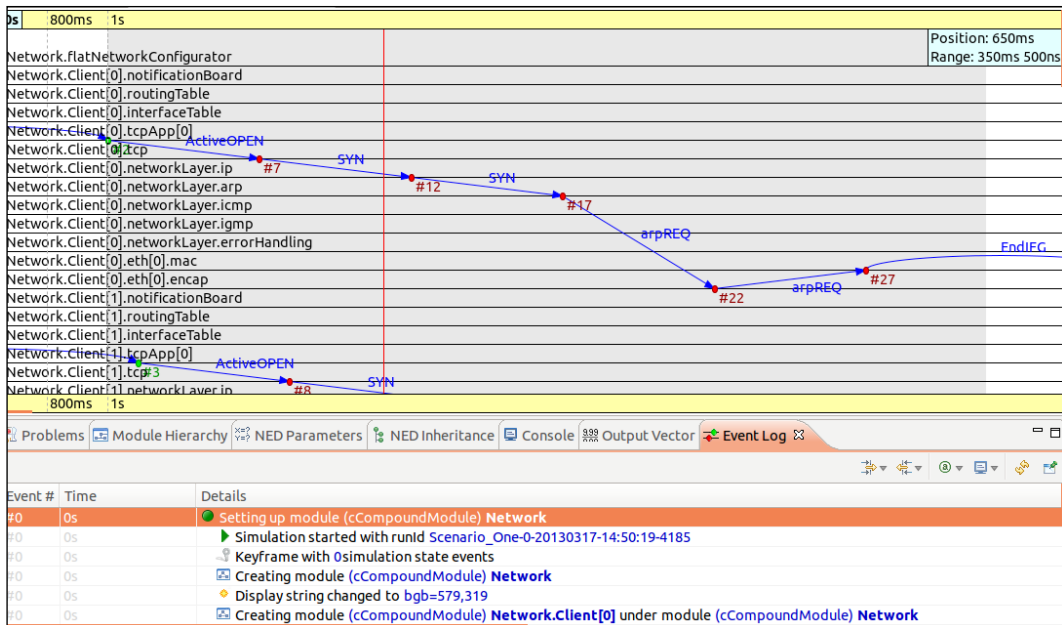


This folder is created because the `record-eventlog` is set to `true`.

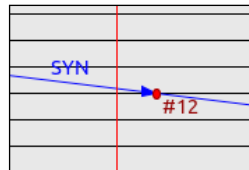
Inside this new folder, you'll see four new files and the names of these files will depend on the configuration you choose for your simulation. I chose a configuration called `Scenario_One`; therefore the four files are named as follows:

File Name	Description
<code>Scenario_One-0.elog</code>	This file is the event log of the simulation.
<code>Scenario_One-0.sca</code>	This file stores scalar data that was recorded during the simulation.
<code>Scenario_one-0.vec</code>	This file stores vector data that was recorded during the simulation.
<code>Scenario_One-0.vci</code>	This is an index file for the vector file generated. This index file allows the vector file to be accessed much faster.

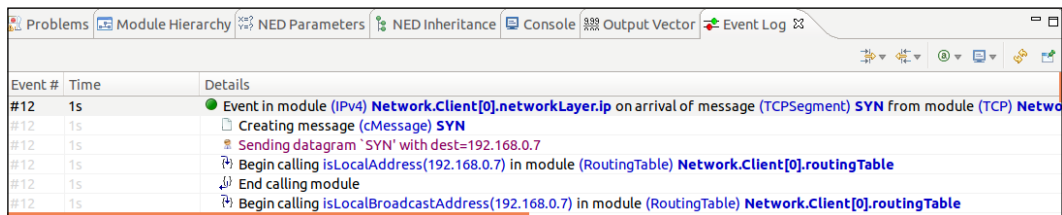
Let's have a look at the event log file that was generated from running the `HelloWorld` example where `Scenario_One` was selected as the configuration. To open this file in **Project Explorer**, I double-clicked on the file. This is what the file looks like when opened inside OMNeT++:



This file shows us a breakdown of every event that has occurred during the simulation. By clicking on one of the red circles in the upper part of the file, we can see a detailed list of what is happening in that particular event. I clicked on the event #12 (click the red circle) as shown in the following screenshot:

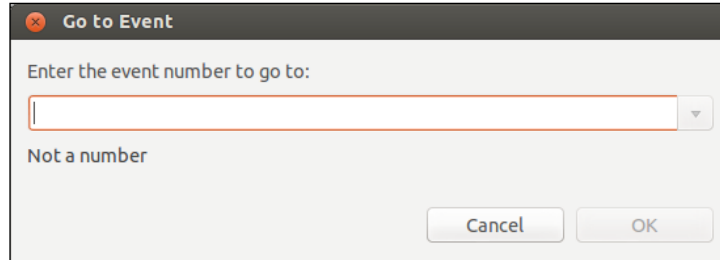


I was then taken to the event in the **Event Log** tab:



