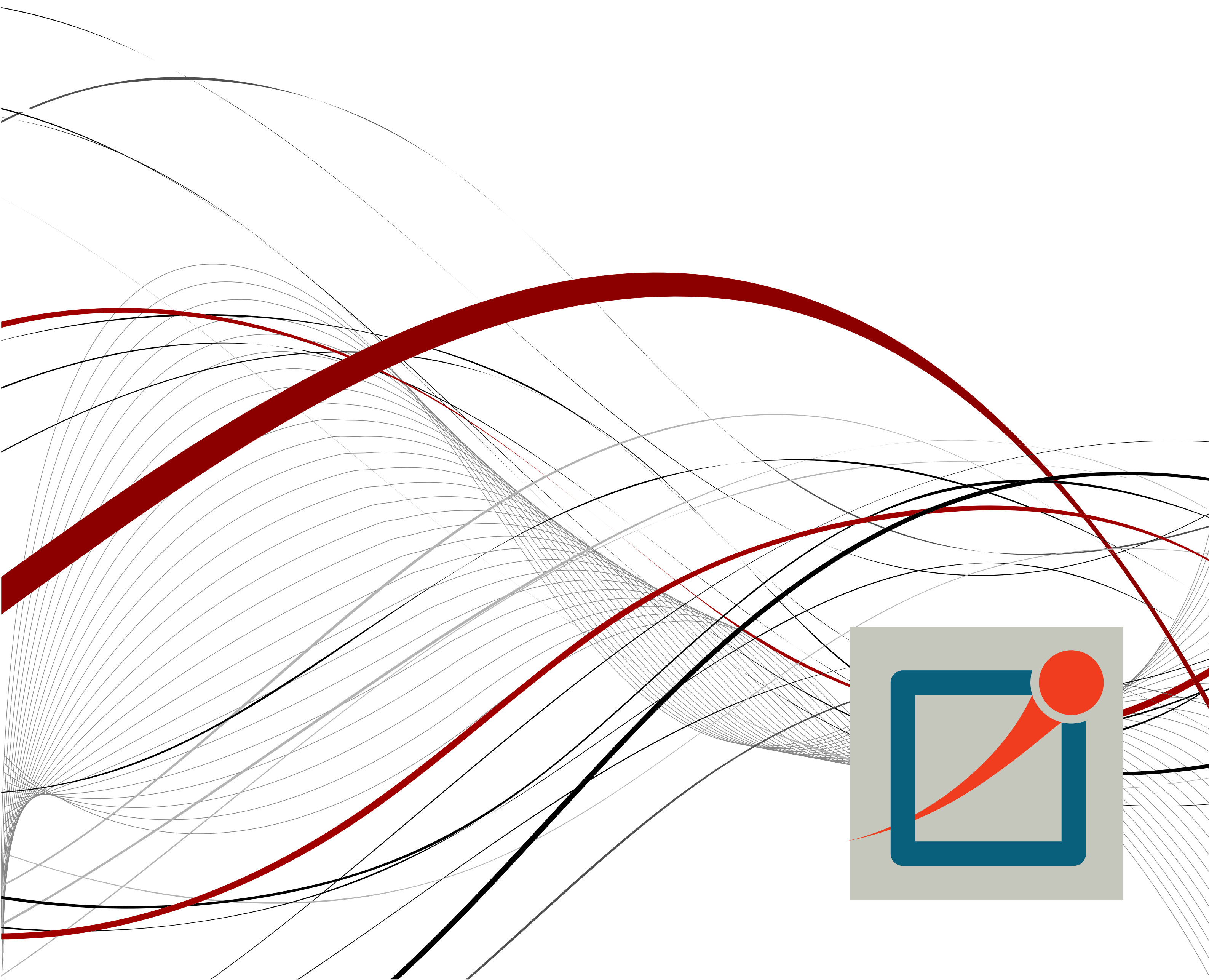
OMNeT++

Installation Guide

Version 6.0.2



Build: 231006-7d5ca33dba

CONTENTS

CHAPTER

ONE

GENERAL INFORMATION

1.1 Introduction

This document describes how to install OMNeT++ on various platforms. One chapter is dedicated to each operating system.

1.2 Supported Platforms

OMNeT++ has been tested and is supported on the following operating systems:

- Windows on x86_64 architecture
- MacOS 10.15 and 11.x on x86_64 architecture
- Linux distributions covered in this Installation Guide

The Simulation IDE is supported on the following platforms:

- Linux x86_64/aarch64
- Windows x86_64
- MacOS 10.15 and 11.x (x86_64)

Note: Simulations can be run practically on any unix-like environment with a decent and fairly up-to-date C++ compiler, for example gcc 8.x. Certain OMNeT++ features (Qtenv, parallel simulation, XML support, etc.) depend on the availability of external libraries (Qt, MPI, LibXML, etc.)

IDE platforms are restricted because the IDE relies on a native shared library, which we compile for the above platforms and distribute in binary form for convenience.

CHAPTER

TWO

WINDOWS - USING THE MINGW64 COMPILER TOOLCHAIN

2.1 Supported Windows Versions

OMNeT++ is supported only on 64-bit versions of Windows.

Note: 32-bit Windows versions are no longer supported. If you need 32-bit builds on Windows, we recommend using OMNeT++ 5.0

2.2 Installing OMNeT++

Download the OMNeT++ source code from https://omnetpp.org. Make sure you select the Windows-specific archive, named `omnetpp-6.0.2-windows-x86_64.zip`.

The package is self-contained: in addition to OMNeT++ files it includes a C++ compiler, a command-line build environment, and all libraries and programs required by OMNeT++.

Copy the OMNeT++ archive to the directory where you want to install it. Choose a directory whose full path **does not contain any space**; for example, do not put OMNeT++ under *Program Files*.

Extract the zip file. To do so, right-click the zip file in Windows Explorer, and select *Extract All* from the menu. You can also use external programs like Winzip or 7zip.

When you look into the new `omnetpp-6.0.2` directory, should see directories named `doc`, `images`, `include`, `tools`, etc., and files named `mingwenv.cmd`, `configure`, `Makefile`, and others.

2.3 Configuring and Building OMNeT++

Start `mingwenv.cmd` in the `omnetpp-6.0.2` directory by double-clicking it in Windows Explorer. It will bring up a console with the MSYS *bash* shell, where the path is already set to include the `omnetpp-6.0.2/bin` directory. On the first start of the shell, you may need to wait for the extraction of the `tools` directory.

Note: If you want to start simulations from outside the shell as well (for example from Explorer), you need to add OMNeT++'s `bin` directory and also the `bin` directories in the tools folder to the path; instructions are provided later.

First, check the contents of the `configure.user` file to make sure it contains the settings you need. In most cases you don't need to change anything.

```
notepad configure.user
```

Then enter the following commands:

```
$ ./configure
$ make
```

The build process will create both debug and release binaries.

2.4 Verifying the Installation

You should now test all samples and check they run correctly. As an example, the *aloha* example is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the graphical Qtenv environment. You should see GUI windows and dialogs.

2.5 Starting the IDE

OMNeT++ comes with an Eclipse-based Simulation IDE. You should be able to start the IDE by typing:

```
$ omnetpp
```

We recommend that you start the IDE from the command-line. You can also create a shortcut for starting the IDE. To do so, locate the `opp_ide.exe` program in the `omnetpp-6.0.2/ide` directory in Windows Explorer, right-click it, and choose *Send To > Desktop (create shortcut)* from the menu. You can right-click the taskbar icon while the IDE is running, and select *Pin this program to taskbar* from the context menu.

2.6 Environment Variables

In general OMNeT++ requires that certain environment variables are set. Always use the the provided shell window to start the IDE or your simulations.

2.7 Reconfiguring the Libraries

If you need to recompile the OMNeT++ components with different flags (e.g. different optimization), then change the top-level OMNeT++ directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make clean
$ make
```

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

Note: The built libraries and programs are immediately copied to the `lib/` and `bin/` subdirs.

2.8 Portability Issues

OMNeT++ has been tested with both the gcc and the clang compiler from the MinGW-w64 package.

Microsoft Visual C++ is not supported in the Academic Edition.

2.9 Additional Packages

2.9.1 MPI

MPI is only needed if you would like to run parallel simulations.

There are several MPI implementations for Windows, and OMNeT++ does not mandate any specific one. We recommend DeinoMPI, which can be downloaded from http://mpi.deino.net.

After installing DeinoMPI, adjust the `MPI_DIR` setting in OMNeT++'s `configure.user`, and reconfigure and recompile OMNeT++:

```
$ ./configure
$ make cleanall
$ make
```

Note: In general, if you would like to run parallel simulations, we recommend that you use Linux, macOS, or another unix-like platform.

2.9.2 Akaroa

Akaroa 2.7.9, which is the latest version at the time of writing, does not support Windows. You may try to port it using the porting guide from the Akaroa distribution.

CHAPTER

THREE

WINDOWS - USING WINDOWS SUBSYSTEM FOR LINUX (WSL) VERSION 2

WSL 2 supports running a full Linux distribution on a Windows machine. Running OMNeT++ in WSL 2 has certain advantages and disadvantages compared to running OMNeT++ natively on Windows:

Advantages:

- You will probably see significant speedup on certain tasks (like compilation) compared to the native Windows (MinGW64) toolchain, because the compiler toolchain and the filesystem (ext4) is much faster in WSL 2 than their Windows equivalents.
- The native MinGW64 toolchain on Windows is basically a mini (Unix-like) system, emulated on top of Windows. Because of the emulation, it may have incompatibilities and limitations compared to the Linux tools. You will have fewer issues and surprises when running OMNeT++ on Linux.

Disadvantages:

- WSL 2 does not (yet) support running Linux GUI applications. This means that you must install and run an X Server process on Windows to be able to use any GUI tools (i.e. IDE, Qtenv, etc.) from OMNeT++.
- Because of a limitation of the available X Server software, 3D acceleration is not working. You will not be able to use the OMNeT++ OpenSceneGraph and osgEarth integration in this setup and it is recommended to explicitly disable these features when you build OMNeT++.

3.1 WSL 2 Requirements

Installing OMNeT++ on WSL 2 is supported on Windows 10 version 1903 (build 18362.1049) or later. Note especially the minor build number. Your Windows version must have at least 1049 as a minor build number.

3.2 Enabling WSL 2 on Windows

Open a PowerShell with Administrator privileges. On newer versions of Windows, you can install the WSL subsystem by typing:

