OMNeT++

Installation Guide

Version 5.7

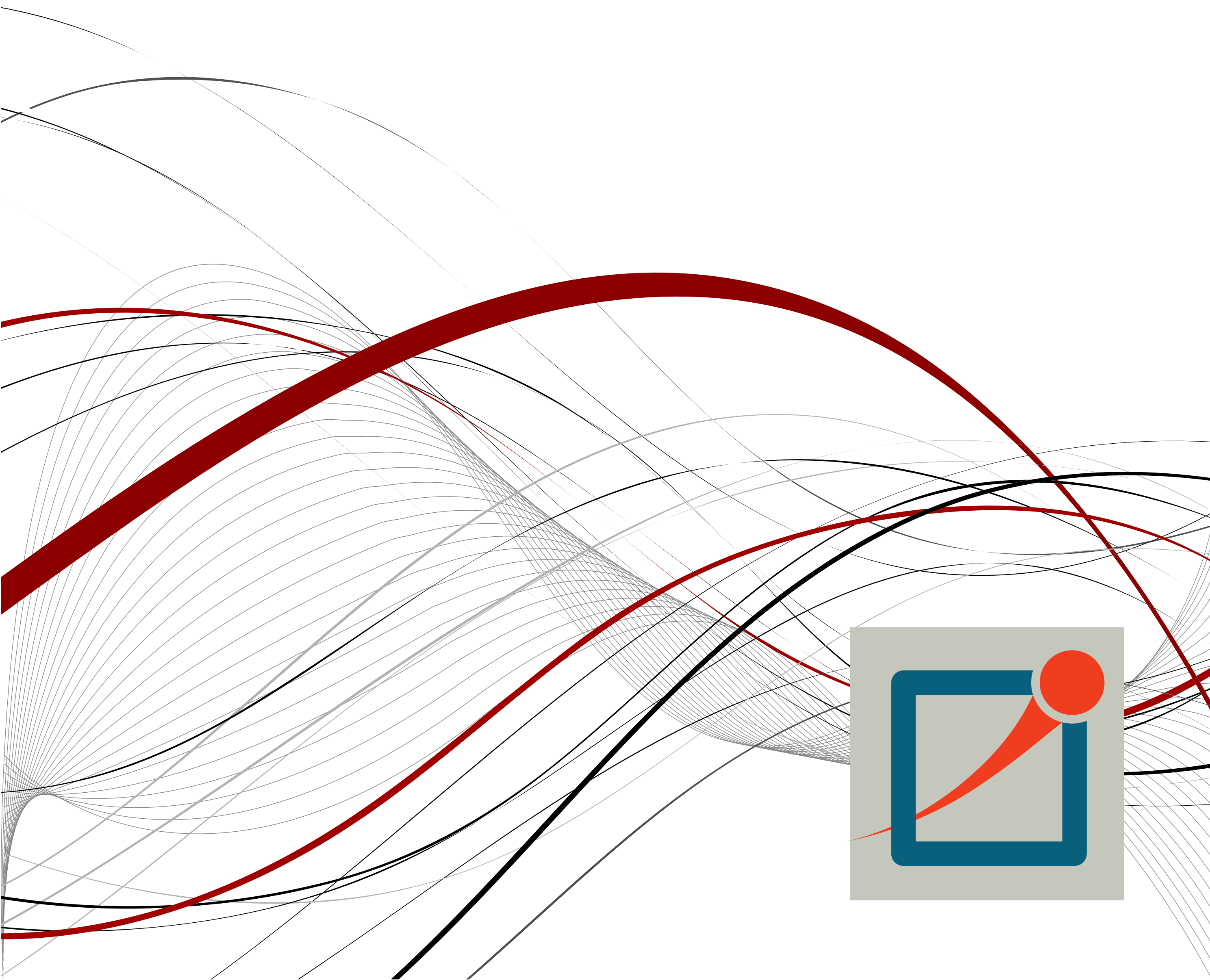


Table of Contents

1. General Information 1
2. Windows 2
3. macOS 5
4. Linux 11
5. Ubuntu 17
6. Fedora 34 20
7. Red Hat 22
8. OpenSUSE 24
9. Generic Unix 26
10. Build Options 32

Chapter 1. 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 x86_64
- MacOS 10.12 or later
- Linux distributions covered in this Installation Guide

The Simulation IDE is supported on the following platforms:

- Linux x86_64
- Windows x86_64
- MacOS 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 5.x. Certain OMNeT++ features (Qtenv, parallel simulation, XML support, etc.) depend on the availability of external libraries (Tcl/Tk, MPI, LibXML or Expat, 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 2. Windows

2.1. Supported Windows Versions

OMNeT++ supports 64-bit versions of Windows.