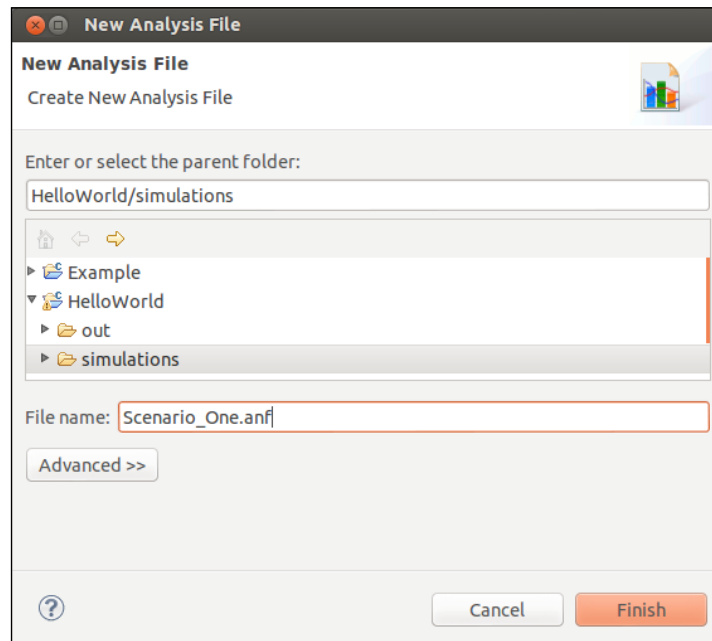
So from this detailed list, we can see that a message is sent as an SYN packet to the IP address 192.168.0.7.

You can also navigate to the event log by right-clicking inside the diagram view and then navigating to **Go To | Go to Event**. You will see the following dialog box:



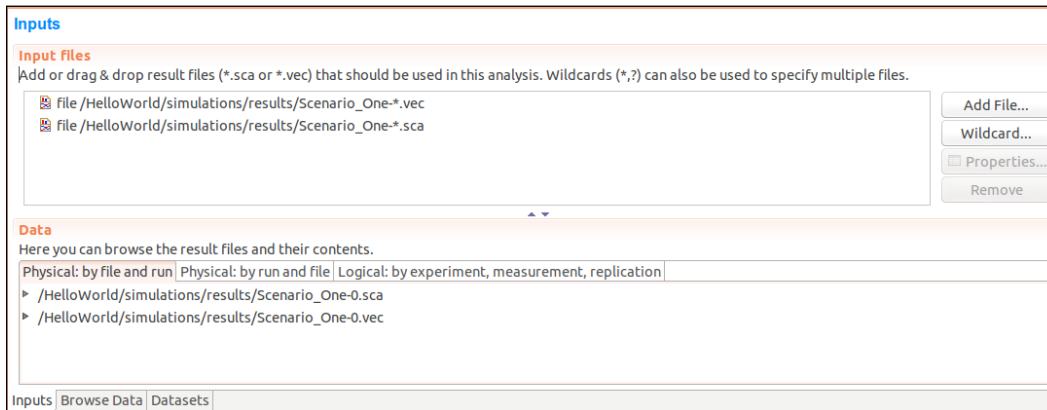
Now just type the event number you'd like to go to, then click on the **OK** button, and it will navigate to that event. Similarly, you can right-click inside the diagram view and navigate to **Go To | Go to Simulation Time...** if you wish to navigate to a particular time in the simulation instead of a particular event. If you know that your network starts behaving strangely at a certain time in the simulation, this feature is very useful in order to see exactly what is happening at that time and which events are occurring.

The other files generated are the scalar and vector data files, which allow us to create insightful data visualizations. When you double-click to open one of these files, you will see the following screenshot:



This dialog asks you to create an analysis file. To use the vector and scalar data files, you must first create an analysis file which will use all the available vector and scalar files.