```
wsl --install
```

Alternatively, if your system does not have a `wsl` command, use the following commands:

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-
↪Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /
↪norestart
```

After a successful installation, reboot your computer and open an Administrator PowerShell again to set the default WSL version to 2.

```
wsl.exe --set-default-version 2
```

Tip: We recommend installing and using the Windows Terminal application, which is available at https://www.microsoft.com/store/productId/9N0DX20HK701

3.3 Installing an Ubuntu distribution

As a next step, you must install a Linux distribution from the Microsoft Store. We recommend using Ubuntu 20.04 from https://www.microsoft.com/store/productId/9n6svws3rx71.

Once the installation is done, run the distro and finish the setup process by setting up a user name and password. At this point, you could install OMNeT++, but GUI programs would not work.

3.4 Install VcXserver

Note: There is ongoing work (called WSLg https://github.com/microsoft/wslg) to make Linux GUI applications work on Windows by default. On later versions of Windows (21H2+) you may be able to skip the whole X Server installation step.

To use GUI programs from Linux, you must install an X Server application from: https://sourceforge.net/projects/vcxsrv/

Start the installation and make sure that you:

- select "Disable access control"
- set display number to 0
- check "Private networks, such as my home or work network" and click "Allow access" when the Windows Defender Firewall asks for permission.

Open the Windows Terminal and launch the Ubuntu distribution from the dropdown menu. Add the following line to the `/etc/bash.bashrc` or `~/.bashrc` file.

```
export DISPLAY=$(grep -m 1 nameserver /etc/resolv.conf | awk '{print $2}
↪'):0.0
```

This will ensure that Linux programs will always find the X Server process running on Windows. Exit from the Ubuntu shell, and restart it to make sure that the change was applied correctly. Check if

```
$ echo $DISPLAY
```

displays the correct IP address of the Windows machine.

In the future, make sure that the X Server is always running when you want to run Linux GUI programs by either making the X Server automatically start or launching it manually.

3.5 Install OMNeT++ Linux

At this point, you have a fully functional Linux environment that can run GUI apps. You can go on and follow the Ubuntu specific installation steps to finally install OMNeT++ on your system.

CHAPTER

FOUR

MACOS

4.1 Supported Releases

This chapter provides additional information for installing OMNeT++ on macOS.

The following releases are covered:

- macOS 11.x

4.2 Installing the Prerequisite Packages

Install the command line developer tools for macOS (compiler, debugger, etc.)

```
$ xcode-select --install
```

Installing additional packages will enable more functionality in OMNeT++; see the *Additional packages* section at the end of this chapter.

4.3 Enabling Development Mode in Terminal

MacOS has a strict default security policy preventing the execution of unsigned code. This behavior often interferes with the development process so you must explicitly allow running unsigned code from a Terminal. On the *System Preferences / Security and Privacy / Privacy* tab, select *Development Tools* on the left side, unlock the panel with the lock icon on the bottom left and select the Terminal app on the right side to override the default security policy for the Terminal app.

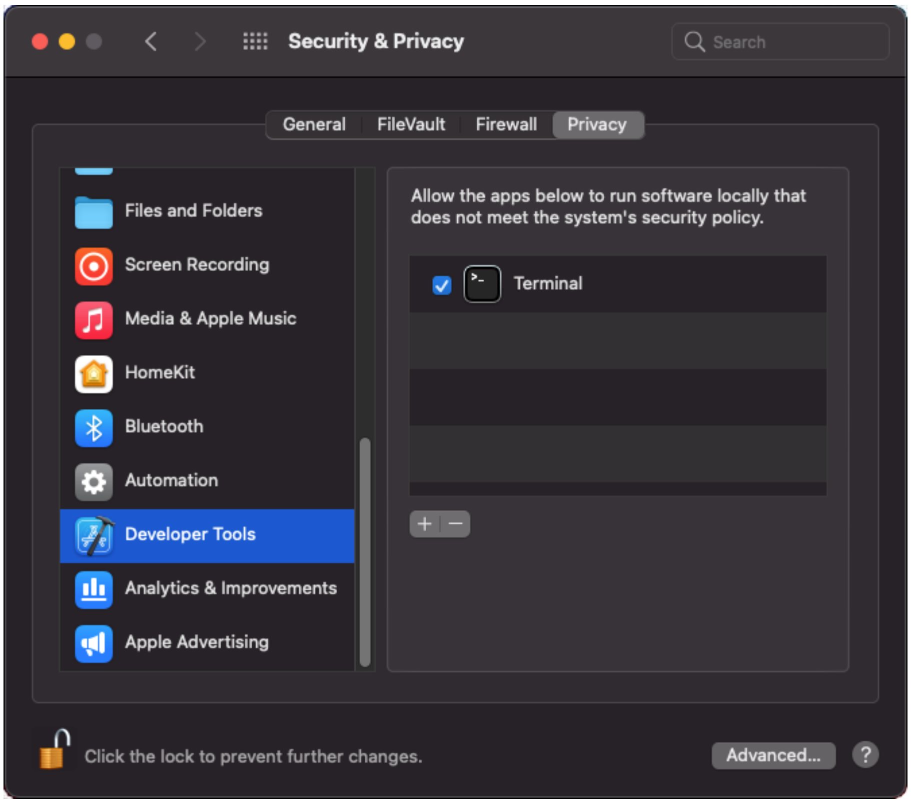


Fig. 4.1: Enable Running Unsigned Code in Terminal

Note: If you do not see the *Terminal* item in the *Development Tools* section, you should execute *spctl developer-mode enable-terminal* in the terminal and then restart *System Preferences* applet.

4.4 Debugging Unsigned Code

Even if you have enabled development mode in the terminal, missing code signatures will still cause problems during debugging, because the debugged process is started by the IDE, not the terminal. To be able to debug, you must disable code signature checking globally by typing:

```
$ sudo spctl --master-disable
```

or since Mac OS 10.15.7 (Catalina)

```
$ sudo spctl --global-disable
```

After issuing the above command go to *System Preferences / Security and Privacy / General* and select *Any* at the bottom of the dialog. After restarting your terminal application, you will be able to debug your unsigned simulation models.

4.5 Running OMNeT++ on Apple Silicon

OMNeT++ does not currently support the Apple M1 processor natively, but you can run the x86_64 version using the Rosetta 2 emulator. To run OMNeT++ under emulation, open a terminal window, then execute:

```
$ arch -x86_64 /bin/zsh --login
```

After this, follow the normal installation instructions and be sure to execute all commands in this terminal.

Note: The above command may graphically prompt you to allow the installation of the emulator component. You can also manually trigger the installation from the command line using the following command: *softwareupdate –install-rosetta –agree-to-license*.

Note: Typing *source setenv* will launch the x86_64 emulator automatically for you. Make sure to execute all commands from that terminal.

4.6 Additional Steps Required on macOS to Use the Debugger

The Command Line Developer Tools package contains the `lldb` debugger. OMNeT++ 6.0 and later contains the necessary driver binary (`lldbmi2`) that allows `lldb` to be used in the OMNeT++ IDE. If you are upgrading from an earlier version of OMNeT++, be sure to delete and recreate all Launch Configurations in the IDE. This is required because older Launch Configurations were using `gdb` as the debugger, but the new IDE uses `lldbmi2` as the debugger executable.

On the first debug session the OS may prompt you to allow debugging with the `lldb` executable.

4.7 Downloading and Unpacking OMNeT++

Download OMNeT++ from https://omnetpp.org. Make sure you select to download the macOS specific archive, `omnetpp-6.0.2-macos-x86_64.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/Users/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar zxvf omnetpp-6.0.2-macos-x86_64.tgz
```

A subdirectory called `omnetpp-6.0.2` will be created, containing the simulator files.

Alternatively, you can also unpack the archive using Finder.

Note: The Terminal can be found in the Applications / Utilities folder.

4.8 Environment Variables

In general OMNeT++ requires that certain environment variables are set and the `omnetpp-6.0.2/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnetpp-6.0.2
$ source setenv
```

To set the environment variables permanently, edit `.profile`, `.zprofile` or `.zshenv` in your home directory and add a line something like this:

```
[ -f "$HOME/omnetpp-6.0.2/setenv" ] && source "$HOME/omnetpp-6.0.2/setenv"
```

4.9 Configuring and Building OMNeT++

Check `configure.user` to make sure it contains the settings you need. In most cases you don't need to change anything in it.

In the top-level OMNeT++ directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

Note: If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (You may need to increase the scrollback buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

When `./configure` has finished, you can compile OMNeT++. Type in the terminal:

```
$ make
```

Tip: To take advantage of multiple processor cores, add the `-j4` option to the `make` command line.

Note: The build process will not write anything outside its directory, so no special privileges are needed.