> 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 http://omnetpp.org. Make sure you select the Windows-specific archive, named `omnetpp-5.7-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-5.7` 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-5.7` 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-5.7/bin` directory. On the first start of the shell, you may need to wait for the extraction of the `tools` directory.

> 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. Yo can also create a shortcut for starting the IDE. To do so, locate the `omnetpp.exe` program in the `omnetpp-5.7/ide` directory in Windows Explorer, right-click it, and choose *Send To > Desktop (create shortcut)* from the menu. On Windows 7, 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

If you want to start OMNeT++ simulations outside the shell as well (for example from Exlorer), you need to add OMNeT++'s `bin`, `tools/win64/usr/bin` and `tools/win64/mingw64/bin` directories to the path.

First, open the *Environment Variables* dialog.

Click the Start button, then start typing `environment variables` into the search box. Choose *Edit environment variables for your account* when it appears in the list. The dialog comes up.

In the dialog, select `path` or `PATH` in the list, click *Edit*. Append "`;<omnetpp-dir>\bin`" and the other needed directories to the value (without quotes), where `<omnetpp-dir>` is the name of the OMNeT++ root directory (for example `C:\omnetpp-5.7`). Hit Enter to accept.

You need to log-out and then log-in for the changes to take effect.

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.

2.9. Additional Packages

Note that Doxygen and GraphViz are already included in the OMNeT++ package, and do not need to be downloaded.

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 3. macOS

3.1. Supported Releases

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

The following releases are covered:

- macOS 11.x

3.2. Installing the Prerequisite Packages

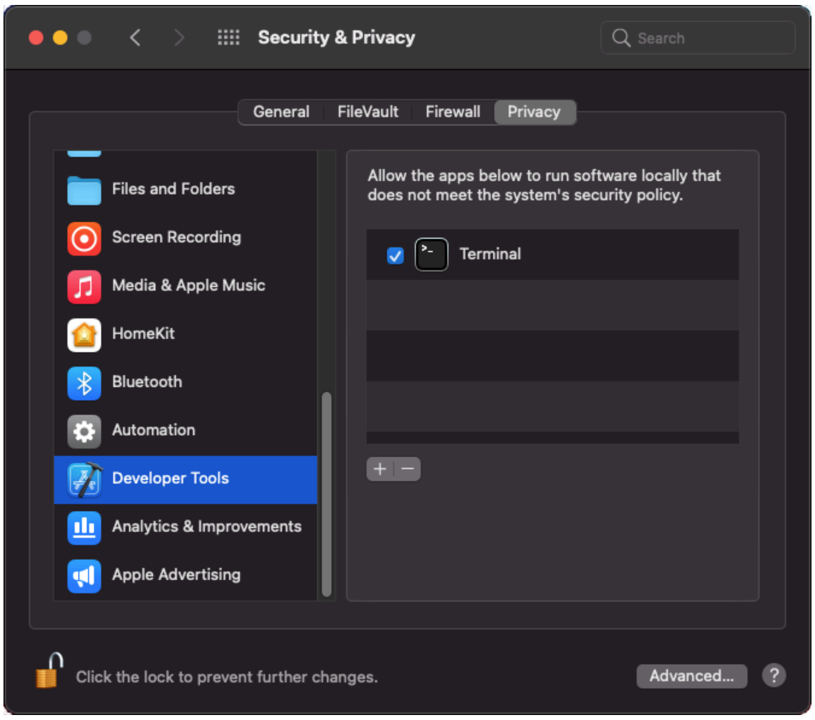
Install the command line developer tools for macOS (compiler, debugger, etc.)

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

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

3.3. 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.


Allow the apps below to run software locally that does not meet the system's security policy.

Terminal ✓

Click the lock to prevent further changes.

Figure 3.1. Enable Running Unsigned Code in Terminal

Make sure that executing developer signed code is allowed at the bottom of the **System Preferences / Security and Privacy / General** tab.

3.4. 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. Executing `source setenv` will automatically do this for you if it detects that it is running on Apple silicon.

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`.

3.5. Additional Steps Required on macOS to Use the Debugger

The Command Line Developer Tools package contains the `lldb` debugger. OMNeT++ 5.7 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.

3.6. Downloading and Unpacking OMNeT++

Download OMNeT++ from http://omnetpp.org. Make sure you select to download the macOS specific archive, `omnetpp-5.7-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-5.7-macos-x86_64.tgz
```

A subdirectory called `omnetpp-5.7` 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.

3.7. Environment Variables

OMNeT++ needs its `bin/` and `tools/macosx/bin` directories to be in the path. To add them to `PATH` temporarily (in the current shell only), change into the OMNeT++ directory and source the `setenv` script:

```
$ cd omnetpp-5.7
$ 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 "$HOME/omnetpp-5.7/setenv" ] && source "$HOME/omnetpp-5.7/setenv"
```

3.8. 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.

Normally, the `configure` script needs to be running under the graphical environment in order to test for `wish`, the Tcl/Tk shell. If you are logged in via an ssh session or you want to compile OMNeT++ without Tcl/Tk and/or Qtenv, use the command

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

instead of plain `./configure`.

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
```

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

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

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`:

```
$ make MODE=release
```

3.9. 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.