Click on the **Finish** button for OMNeT++ to create the analysis file for you. OMNeT++ will now open the analysis file and you will see the following screen:

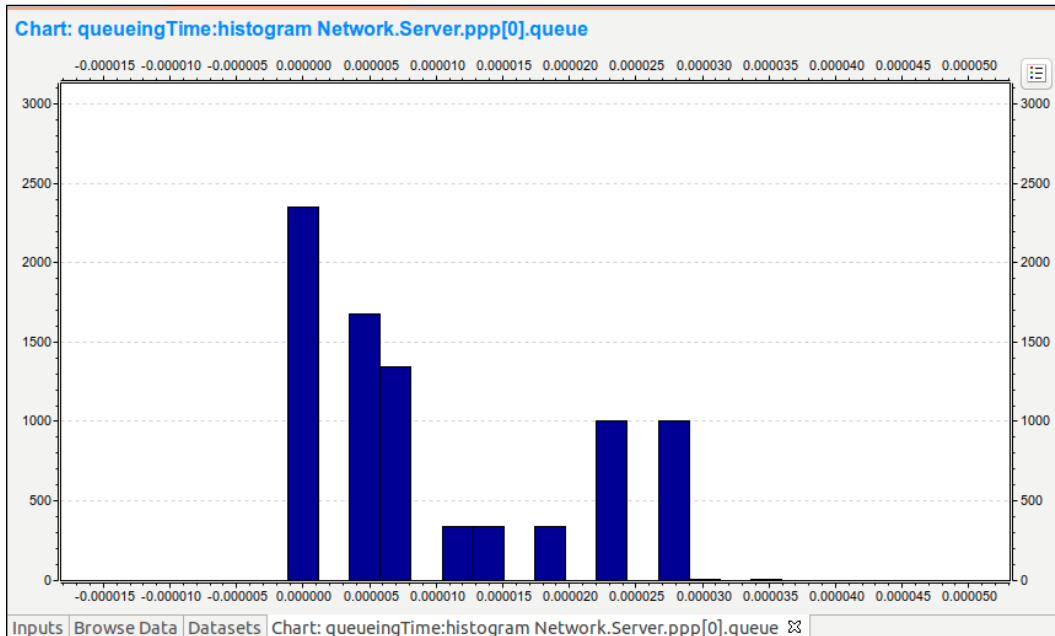


Click on the **Browse Data** tab at the bottom and then click on the **Histograms** tab at the top. Now you will see a list of histograms displayed as follows:

The screenshot shows the 'Browse Data' dialog box in OMNeT++. The 'Histograms (18 / 18)' tab is selected. The table below shows a list of histograms with the following columns: Folder, File name, Config name, Run id, Module, Name, Count, Mean, StdDev, and Variance.

Folder	File name	Config name	Run id	Module	Name	Count	Mean	StdDev	Variance
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.Client[4].tcpA	endToEndDelay:histogra	336	0.0	0.0	0.0
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.router.ppp[0]	queueingTime:histogram	1681	0.0	0.0	0.0
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.router.ppp[1]	queueingTime:histogram	1681	0.0	0.0	0.0
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.router.ppp[2]	queueingTime:histogram	1681	0.0	0.0	0.0
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.router.ppp[3]	queueingTime:histogram	1680	0.0	0.0	0.0
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.router.ppp[4]	queueingTime:histogram	1680	0.0	0.0	0.0
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.router.ppp[5]	queueingTime:histogram	10084	1.69729	1.72112	2.962256178703902E-10
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.Server.ppp[0]	queueingTime:histogram	8403	9.81514	9.91476	9.830259210961405E-11
/HelloW	Scenario_One	Scenario_C_0	Scenario_One	Network.Server.tcpApp	endToEndDelay:histogra	1680	0.0	0.0	0.0

After clicking on **queueingTime** named **dataset**, I can right-click on it and then click on **Plot** which would show me the following screenshot:



We can see how the amount in the **Server** queue changes over time.