Tip: The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

4.10 Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the aloha simulation is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the Qtenv environment. You should see nice gui windows and dialogs.

4.11 Starting the IDE

OMNeT++ comes with an Eclipse-based simulation IDE.

Start the IDE by typing:

```
$ omnetpp
```

If you would like to be able to launch the IDE via Applications, the Dock or a desktop shortcut, do the following: open the `omnetpp-6.0.2` folder in Finder, go into the `ide` subfolder, create an alias for the omnetpp program there (right-click, *Make Alias*), and drag the new alias into the Applications folder, onto the Dock, or onto the desktop.

Alternatively, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

which will do roughly the same.

4.12 Using the IDE

When you try to build a project in the IDE, you may get the following warning message:

> Toolchain "..." is not supported on this platform or installation. Please go to the Project menu, and activate a different build configuration. (You may need to switch to the C/C++ perspective first, so that the required menu items appear in the Project menu.)

If you encounter this message, choose *Project* > *Properties* > *C/C++ Build* > *Tool Chain Editor* > *Current toolchain* > *GCC for OMNeT++*.

The IDE is documented in detail in the *User Guide*.

4.13 Reconfiguring the Libraries

If you need to recompile the OMNeT++ components with different flags (e.g. different optimization), then change the top-level OMNeT++ directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make clean
$ make
```

Tip: To take advantage of multiple processor cores, add the `-j4` option to the `make` command line.

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

Note: The built libraries and programs are immediately copied to the `lib/` and `bin/` subdirectories.

4.14 Additional Packages

4.14.1 OpenMPI

MacOS does not come with OpenMPI, so you must install it manually. You can install it from the Homebrew repo (http://brew.sh) by typing `brew install open-mpi`. In this case, you have to manually set the MPI_CFLAGS and MPI_LIBS variables in `configure.user` and re-run `./configure`.

4.14.2 Akaroa

Akaroa 2.7.9, which is the latest version at the time of writing, does not support macOS. You may try to port it using the porting guide from the Akaroa distribution.

CHAPTER

FIVE

LINUX

5.1 Supported Linux Distributions

This chapter provides instructions for installing OMNeT++ on selected Linux distributions:

- Ubuntu 20.04 and 22.4 LTS
- Fedora Workstation 31
- Red Hat Enterprise Linux Desktop Workstation 8.x
- OpenSUSE Leap 15.3

This chapter describes the overall process. Distro-specific information, such as how to install the prerequisite packages, are covered by distro-specific chapters.

Note: If your Linux distribution is not listed above, you still may be able to use some distro-specific instructions in this Guide.

Ubuntu derivatives (Ubuntu instructions may apply):

- Kubuntu, Xubuntu, Edubuntu, . . .
- Linux Mint

Some Debian-based distros (Ubuntu instructions may apply, as Ubuntu itself is based on Debian):

- Knoppix and derivatives
- Mepis

Some Fedora-based distros (Fedora instructions may apply):

- Simplis
- Eeedora

5.2 Installing the Prerequisite Packages

OMNeT++ requires several packages to be installed on the computer. These packages include the C++ compiler (gcc or clang) and several other libraries and programs. These packages can be installed from the software repositories of your Linux distribution.

See the chapter specific to your Linux distribution for instructions on installing the packages needed by OMNeT++.

Generally, you will need superuser permissions to install packages.

Not all packages are available from software repositories; some (optional) ones need to be downloaded separately from their web sites, and installed manually. See the section *Additional Packages* later in this chapter.

5.3 Downloading and Unpacking

Download OMNeT++ from https://omnetpp.org. Make sure you select to download the Linux specific archive, `omnetpp-6.0.2-linux-x86_64.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/home/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar xvfz omnetpp-6.0.2-linux-x86_64.tgz
```

This will create an `omnetpp-6.0.2` subdirectory with the OMNeT++ files in it.

Note: On how to open a terminal on your Linux installation, see the chapter specific to your Linux distribution.

5.4 Environment Variables

In general OMNeT++ requires that certain environment variables are set and the *omnetpp-6.0.2/bin* directory is in the PATH. Source the *setenv* script to set up all these variables.

```
$ cd omnetpp-6.0.2
$ source setenv
```

To set the environment variables permanently, edit `.profile` or `.zprofile` in your home directory and add a line something like this:

```
[ -f "$HOME/omnetpp-6.0.2/setenv" ] && source "$HOME/omnetpp-6.0.2/setenv"
```

Note: If you use a shell other than bash, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

Note: If you use a shell other than *bash*, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

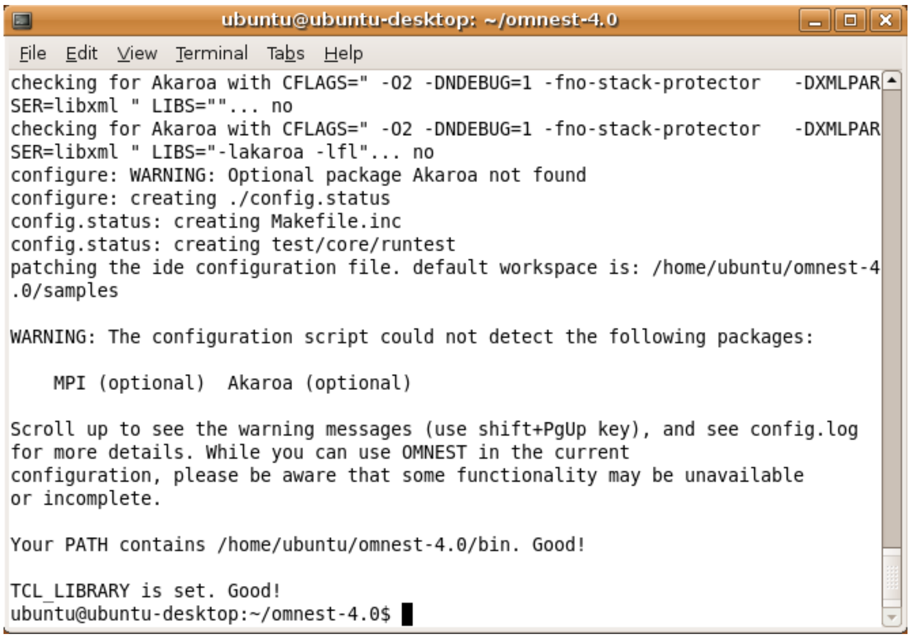
Note that all Linux distributions covered in this Installation Guide use *bash* unless the user has explicitly selected another shell.

5.5 Configuring and Building OMNeT++

In the top-level OMNeT++ directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.


ubuntu@ubuntu-desktop: ~/omnest-4.0
File Edit View Terminal Tabs Help

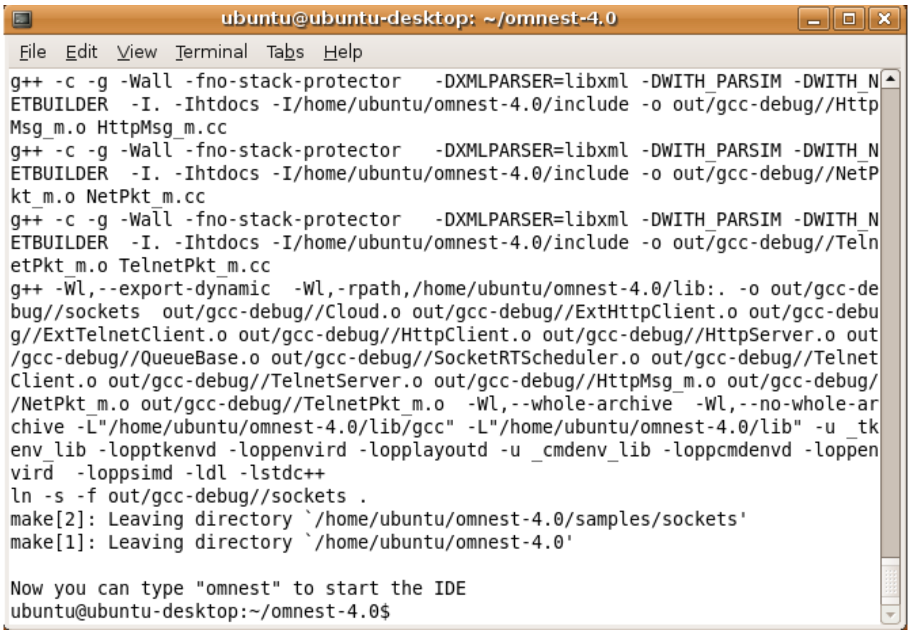
```
checking for Akaroa with CFLAGS=" -O2 -DNDEBUG=1 -fno-stack-protector   -DXMLPAR
SER=libxml " LIBS=""... no
checking for Akaroa with CFLAGS=" -O2 -DNDEBUG=1 -fno-stack-protector   -DXMLPAR
SER=libxml " LIBS="-lakaroa -lfl"... no
configure: WARNING: Optional package Akaroa not found
configure: creating ./config.status
config.status: creating Makefile.inc
config.status: creating test/core/runtest
patching the ide configuration file. default workspace is: /home/ubuntu/omnest-4
.0/samples

WARNING: The configuration script could not detect the following packages:

    MPI (optional)  Akaroa (optional)

Scroll up to see the warning messages (use shift+PgUp key), and see config.log
for more details. While you can use OMNEST in the current
configuration, please be aware that some functionality may be unavailable
or incomplete.

Your PATH contains /home/ubuntu/omnest-4.0/bin. Good!

TCL_LIBRARY is set. Good!
ubuntu@ubuntu-desktop:~/omnest-4.0$
```