3.10. 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-5.7` 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.

3.11. 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*.

3.12. 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
```

> 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.

NOTE

The Tcl/Tk environment uses the native Aqua version of Tcl/Tk, so you will see native widgets. However, due to problems in the Tk/Aqua port, you may experience minor UI quirks. We recommend using Qtenv whenever it is possible.

3.13. Additional Packages

3.13.1. 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`.

3.13.2. GraphViz

GraphViz is needed if you want to have diagrams in HTML documentation that you generate from NED files in the IDE (*Generate NED Documentation...* item in the project context menu).

You can install it from the Homebrew project (http://brew.sh) by typing `brew install graphviz`.

After installation, make sure that the `dot` program is available from the command line. Open a terminal, and type

```
$ dot -V
```

Note the capital *V*. The command should normally work out of the box. If you get the "*command not found*" error, you need to put `dot` into the path. Find the `dot` program in the GraphViz installation directory, and soft link it into `/usr/local/bin` (`sudo ln -s <path>/dot /usr/local/bin`).

3.13.3. Doxygen

Doxygen is needed if you want to generate documentation for C++ code, as part of the HTML documentation that you generate from NED files in the IDE (*Generate NED Documentation...* item in the project context menu).

You can install it from the Homebrew project (http://brew.sh) by typing `homebrew install doxygen`.

After installation, ensure that the `doxygen` program is available from the command line. Open a terminal, and type

```
$ doxygen
```

3.13.4. 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 4. Linux

4.1. Supported Linux Distributions

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

- Ubuntu 18.04LTS, 20.04LTS
- Fedora Core 34
- RHEL 8.x
- OpenSUSE Leap 15.x

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

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

4.2. Installing the Prerequisite Packages

OMNeT++ requires several packages to be installed on the computer. These packages include the C++ compiler (gcc or clang), the Java runtime, 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.

4.3. Downloading and Unpacking

Download OMNeT++ from http://omnetpp.org. Make sure you select to download the linux specific archive, `omnetpp-5.7-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-5.7-linux-x86_64.tgz
```

This will create an `omnetpp-5.7` 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.

4.4. Environment Variables

OMNeT++ needs its `bin/` directory to be in the path. To add `bin/` to `PATH` temporarily (in the current shell only), change into the OMNeT++ directory and source the `setenv` script:

```
$ cd omnetpp-5.7
$ source setenv
```

The script also adds the `lib/` subdirectory to `LD_LIBRARY_PATH`, which may be necessary on systems that don't support the rpath mechanism.

To set the environment variables permanently, edit `.bashrc` in your home directory. Use your favourite text editor to edit `.bashrc`, for example gedit:

```
$ gedit ~/.bashrc
```

Add the following line at the end of the file, then save it:

```
export PATH=$HOME/omnetpp-5.7/bin:$PATH
```

You need to close and re-open the terminal for the changes to take effect.

Alternatively, you can put the above line into `~/.bash_profile`, but then you need to log out and log in again for the changes to take effect.

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.

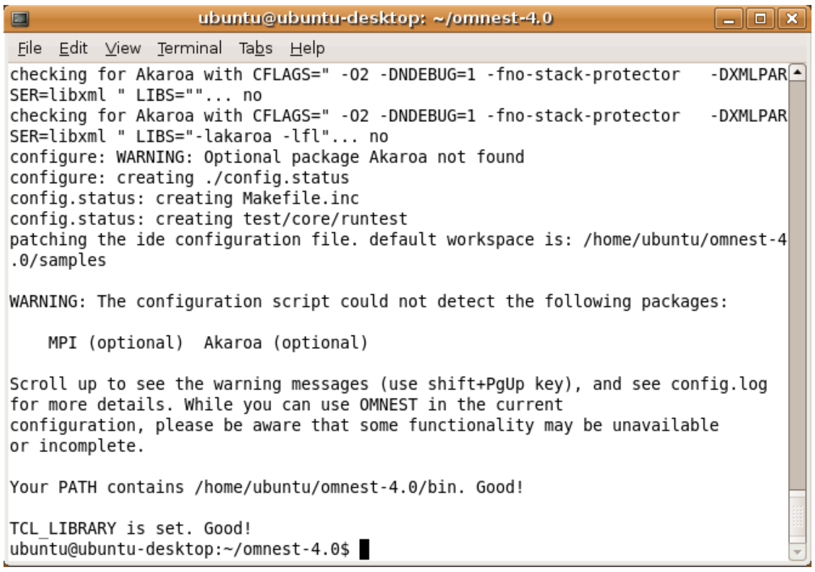
Note that all Linux distributions covered in this Installation Guide use *bash* unless the user has explicitly selected another shell.