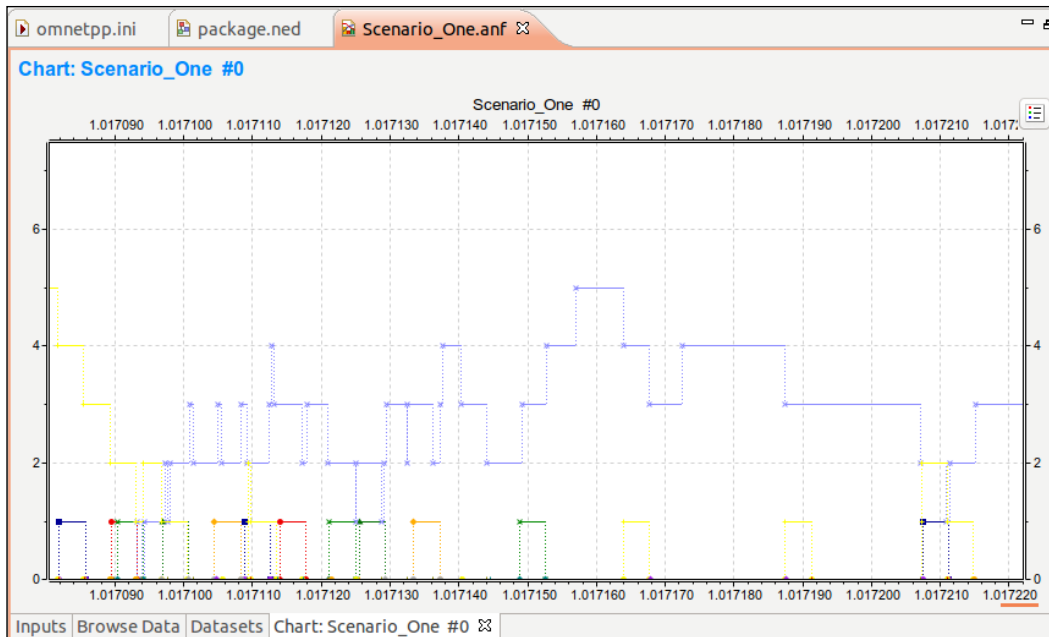
Visualizing gathered data

In this section, I will be using the data recorded from running the `Hello World` simulation. Follow the steps from the previous section to create the event log, and the vector, scalar, and analysis files.

OMNeT++ allows you to create many different visualizations from your networks. They are defined in the sections that follow.

Vector charts

This chart shows the relationship between two or more datasets. This includes the histogram which you have seen earlier. With the analysis file for the network open, open the **Vectors** tab. In the location **module filter**, click on the downward arrow of the combo box, and then click on ***.queue** to filter in only modules that use a queue. Now highlight any number of rows, right-click on the highlighted part, and now click on **Plot**. This is the output I see once I have zoomed in on a particular part of a chart:



This output shows us the various queues I selected and how they rise and fall over time.

Scalar charts

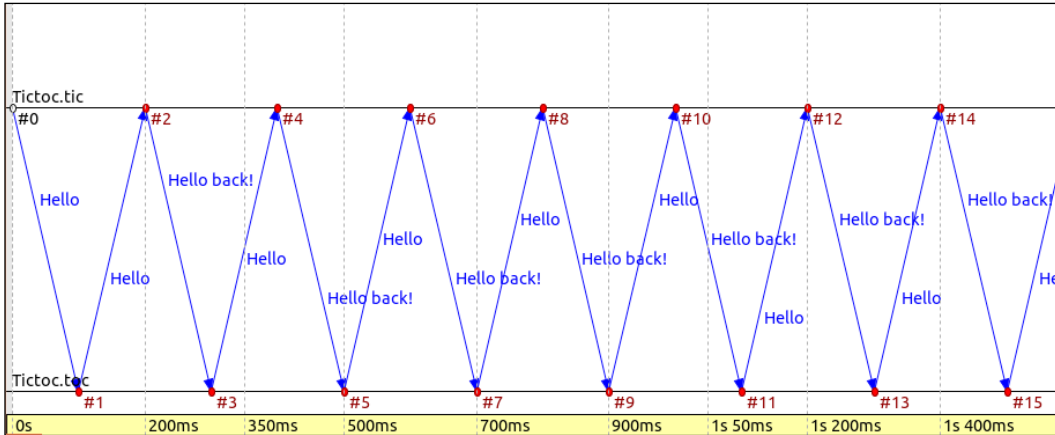
Repeat what you have done in the previous section, but select only the scalar data that you can find inside the **Scalars** tab next to the **Vectors** tab. This will produce bar charts and histograms.

Analysis of the Tictoc example project

Let's use the event log to have a detailed look into the `Tictoc` example that we created earlier. First, modify the configuration file to contain the following new line:

```
record-eventlog = true
```

Run the simulation until at least 20-30 events have occurred. Now navigate to the `Results` folder and open the `.elog` file. You will see the following screenshot:



This is a very clear overview of the simulation that was run. You can see the time that has passed for each event that has occurred at the bottom in yellow. The blue lines show the messages going back and forth between the two lines named **Tictoc.tic**, and **Tictoc.toc**. We can see that **Tictoc.toc** always sends the message **Hello** and **Tictoc.tic** always sends the message **Hello back!**. When we click on an event, for example on #7, we see the following in the **Event Log** tab:

Event #	Time	Details
#7	0.7s	Event in module (example::Txc) Tictoc.toc on arrival of message (cMessage) Hello back! from module (example::Txc) Tictoc.tic
#7	0.7s	Creating message (cMessage) Hello
#7	0.7s	Sending message (cMessage) Hello arriving at 0.8s, now + 100ms kind = 0
#7	0.7s	Sending through module (example::Txc) toc gate out propagation delay = 0.1s
#7	0.7s	Arrival at 0.8s, now + 100ms
#8	0.8s	Event in module (example::Txc) Tictoc.tic on arrival of message (cMessage) Hello from module (example::Txc) Tictoc.toc
#8	0.8s	Creating message (cMessage) Hello back!
#8	0.8s	Sending message (cMessage) Hello back! arriving at 0.9s, now + 100ms kind = 0

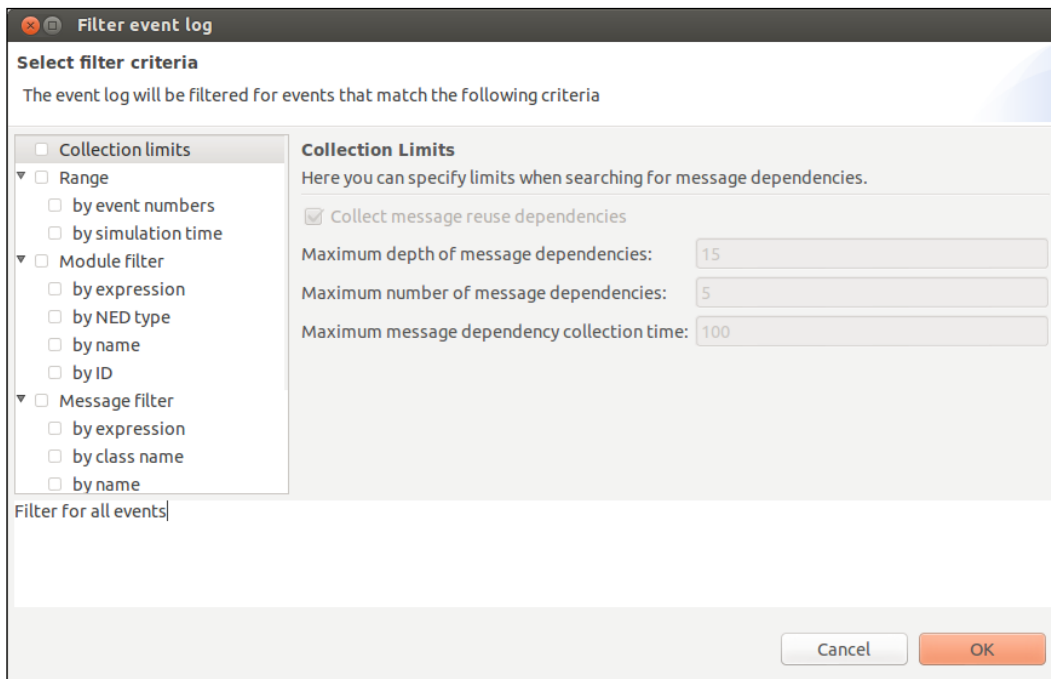
Inside the event #7, we can see a breakdown of what happens as follows:

1. **Tictoc.toc** receives a message **Hello back!** from **Tictoc.tic**.
2. **Tictoc.toc** creates a message **Hello**.
3. **Tictoc.toc** sends the message from its out gate with a delay of 0.1 seconds.
4. **Tictoc.tic** receives the message 0.1 seconds or 100 milliseconds after the message was sent.

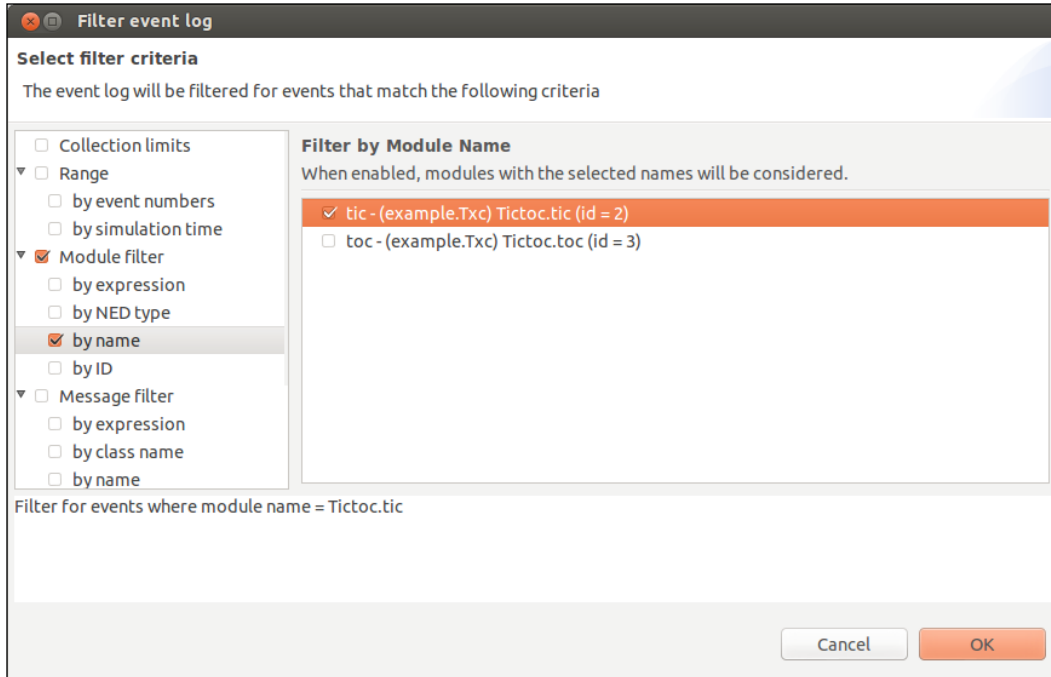
This is a very pragmatic approach to analyzing your network simulation and it gives a breakdown of each event that occurs in great detail.