Fig. 5.1: Configuring OMNeT++

Note: If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (Use Shift+PgUp; you may need to increase the scroll-back buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

When `./configure` has finished, you can compile OMNeT++. Type in the terminal:

```
$ make
```


ubuntu@ubuntu-desktop: ~/omnest-4.0
File Edit View Terminal Tabs Help

```
g++ -c -g -Wall -fno-stack-protector   -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_N
ETBUILDER  -I. -Ihtdocs -I/home/ubuntu/omnest-4.0/include -o out/gcc-debug//Http
Msg_m.o HttpMsg_m.cc
g++ -c -g -Wall -fno-stack-protector   -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_N
ETBUILDER  -I. -Ihtdocs -I/home/ubuntu/omnest-4.0/include -o out/gcc-debug//NetP
kt_m.o NetPkt_m.cc
g++ -c -g -Wall -fno-stack-protector   -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_N
ETBUILDER  -I. -Ihtdocs -I/home/ubuntu/omnest-4.0/include -o out/gcc-debug//Teln
etPkt_m.o TelnetPkt_m.cc
g++ -Wl,--export-dynamic  -Wl,-rpath,/home/ubuntu/omnest-4.0/lib:. -o out/gcc-de
bug//sockets  out/gcc-debug//Cloud.o out/gcc-debug//ExtHttpClient.o out/gcc-debu
g//ExtTelnetClient.o out/gcc-debug//HttpClient.o out/gcc-debug//HttpServer.o out
/gcc-debug//QueueBase.o out/gcc-debug//SocketRTScheduler.o out/gcc-debug//Telnet
Client.o out/gcc-debug//TelnetServer.o out/gcc-debug//HttpMsg_m.o out/gcc-debug/
/NetPkt_m.o out/gcc-debug//TelnetPkt_m.o  -Wl,--whole-archive  -Wl,--no-whole-ar
chive -L"/home/ubuntu/omnest-4.0/lib/gcc" -L"/home/ubuntu/omnest-4.0/lib" -u _tk
env_lib -lopptkenvd -loppenvird -lopplayoutd -u _cmdenv_lib -loppcmdenvd -loppen
vird  -loppsimd -ldl -lstdc++
ln -s -f out/gcc-debug//sockets .
make[2]: Leaving directory `/home/ubuntu/omnest-4.0/samples/sockets'
make[1]: Leaving directory `/home/ubuntu/omnest-4.0'

Now you can type "omnest" to start the IDE
ubuntu@ubuntu-desktop:~/omnest-4.0$
```

Fig. 5.2: Building OMNeT++

Tip: To take advantage of multiple processor cores, add the `-j8` option to the `make` command line.

Note: The build process will not write anything outside its directory, so no special privileges are needed.

Tip: The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

5.6 Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the aloha simulation is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the Qtenv environment. You should see nice gui windows and dialogs.

5.7 Starting the IDE

You can launch the OMNeT++ Simulation IDE by typing the following command in the terminal:

```
$ omnetpp
```

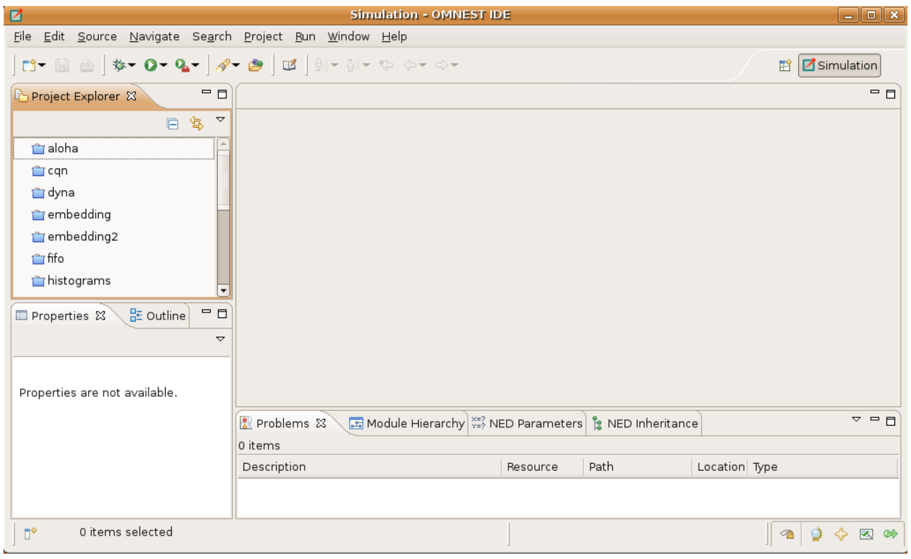


Fig. 5.3: The Simulation IDE

If you would like to be able to access the IDE from the application launcher or via a desktop shortcut, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

Or add a shortcut that points to the `omnetpp` program in the `ide` subdirectory by other means, for example using the Linux desktop's context menu.

5.8 Using the IDE

When you try to build a project in the IDE, you may get the following warning message:

> Toolchain "..." is not supported on this platform or installation. Please go to the Project menu, and activate a different build configuration. (You may need to switch to the C/C++ perspective first, so that the required menu items appear in the Project menu.)

If you encounter this message, choose *Project* > *Properties* > *C/C++ Build* > *Tool Chain Editor* > *Current toolchain* > *GCC for OMNeT++*.

The IDE is documented in detail in the *User Guide*.

5.9 Reconfiguring the Libraries

If you need to recompile the OMNeT++ components with different flags (e.g. different optimization), then change the top-level OMNeT++ directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make cleanall
$ make
```

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

Note: For detailed description of all options please read the *Build Options* chapter.

5.10 Additional Packages

Note that at this point, MPI, Doxygen and GraphViz have been installed as part of the prerequisites.

5.10.1 Qtenv

OMNeT++ comes with a Qt based runtime environment that supports also 3D visualization. The new environment can be disabled by the WITH_QTENV=no variable in the configure.user file and then running `./configure`.

5.10.2 Akaroa

Linux distributions do not contain the Akaroa package. It must be downloaded, compiled and installed manually before installing OMNeT++.

Note: As of version 2.7.9, Akaroa only supports Linux and Solaris.

Download Akaroa 2.7.9 from: http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.chtml

Extract it into a temporary directory:

```
$ tar xfz akaroa-2.7.9.tar.gz
```

Configure, build and install the Akaroa library. By default, it will be installed into the `/usr/local/akaroa` directory.

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

Go to the OMNeT++ directory, and (re-)run the `configure` script. Akaroa will be automatically detected if you installed it to the default location.

5.10.3 Nemiver

Nemiver is the default debugger for the OMNeT++ just-in-time debugging facility (see the `debugger-attach-on-startup` and `debugger-attach-on-error` configuration options). Nemiver can be installed via the package manager in most Linux distros. For example, on Ubuntu and other Debian-based distros you can install it by the following command:

```
$ sudo apt-get install nemiver
```

CHAPTER

SIX

UBUNTU

6.1 Supported Releases

This chapter provides additional information for installing OMNeT++ on Ubuntu Linux installations. The overall installation procedure is described in the *Linux* chapter.

The following Ubuntu releases are covered:

- Ubuntu 20.04 LTS or 22.04 LTS

They were tested on the following architectures:

- Intel/AMD 64-bit

The instructions below assume that you use the default desktop and the bash shell. If you use another desktop environment or shell, you may need to adjust the instructions accordingly.

6.2 Opening a Terminal

Type *terminal* in your program launcher and click on the Terminal icon.

6.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

6.3.1 Command-Line Installation

Before starting the installation, refresh the database of available packages. Type in the terminal:

```
$ sudo apt-get update
```

To install the required packages, type in the terminal:

```
$ sudo apt-get install build-essential clang lld gdb bison flex perl \
    python3 python3-pip qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools \
    libqt5opengl5-dev libxml2-dev zlib1g-dev doxygen graphviz \
    libwebkit2gtk-4.0-37 xdg-utils
$ python3 -m pip install --user --upgrade numpy pandas matplotlib scipy \
    seaborn posix_ipc
```

To use Qtenv with 3D visualization support, install the development packages for OpenSceneGraph (3.4 or later) and the osgEarth (2.9 or later) packages. (You may need to enable the *Universe* software repository in Software Sources. and also enable *WITH_OSGEARTH* in *configure.user*.)