Let's now filter only the module **tic** in the following manner:

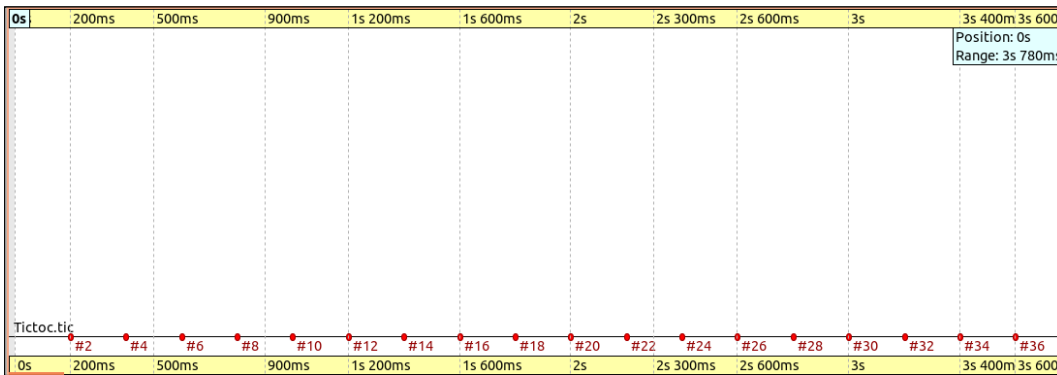
1. Right-click on the diagram view and go to **Filter | Filter event log**. You will now see the following window:



2. On the left of the window under **Module filter**, tick the **by name** checkbox. Now select which module you wish to filter as shown in the following screenshot:



3. Click on the **OK** button and now your event log will look like the following screenshot:



Only the **Tictoc.tic** events are now shown in the event log. This makes managing large amounts of data more easy as you can simply filter only what you want to see.

Generating capture packet data

OMNeT++ lets you generate PCAP data. This file format (.pcap) is created by most packet capture tools such as Wireshark. The about page on the Wireshark website (found at www.wireshark.org) says the following:

"Wireshark is the world's foremost network protocol analyzer. It lets you capture and interactively browse the traffic running on a computer network. It is the de facto (and often de jure) standard across many industries and educational institutions."

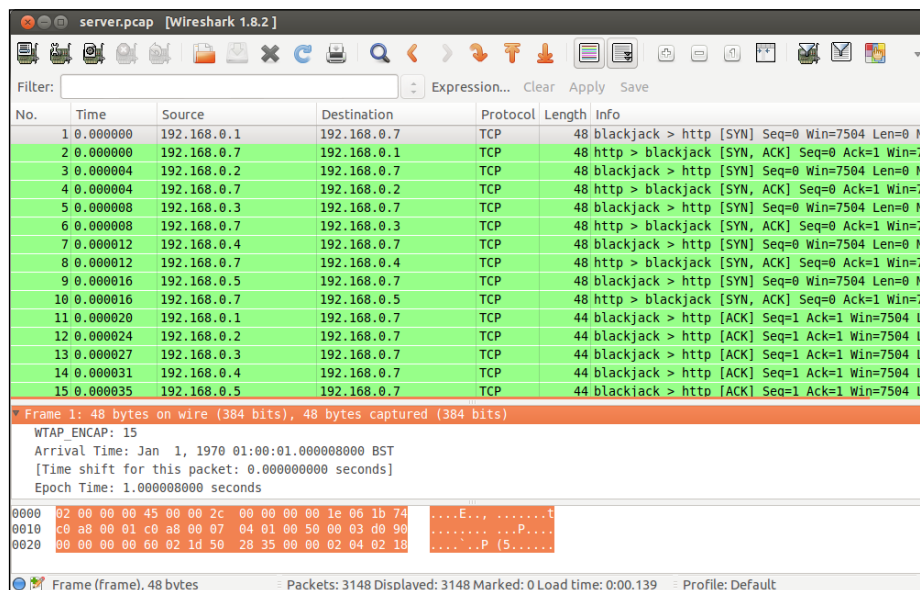
When you use PCAP files to analyze your network, it will give you the same feel as analyzing a real-life network.

To allow your `Hello World` simulation to generate PCAP files, you must modify your configuration file by adding the following code:

```
** .numPcapRecorders = 1
** .Server.pcapRecorder[0].pcapFile = "results/server.pcap"
```

The second line of the preceding code refers to my node called `Server`, which means that I want to generate only PCAP files for this node. This will give a result similar to the result got by running Wireshark in real life on that server. After running the simulation, there will now be a file called `server.pcap` in your `Results` folder. I recommend you get Wireshark as the viewer for this output. Wireshark can be downloaded from the Wireshark website at www.wireshark.org.

On opening the `server.pcap` file in Wireshark, we see the following screenshot:



You may prefer to use Wireshark to analyze your network simulation as Wireshark is an industry standard for analyzing real networks.

Summary

That is the end of this book. I hope that it has been enjoyable and that you've learned about the power of using OMNeT++ as your choice for network simulation frameworks.

Index

Symbols

@display() tags 59

A

Akaroa 14

C

capture packet data, Tictoc example project
generating 83

Channels object 27

components, OMNeT++ simulation

about 23

Channels 27

Compound Module 26

Module 24, 25

Network 23

Compound Module object 26

configuration file, OMNeT++ 31-33

D

data

gathering 73-75

visualizing 78

Define_Module() 46

E

EBitSim 34

example project

creating 48-53

example simulation, INET

running, on Linux 38-42

F

flow types, Hello World network simulation

express run 70

fast run 70

run 70

run until 70

G

Google Earth demo 35-37

H

handleMessage() method 53

Hello World network simulation

about 55

flow, controlling 69, 70

flow types 70

functioning 55

multiple scenarios, creating 67-69

network, defining 56-65

HTTP tools 34

I

INET

about 19, 34, 37

downloading 19

examples 38

importing, for preparing for next chapter
20, 21

URL 19, 37

initialize() method 53

installation

OMNeT++ 11

L

libpcap 15

Linux

- OMNeT++, compiling 16
- OMNeT++, installing 15, 17
- OMNeT++, running 18
- OMNeT++ source code, downloading 12

M

Module object 24, 25

MPI 13

N

NED

- about 27, 28
- Network component, defining 28
- network topology 28
- source code 29, 30

Network object 23, 24

O

Objective Modular Network Testbed in C++ (OMNeT++)

- about 7
- compiling, on Linux 16
- compiling, on Windows 16
- configuration file 31
- data, gathering 73-78
- downloading 11
- example project, creating 48
- Google Earth demo 35
- Hello World network simulation 55
- installing, on Linux 15
- installing, on Windows 14, 15
- pre-requisites 13
- project, creating 46
- releases 11
- running 17
- running, on Linux 18
- running, on Windows 19
- simulation frameworks 34
- source code, downloading 11
- URL 11

workspace, switching 54

OMNeT++ IDE 43-46

OMNeT++ network simulation 23

omnetpp 19

OMNeT++ simulation

- about 23
- components 23

P

PCAP 13

pre-requisites, OMNeT++

- about 13
- Akaroa 14
- MPI 13
- PCAP 13

project

- creating 46
- empty project, creating 46, 47
- importing 48

R

real network

- versus, simulated network 8

S

scalar charts 79

sendInitialMessage parameter 51

simulation

- examples 8, 9
- need for 8
- uses, for airport 9
- uses, under crowd management 9
- uses, under manufacturing 9
- uses, under weather forecasting 9