```
$ sudo apt-get install openscenegraph-plugin-osgearth libosgearth-dev
```

Warning: Ubuntu 22.04 no longer provides the *libosgearth* package so osgEarth must be installed from sources. OpenSceneGraph can still be installed using *sudo apt-get install libopenscenegraph-dev*.

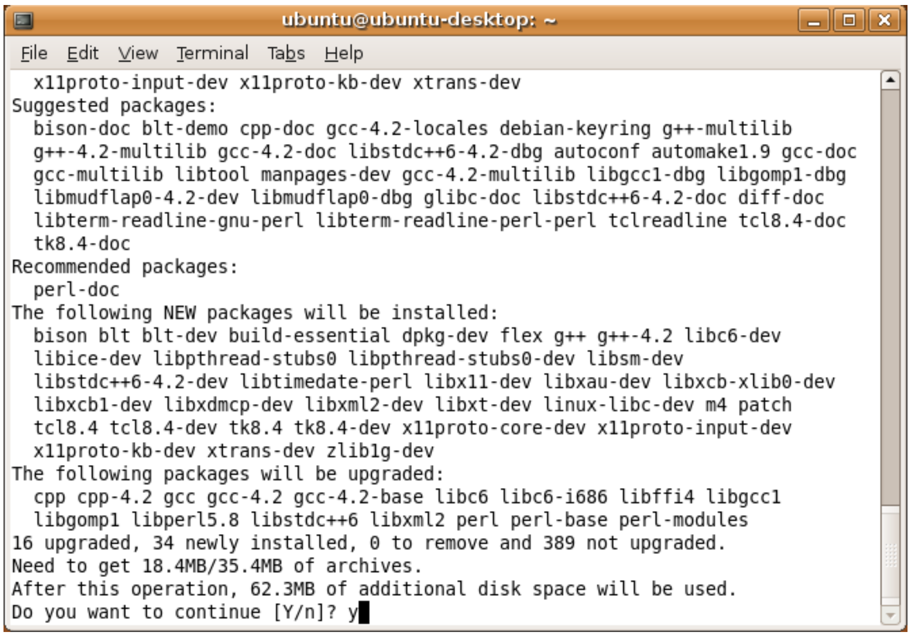
Warning: The IDE requires GLIBC 2.28 version or later, so you Ubuntu 18.04 is NOT supported because it comes with GLIBC 2.27.

Note: You may opt to use gcc instead of the clang compiler and/or use the system default linker instead of *lld* by setting the `PREFER_CLANG` and `PREFER_LLD` variables in the *configure.user* file. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

To enable the optional parallel simulation support you will need to install the MPI packages:

```
$ sudo apt-get install mpi-default-dev
```

At the confirmation questions (*Do you want to continue? [Y/N]*), answer *Y*.



```
  x11proto-input-dev x11proto-kb-dev xtrans-dev
Suggested packages:
  bison-doc blt-demo cpp-doc gcc-4.2-locales debian-keyring g++-multilib
  g++-4.2-multilib gcc-4.2-doc libstdc++6-4.2-dbg autoconf automake1.9 gcc-doc
  gcc-multilib libtool manpages-dev gcc-4.2-multilib libgcc1-dbg libgomp1-dbg
  libmudflap0-4.2-dev libmudflap0-dbg glibc-doc libstdc++6-4.2-doc diff-doc
  libterm-readline-gnu-perl libterm-readline-perl-perl tclreadline tcl8.4-doc
  tk8.4-doc
Recommended packages:
  perl-doc
The following NEW packages will be installed:
  bison blt blt-dev build-essential dpkg-dev flex g++ g++-4.2 libc6-dev
  libice-dev libpthread-stubs0 libpthread-stubs0-dev libsm-dev
  libstdc++6-4.2-dev libtimedate-perl libx11-dev libxau-dev libxcb-xlib0-dev
  libxcb1-dev libxdmcp-dev libxml2-dev libxt-dev linux-libc-dev m4 patch
  tcl8.4 tcl8.4-dev tk8.4 tk8.4-dev x11proto-core-dev x11proto-input-dev
  x11proto-kb-dev xtrans-dev zlib1g-dev
The following packages will be upgraded:
  cpp cpp-4.2 gcc gcc-4.2 gcc-4.2-base libc6 libc6-i686 libffi4 libgcc1
  libgomp1 libperl5.8 libstdc++6 libxml2 perl perl-base perl-modules
16 upgraded, 34 newly installed, 0 to remove and 389 not upgraded.
Need to get 18.4MB/35.4MB of archives.
After this operation, 62.3MB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

Fig. 6.1: Command-Line Package Installation

6.3.2 Graphical Installation

Open the dash and type *Synaptic*.

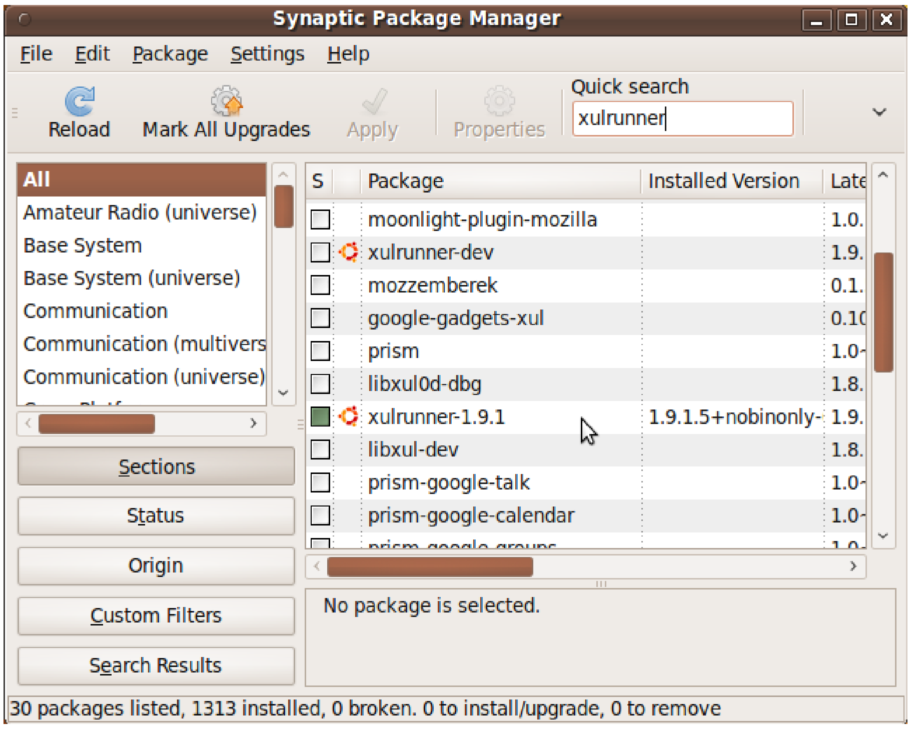
Since software installation requires root permissions, Synaptic will ask you to type your password.

Search for the following packages in the list, click the squares before the names, then choose *Mark for installation* or *Mark for upgrade*.

If the *Mark additional required changes?* dialog comes up, choose the *Mark* button.

The packages:

- required: build-essential, gcc, g++, bison, flex, perl, qtbase5-dev, qtchooser, qt5-qmake, qtbase5-dev-tools, python3, doxygen, graphviz, libwebkit2gtk-4.0-37, xdg-utils
- recommended: libopenscenegraph-dev, openscenegraph-plugin-osgearth, libosgearth-dev, mpi-default-dev, libxml2-dev, zlib1g-dev



Quick search: xulrunner

S	Package	Installed Version	Late
☐	moonlight-plugin-mozilla		1.0.
☐	xulrunner-dev		1.9.
☐	mozzemberek		0.1.
☐	google-gadgets-xul		0.10
☐	prism		1.0-
☐	libxul0d-dbg		1.8.
🟩	xulrunner-1.9.1	1.9.1.5+nobinonly-	1.9.
☐	libxul-dev		1.8.
☐	prism-google-talk		1.0-
☐	prism-google-calendar		1.0-

No package is selected.

30 packages listed, 1313 installed, 0 broken. 0 to install/upgrade, 0 to remove

Fig. 6.2: Synaptic Package Manager

Click *Apply*, then in the *Apply the following changes?* window, click *Apply* again. In the *Changes applied* window, click *Close*.

After this, you still have to install some required Python packages from command line:

```
$ python3 -m pip install --user --upgrade numpy pandas matplotlib scipy \
    seaborn posix_ipc
```

6.3.3 Post-Installation Steps

Setting Up Debugging

By default, Ubuntu does not allow ptracing of non-child processes by non-root users. That is, if you want to be able to debug simulation processes by attaching to them with a debugger, or similar, you want to be able to use OMNeT++ just-in-time debugging (`debugger-attach-on-startup` and `debugger-attach-on-error` configuration options), you need to explicitly enable them.

To temporarily allow ptracing non-child processes, enter the following command:

```
$ echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope
```

To permanently allow it, edit `/etc/sysctl.d/10-ptrace.conf` and change the line:

```
kernel.yama.ptrace_scope = 1
```

to read

```
kernel.yama.ptrace_scope = 0
```

CHAPTER

SEVEN

FEDORA 33

7.1 Supported Releases

This chapter provides additional information for installing OMNeT++ on Fedora installations. The overall installation procedure is described in the *Linux* chapter.

The following Fedora release is covered:

- Fedora 33

It was tested on the following architectures:

- Intel 64-bit

7.2 Opening a Terminal

Open the Search bar, and type *Terminal*.

7.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

7.3.1 Command-Line Installation

To install the required packages, type in the terminal:

```
$ sudo dnf install make gcc gcc-c++ clang lld bison flex perl \
    python3 python3-pip qt5-devel libxml2-devel \
    zlib-devel doxygen graphviz webkitgtk
$ python3 -m pip install --user --upgrade numpy pandas \
    matplotlib scipy seaborn posix_ipc
```

To use 3D visualization support in Qtenv, you should install OpenSceneGraph 3.2 or later and osgEarth 2.7 or later (recommended):

```
$ sudo dnf install OpenSceneGraph-devel osgearth-devel
```

Note: You may opt to use gcc instead of the clang compiler and/or use the system default linker instead of *lld* by setting the `PREFER_CLANG` and `PREFER_LLD` variables in the *configure.user* file. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

To enable the optional parallel simulation support you will need to install the MPI package:

```
$ sudo dnf install openmpi-devel
```

Note that *openmpi* will not be available by default, it needs to be activated in every session with the

```
$ module load mpi/openmpi-x86_64
```

command. When in doubt, use `module avail` to display the list of available modules. If you need MPI in every session, you may add the `module load` command to your startup script (`.bashrc`).

7.3.2 Graphical Installation

The graphical package manager can be launched by opening the Search bar and typing *dnf*.

Search for the following packages in the list. Select the checkboxes in front of the names, and pick the latest version of each package.

The packages:

- make, bison, gcc, gcc-c++, clang, lld, flex, perl, python3, python3-pip, qt5-devel, libxml2-devel, zlib-devel, webkitgtk, doxygen, graphviz, openmpi-devel, OpenSceneGraph-devel, osgearth-devel

Click *Apply*, then follow the instructions.

Open a terminal and enter the following command to install the required Python packages:

```
$ python3 -m pip install --user --upgrade numpy pandas matplotlib scipy \
    seaborn posix_ipc
```

CHAPTER

EIGHT

RED HAT

8.1 Supported Releases

This chapter provides additional information for installing OMNeT++ on Red Hat Enterprise Linux installations. The overall installation procedure is described in the *Linux* chapter.

The following Red Hat release is covered:

- Red Hat Enterprise Linux Desktop Workstation 8.x

It was tested on the following architectures:

- Intel 64-bit

8.2 Opening a Terminal

Choose *Applications* > *Accessories* > *Terminal* from the menu.

8.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

Note: You will need Red Hat Enterprise Linux Desktop Workstation for OMNeT++. The *Desktop Client* version does not contain development tools.

8.3.1 Command-Line Installation

To install the required packages, type in the terminal:

```
$ su -c 'yum install make gcc gcc-c++ clang lld bison flex perl \
    python3 python3-pip qt-devel libxml2-devel zlib-devel doxygen \
    graphviz xdg-utils'
$ python3 -m pip install --user --upgrade numpy pandas matplotlib scipy \
    seaborn posix_ipc
```

To use 3D visualization support in Qtenv (recommended), you should install the OpenSceneGraph-devel (3.2 or later) and osgEarth-devel (2.7 or later) packages. These packages are not available from the official RedHat repository so you may need to get them from different sources (e.g. rpmfind.net).

Warning: The IDE requires GLIBC 2.28 version or later, so RedHat 6 and 7 is NOT supported.

Note: You may opt to use gcc instead of the clang compiler and/or use the system default linker instead of *lld* by setting the `PREFER_CLANG` and `PREFER_LLD` variables in the *configure.user* file. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

To install additional (optional) packages for parallel simulation, type:

```
$ su -c 'yum install openmpi-devel'
```

Note that *openmpi* will not be available by default, it needs to be activated in every session with the

```
$ module load openmpi_<arch>
```

command, where `<arch>` is your architecture (usually `i386` or `x86_64`). When in doubt, use `module avail` to display the list of available modules. If you need MPI in every session, you may add the `module load` command to your startup script (`.bashrc`).`

8.3.2 Graphical Installation

The graphical installer can be launched by choosing *Applications > Add/Remove Software* from the menu.

Search for the following packages in the list. Select the checkboxes in front of the names, and pick the latest version of each package.

The packages:

- make, gcc, gcc-c++, clang, lld, bison, flex, perl, python3, qt-devel, libxml2-devel, zlib-devel, doxygen, graphviz, openmpi-devel, xdg-utils

Click *Apply*, then follow the instructions.

8.4 SELinux

You may need to turn off SELinux when running certain simulations. To do so, click on *System > Administration > Security Level > Firewall*, go to the *SELinux* tab, and choose *Disabled.*

You can verify the SELinux status by typing the `sestatus` command in a terminal.

Note: From OMNeT++ 4.1 on, makefiles that build shared libraries include the `chcon -t textrel_shlib_t lib<name>.so` command that properly sets the security context for the library. This should prevent the SELinux-related *"cannot restore segment prot after reloc: Permission denied"* error from occurring, unless you have a shared library which was built using an obsolete or hand-crafted makefile that does not contain the `chcon` command.

CHAPTER

NINE

OPENSUSE

9.1 Supported Releases

This chapter provides additional information for installing OMNeT++ on openSUSE installations. The overall installation procedure is described in the *Linux* chapter.

The following openSUSE release is covered:

- openSUSE Leap 15.3

It was tested on the following architectures:

- Intel 64-bit

9.2 Opening a Terminal

Open the Search bar, and type *Terminal*.

9.3 Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

9.3.1 Command-Line Installation

To install the required packages, type in the terminal:

```
$ sudo zypper install make gcc gcc-c++ clang lld bison flex perl \
    python3 python3-pip libqt5-qtbase-devel libxml2-devel zlib-devel \
    doxygen graphviz xdg-utils
$ python3 -m pip install --user --upgrade numpy pandas matplotlib scipy \
    seaborn posix_ipc
```

Note: You may opt to use gcc instead of the clang compiler and/or use the system default linker instead of *lld* by setting the `PREFER_CLANG` and `PREFER_LLD` variables in the *configure.user* file. In this case, you don't have to install the `clang` and `lld` packages. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

To use 3D visualization support in Qtenv (recommended), you should install the OpenSceneGraph-devel (3.2 or later) and osgEarth-devel (2.7 or later) packages. These packages are not available from the official RedHat repository so you may need to get them from different sources (e.g. rpmfind.net).

To enable the optional parallel simulation support you will need to install the MPI package:

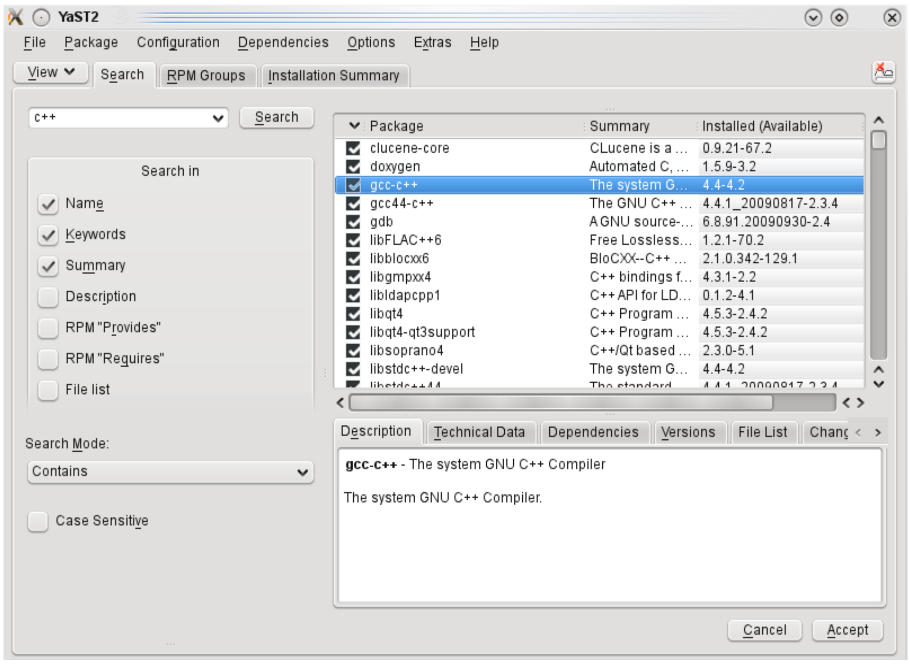
```
$ sudo zypper install openmpi-devel
```

Note that *openmpi* will not be available by default, first you need to log out and log in again, or source your `.profile` script:

```
$ . ~/.profile
```

9.3.2 Graphical Installation

The graphical installer can be launched by opening the Search bar and typing *Software Management*.


YaST2
File Package Configuration Dependencies Options Extras Help
View Search RPM Groups Installation Summary
c++
Search
Search in
Name
Keywords
Summary
Description
RPM "Provides"
RPM "Requires"
File list
Search Mode:
Contains
Case Sensitive
Description Technical Data Dependencies Versions File List Chang
Cancel
Accept

Package	Summary	Installed (Available)
✓ clucene-core	CLucene is a ...	0.9.21-67.2
✓ doxygen	Automated C, ...	1.5.9-3.2
✓ gcc-c++	The system G...	4.4-4.2
✓ gcc44-c++	The GNU C++ ...	4.4.1_20090817-2.3.4
✓ gdb	A GNU source-...	6.8.91.20090930-2.4
✓ libFLAC++6	Free Lossless...	1.2.1-70.2
✓ libblocxx6	BloCXX--C++ ...	2.1.0.342-129.1
✓ libgmpxx4	C++ bindings f...	4.3.1-2.2
✓ libldapcpp1	C++ API for LD...	0.1.2-4.1
✓ libqt4	C++ Program ...	4.5.3-2.4.2
✓ libqt4-qt3support	C++ Program ...	4.5.3-2.4.2
✓ libsoprano4	C++/Qt based ...	2.3.0-5.1
✓ libstdc++-devel	The system G...	4.4-4.2

gcc-c++ - The system GNU C++ Compiler

The system GNU C++ Compiler.

Fig. 9.1: Yast Software Management

Search for the following packages in the list. Select the checkboxes in front of the names, and pick the latest version of each package.

The packages:

- make, gcc, gcc-c++, clang, lld, bison, flex, perl, libqt5-qtbase-devel, libxml2-devel, zlib-devel, xdg-utils, doxygen, graphviz, openmpi-devel

Click *Accept*, then follow the instructions.

CHAPTER

TEN

GENERIC UNIX

10.1 Introduction

This chapter provides additional information for installing OMNeT++ on Unix-like operating systems not specifically covered by this Installation Guide. The list includes FreeBSD, Solaris, and Linux distributions not covered in other chapters.

Note: In addition to Windows and macOS, the Simulation IDE will only work on Linux x86/arm 64-bit platforms. Other operating systems (FreeBSD, Solaris, etc.) and architectures may still be used as simulation platforms, without the IDE.