simulation frameworks

- about 34
- EBitSim 34
- HTTP Tools 34
- INET 34
- VoIPTool for INET 34

source code, OMNeT++

- downloading 11
- downloading, for Linux 12, 13
- downloading, for Windows 12

T

Tictoc 49

Tictoc example project

about 79

analysis 79-82

capture packet data, generating 83, 84

Txc module 50

V

vector charts 78

visualizations

creating 78

scalar charts 79

vector charts 78

Visual Studio Express, for Windows

URL 14

VoIPTool 34

W

Windows

OMNeT++, compiling 16

OMNeT++, installing 14-16

OMNeT++, running 19

OMNeT++ source code, downloading 12

WinPcap 15

workspace 54, 55



Thank you for buying
Learning OMNeT++

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

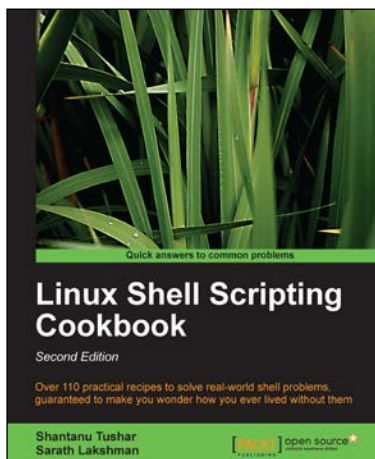
Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

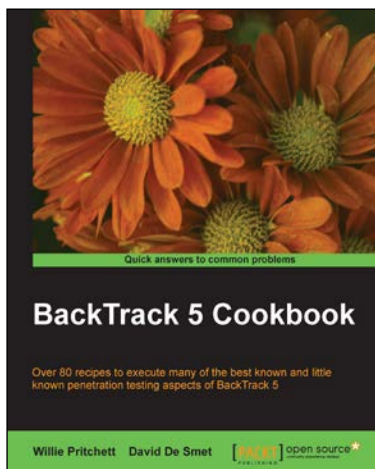


Linux Shell Scripting Cookbook, Second Edition

ISBN: 978-1-78216-274-2 Paperback: 384 pages

Over 110 practical recipes to solve real-world shell problems, guaranteed to make you wonder how you ever lived without them

1. Master the art of crafting one-liner command sequence to perform text processing, digging data from files, backups to sysadmin tools, and a lot more
2. And if powerful text processing isn't enough, see how to make your scripts interact with the web-services such as Twitter, Gmail
3. Explores the possibilities with the shell in a simple and elegant way – you will see how to effectively solve problems in your day-to-day life



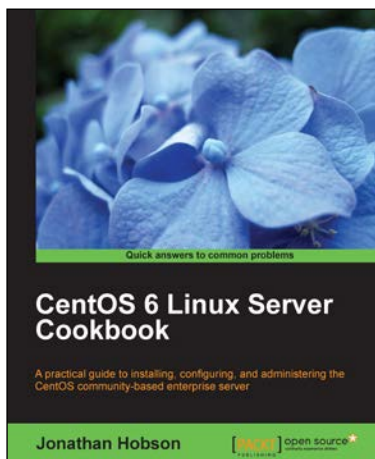
BackTrack 5 Cookbook

ISBN: 978-1-84951-738-6 Paperback: 308 pages

Over 80 recipes to execute many of the best known and little known penetration testing aspects of BackTrack 5

1. Learn to perform penetration tests with BackTrack 5
2. Nearly 100 recipes designed to teach penetration testing principles and build knowledge of BackTrack 5 Tools
3. Provides detailed step-by-step instructions on the usage of many of BackTrack's popular and not-so popular tools

Please check www.PacktPub.com for information on our titles

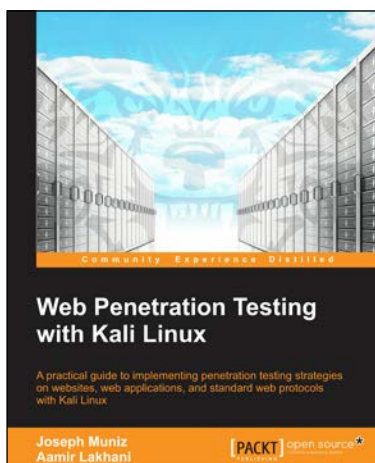


CentOS 6 Linux Server Cookbook

ISBN: 978-1-84951-902-1 Paperback: 374 pages

A practical guide to installing, configuring, and administering the CentOS community-based enterprise server

1. Delivering comprehensive insight into CentOS server with a series of starting points that show you how to build, configure, maintain, and deploy the latest edition of one of the world's most popular community based enterprise servers
2. Providing beginners and more experienced individuals alike with the opportunity to enhance their knowledge by delivering instant access to a library of recipes that addresses all aspects of CentOS server and put you in control



Web Penetration Testing with Kali Linux

ISBN: 978-1-78216-316-9 Paperback: 350 pages

A practical guide to implementing penetration testing strategies on websites, web application, and standard web protocols with Kali Linux

1. Learn key reconnaissance concepts needed as a penetration tester
2. Attack and exploit key features, authentication, and sessions on web applications
3. Learn how to protect systems, write reports, and sell web penetration testing services

Please check www.PacktPub.com for information on our titles

~StormRG~