10.2 Dependencies

The following packages are required for OMNeT++ to work:

build-essential, GNU make, gcc, g++, bison (3.0+), flex, perl, python3, xdg-utils
These packages are needed for compiling OMNeT++ and simulation models, and also for certain OMNeT++ tools to work.

It is also recommended to install the *clang* and *lld* package as they provide faster compilation and linking.

Note: You may opt to use gcc instead of the clang compiler and/or use the system default linker instead of *lld* by setting the `PREFER_CLANG` and `PREFER_LLD` variables in the *configure.user* file. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file, too.

Warning: The IDE requires GLIBC 2.28 version or later, so you will need at least Debian 10, RedHat 8 or Ubuntu 18.10 to run the IDE.

The following packages are strongly recommended, because their absence results in severe feature loss:

Qt 5.9 or later
Required by the Qtenv simulation runtime environment. You need the *devel* packages that include header files as well.

OpenSceneGraph (3.4+) and osgEarth (2.9+)
These packages will enable 3D visualization in Qtenv. You need the *devel* packages that include header files as well.

The following packages are required if you want to take advantage of some advanced OMNeT++ features:

LibXML2
LibXML2 is needed for OMNeT++ to be able to DTD validate an XML file. The *devel* packages (that include the header files) are needed.

GraphViz, Doxygen
These packages are used by the NED documentation generation feature of the IDE. When they are missing, documentation will have less content.

MPI
openmpi or some other MPI implementation is required to support parallel simulation execution.

Akaroa
Implements Multiple Replications In Parallel (MRIP). Akaroa can be downloaded from the project's website.

The exact names of these packages may differ across distributions.

10.3 Determining Package Names

If you have a distro unrelated to the ones covered in this Installation Guide, you need to figure out what is the established way of installing packages on your system, and what are the names of the packages you need.

10.3.1 Qt

If your platform does not have suitable Qt packages, you may still use OMNeT++ to run simulations from the command line. To disable the Qtenv runtime environment, use:

```
$ ./configure WITH_QTENV=no
```

This will prevent the build system to link with Qt libraries. It is also recommended if you are installing OMNeT++ from a remote terminal session.

10.3.2 MPI

OMNeT++ is not sensitive to the particular MPI implementation. You may use OpenMPI, or any other standards-compliant MPI package.

10.4 Downloading and Unpacking

Download OMNeT++ from https://omnetpp.org. Make sure you select to download the generic archive, `omnetpp-6.0.2-core.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/home/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar xvfz omnetpp-6.0.2-core.tgz
```

This will create an `omnetpp-6.0.2` subdirectory with the OMNeT++ files in it.

10.5 Environment Variables

In general OMNeT++ requires that certain environment variables are set and the `omnetpp-6.0.2/bin` directory is in the PATH. Source the `setenv` script to set up all these variables.

```
$ cd omnetpp-6.0.2
$ source setenv
```

To set the environment variables permanently, edit `.profile` or `.zprofile` in your home directory and add a line something like this:

```
[ -f "$HOME/omnetpp-6.0.2/setenv" ] && source "$HOME/omnetpp-6.0.2/setenv"
```

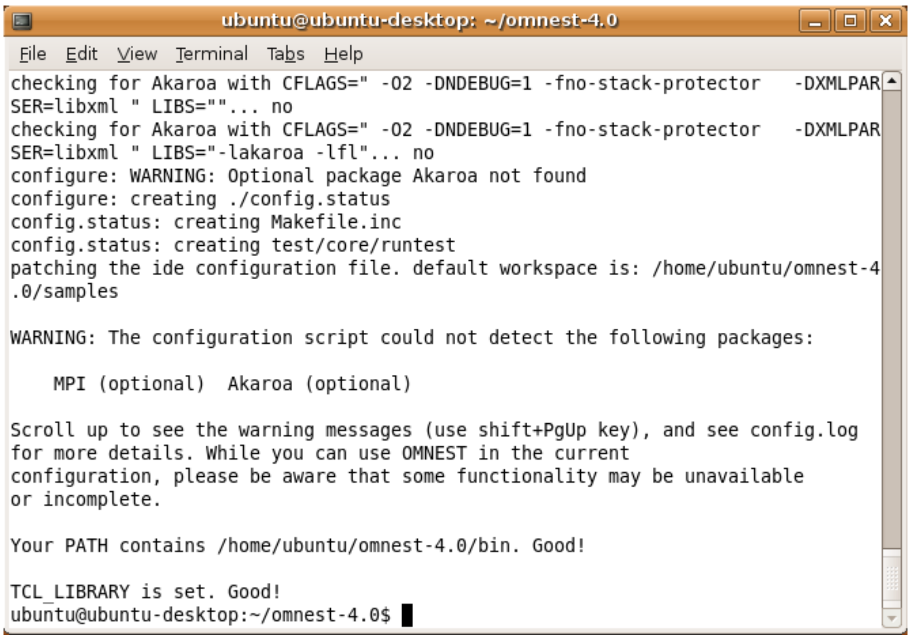
Note: If you use a shell other than bash, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

10.6 Configuring and Building OMNeT++

In the top-level OMNeT++ directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.


ubuntu@ubuntu-desktop: ~/omnest-4.0
File Edit View Terminal Tabs Help

```
checking for Akaroa with CFLAGS=" -O2 -DNDEBUG=1 -fno-stack-protector   -DXMLPAR
SER=libxml " LIBS=""... no
checking for Akaroa with CFLAGS=" -O2 -DNDEBUG=1 -fno-stack-protector   -DXMLPAR
SER=libxml " LIBS="-lakaroa -lfl"... no
configure: WARNING: Optional package Akaroa not found
configure: creating ./config.status
config.status: creating Makefile.inc
config.status: creating test/core/runtest
patching the ide configuration file. default workspace is: /home/ubuntu/omnest-4
.0/samples

WARNING: The configuration script could not detect the following packages:

    MPI (optional)  Akaroa (optional)

Scroll up to see the warning messages (use shift+PgUp key), and see config.log
for more details. While you can use OMNEST in the current
configuration, please be aware that some functionality may be unavailable
or incomplete.

Your PATH contains /home/ubuntu/omnest-4.0/bin. Good!

TCL_LIBRARY is set. Good!
ubuntu@ubuntu-desktop:~/omnest-4.0$ 
```

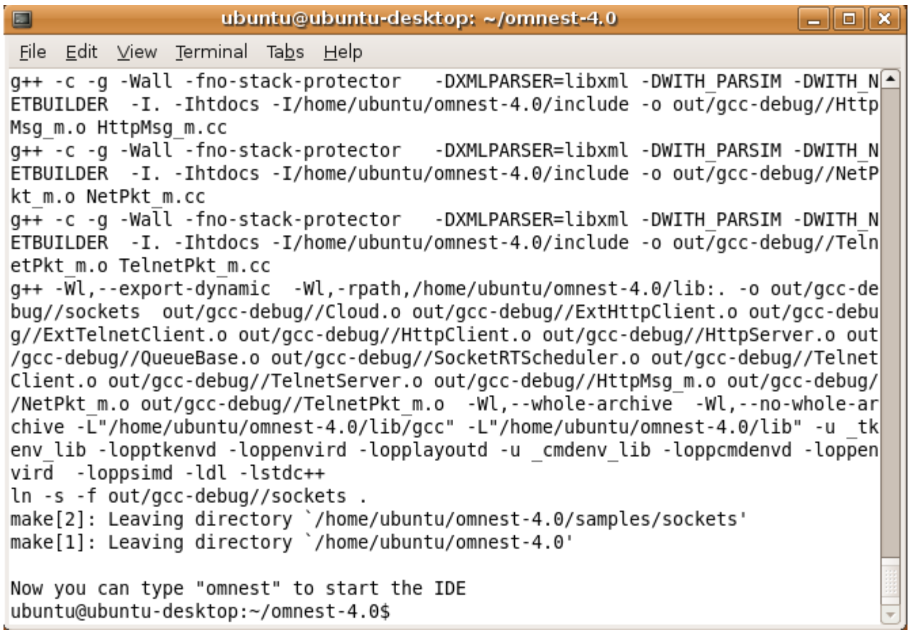
Fig. 10.1: Configuring OMNeT++

Note: If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (Use Shift+PgUp; you may need to increase the scroll-back buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

The `configure` script tries to build and run small test programs that are using specific libraries or features of the system. You can check the `config.log` file to see which test program has failed and why. In most cases the problem is that the script cannot figure out the location of a specific library. Specifying the include file or library location in the `configure.user` file and then re-running the `configure` script usually solves the problem.

When `./configure` has finished, you can compile OMNeT++. Type in the terminal:

```
$ make
```


ubuntu@ubuntu-desktop: ~/omnest-4.0
File Edit View Terminal Tabs Help

```
g++ -c -g -Wall -fno-stack-protector   -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_N
ETBUILDER  -I. -Ihtdocs -I/home/ubuntu/omnest-4.0/include -o out/gcc-debug//Http
Msg_m.o HttpMsg_m.cc
g++ -c -g -Wall -fno-stack-protector   -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_N
ETBUILDER  -I. -Ihtdocs -I/home/ubuntu/omnest-4.0/include -o out/gcc-debug//NetP
kt_m.o NetPkt_m.cc
g++ -c -g -Wall -fno-stack-protector   -DXMLPARSER=libxml -DWITH_PARSIM -DWITH_N
ETBUILDER  -I. -Ihtdocs -I/home/ubuntu/omnest-4.0/include -o out/gcc-debug//Teln
etPkt_m.o TelnetPkt_m.cc
g++ -Wl,--export-dynamic  -Wl,-rpath,/home/ubuntu/omnest-4.0/lib:. -o out/gcc-de
bug//sockets  out/gcc-debug//Cloud.o out/gcc-debug//ExtHttpClient.o out/gcc-debu
g//ExtTelnetClient.o out/gcc-debug//HttpClient.o out/gcc-debug//HttpServer.o out
/gcc-debug//QueueBase.o out/gcc-debug//SocketRTScheduler.o out/gcc-debug//Telnet
Client.o out/gcc-debug//TelnetServer.o out/gcc-debug//HttpMsg_m.o out/gcc-debug/
/NetPkt_m.o out/gcc-debug//TelnetPkt_m.o  -Wl,--whole-archive  -Wl,--no-whole-ar
chive -L"/home/ubuntu/omnest-4.0/lib/gcc" -L"/home/ubuntu/omnest-4.0/lib" -u _tk
env_lib -lopptkenvd -loppenvird -lopplayoutd -u _cmdenv_lib -loppcmdenvd -loppen
vird  -loppsimd -ldl -lstdc++
ln -s -f out/gcc-debug//sockets .
make[2]: Leaving directory `/home/ubuntu/omnest-4.0/samples/sockets'
make[1]: Leaving directory `/home/ubuntu/omnest-4.0'

Now you can type "omnest" to start the IDE
ubuntu@ubuntu-desktop:~/omnest-4.0$
```

Fig. 10.2: Building OMNeT++

Tip: To take advantage of multiple processor cores, add the `-j8` option (for 8 cores) to the `make` command line.

Note: The build process will not write anything outside its directory, so no special privileges are needed.

Tip: The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

10.7 Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the aloha simulation is started by entering the following commands:

```
$ cd samples/aloha
$ ./aloha
```

By default, the samples will run using the Qtenv environment. You should see nice gui windows and dialogs.

10.8 Starting the IDE

Note: The IDE is supported only on 64-bit versions of Windows, macOS and Linux.

You can run the IDE by typing the following command in the terminal:

```
$ omnetpp
```

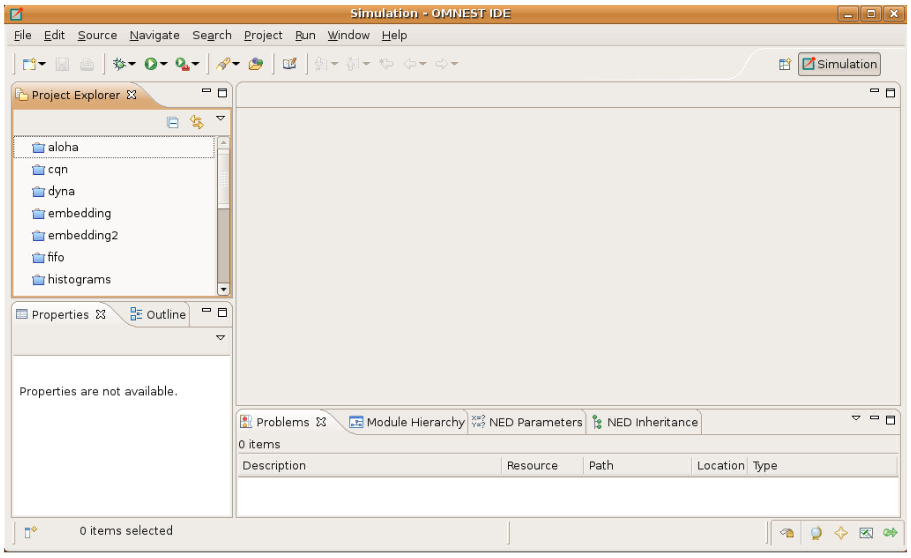


Fig. 10.3: The Simulation IDE

If you would like to be able to access the IDE from the application launcher or via a desktop shortcut, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

Note: The above commands assume that your system has the `xdg` commands, which most modern distributions do.

10.9 Optional Packages

10.9.1 Akaroa

If you wish to use Akaroa, it must be downloaded, compiled, and installed manually before installing OMNeT++.

Note: As of version 2.7.9, Akaroa only supports Linux and Solaris.

Download Akaroa 2.7.9 from: http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.chtml

Extract it into a temporary directory:

```
$ tar xfz akaroa-2.7.9.tar.gz
```

Configure, build and install the Akaroa library. By default, it will be installed into the `/usr/local/akaroa` directory.

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

Go to the OMNeT++ directory, and (re-)run the `configure` script. Akaroa will be automatically detected if you installed it to the default location.

CHAPTER

ELEVEN

BUILD OPTIONS

11.1 Configure.user Options

The `configure.user` file contains several options that can be used to fine-tune the simulation libraries.

You always need to re-run the `configure` script in the installation root after changing the `configure.user` file.

```
$ ./configure
```

After this step, you have to remove all previous libraries and recompile OMNeT++:

```
$ make cleanall
$ make
```

Options:

PREFER_CLANG=no
If both `gcc` and `clang` are installed on your system, setting this variable to `no` will force the configure script to use `gcc` as C++ compiler.

<COMPONENTNAME>_CFLAGS, <COMPONENTNAME>_LIBS
The `configure.user` file contains variables for defining the compile and link options needed by various external libraries. By default, the `configure` command detects these automatically, but you may override the auto detection by specifying the values by hand. (e.g. `<COMP>_CFLAGS=-I/path/to/comp/includedir` and `<COMP>_LIBS=-L/path/to/comp/libdir -lnameoflib`.)

WITH_PARSIM=no
Use this variable to explicitly disable parallel simulation support in OMNeT++.

WITH_NETBUILDER=no
This option allows you to leave out the NED language parser and the network builder. (This is needed only if you are building your network with C++ API calls and you do not use the built-in NED language parser at all.)

WITH_QTENV=no
This will prevent the build system to link with the Qt libraries. Use this option if your platform does not have a suitable Qt package or you will run the simulation only in command line mode. (i.e. You want to run OMNeT++ in a remote terminal session.)

WITH_OSG=no
This will prevent the build system to use OpenScreenGraph which is used for 3D visualization in Qtenv.

WITH_OSGEARTH=no
This will prevent the build system to use osgEarth which is used for 2D/3D mapping and visualization in Qtenv.

CFLAGS_[RELEASE/DEBUG]
To change the compiler command line options the build process is using, you should specify them in the `CFLAGS_RELEASE` and `CFLAGS_DEBUG` variables. By default, the flags required for debugging or optimization are detected automatically by the `configure` script. If you set them manually, you should specify all options you need. It is recommended to check what options are detected automatically (check the `Makefile.inc` after running `configure` and look for the `CFLAGS_[RELEASE/DEBUG]` variables.) and add/modify those options manually in the `configure.user` file.

LDFLAGS
Linker command line options can be explicitly set using this variable. It is recommended to check what options are detected automatically (check the `Makefile.inc` after running `configure` and look for the `LDFLAGS` variable.) and add/modify those options manually in the `configure.user` file.

SHARED_LIBS
This variable controls whether the OMNeT++ build process will create static or dynamic libraries. By default, the OMNeT++ runtime is built as a set of shared libraries. If you want to build a single executable from your simulation, specify `SHARED_LIBS=no` in `configure.user` to create static OMNeT++ libraries and then reconfigure (`./configure`) and recompile OMNeT++ (`make cleanall; make`). Once the OMNeT++ static libraries are correctly built, your own project have to be rebuilt, too. You will get a single, statically linked executable, which requires only the NED and INI files to run.

Warning: It is important to completely delete the OMNeT++ libraries (`make cleanall`) and then rebuild them, otherwise it cannot be guaranteed that the created simulations are linked against the correct libraries.

Note: The `USE_DOUBLE_SIMTIME` and `WITHOUT_CPACKET` options are no longer supported. They were introduced in OMNeT++ 4.0 to help porting model code from OMNeT++ 3.x, and having fulfilled their role, they were removed in OMNeT++ 5.0. If you still have old model code to port, use OMNeT++ 4.x.

11.2 Moving the Installation

When you build OMNeT++ on your machine, several directory names are compiled into the binaries. This makes it easier to set up OMNeT++ in the first place, but if you rename the installation directory or move it to another location in the file system, the built-in paths become invalid and the correct paths have to be supplied via environment variables.

The following environment variables are affected (in addition to `PATH`, which also needs to be adjusted):

OMNETPP_IMAGE_PATH
This variable contains the list of directories where Qtenv looks for icons. Set it to point to the `images/` subdirectory of your OMNeT++ installation.

LD_LIBRARY_PATH
This variable contains the list of additional directories where shared libraries are looked for. Initially, `LD_LIBRARY_PATH` is not needed because shared libraries are located via the *rpath* mechanism. When you move the installation, you need to add the `lib/` subdirectory of your OMNeT++ installation to `LD_LIBRARY_PATH`.

Note: On macOS, `DYLD_LIBRARY_PATH` is used instead of `LD_LIBRARY_PATH`. On Windows, the `PATH` variable must contain the directory where shared libraries (DLLs) are present.

11.3 Using Different Compilers

By default, the configure script detects the following compilers automatically in the path:

- Intel compiler (icc, icpc)
- GNU C/C++ (gcc, g++)
- Clang (clang, clang++)
- Clang/C2 (from Microsoft Visual Studio)
- Sun Studio (cc, cxx)
- IBM compiler (xlc, xlC)

If you want to use compilers other than the above ones, you should specify the compiler name in the `CC` and `CXX` variables, and re-run the configuration script.

Note: Different compilers may have different command line options. If you use a compiler other than the default `gcc`, you may have to revise the `CFLAGS_[RELEASE/DEBUG]` and `LDFLAGS` variables.