4.5. 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$
```

Figure 4.1. Configuring OMNeT++

Normally, the `configure` script needs to be running under the graphical environment (X11) in order to test for `wish`, the Tcl/Tk shell. If you are logged in via an *ssh* session, or there is some other reason why X is not running, the easiest way to work around the problem is to tell OMNeT++ to build without Tcl/Tk and Qtenv. To do that, use the command

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

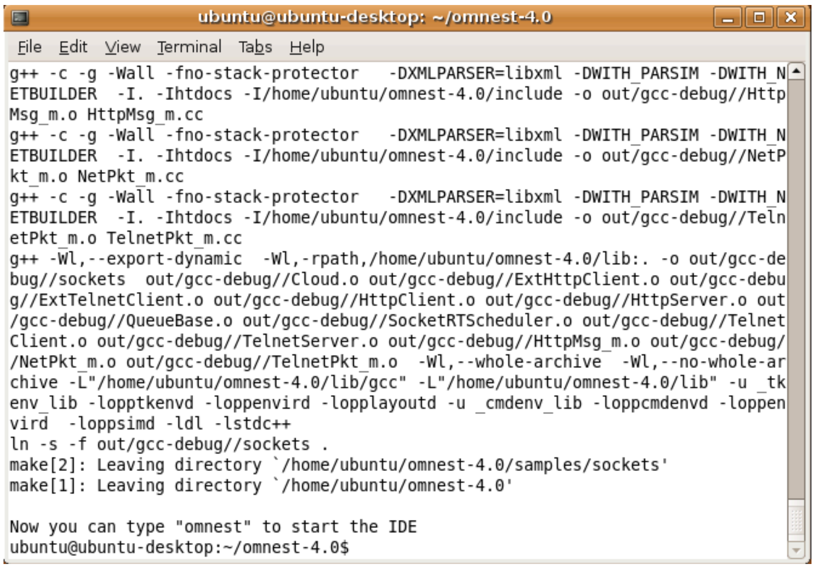
instead of plain `./configure`.

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 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
```


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$
```

Figure 4.2. Building OMNeT++

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

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

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`:

```
$ make MODE=release
```

4.6. 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.

4.7. Starting the IDE

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

```
$ omnetpp
```

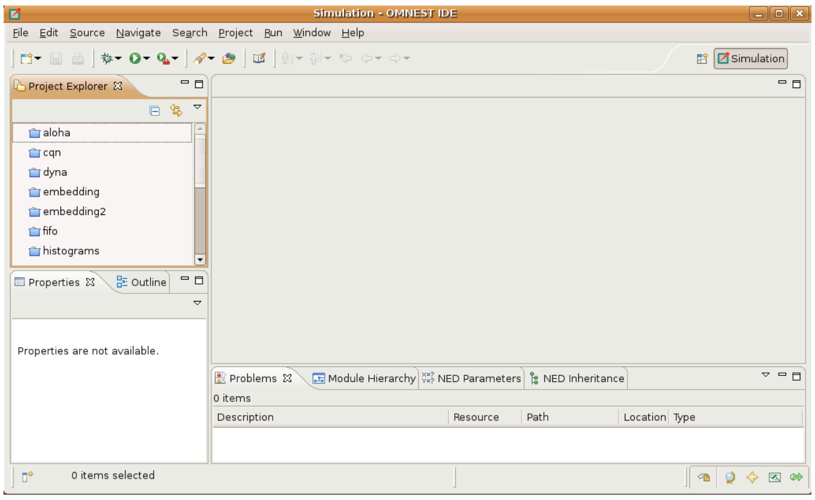


Figure 4.3. The Simulation IDE

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.

4.8. 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*.

4.9. 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.

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

4.10. Additional Packages

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

4.10.1. Qtenv

OMNeT++ 5 comes with a new 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`.

4.10.2. Akaroa

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

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.

4.10.3. 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 5. Ubuntu

5.1. 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 18.04LTS or 20.04LTS

They were tested on the following architectures:

- Intel 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.

5.2. Opening a Terminal

Type *terminal* in Dash and click on the Terminal icon.

5.3. Installing the Prerequisite Packages

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

5.3.1. 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 bison flex perl \
     python python3 qt5-default libqt5opengl5-dev tcl-dev tk-dev \
     libxml2-dev zlib1g-dev doxygen graphviz libwebkit2gtk-4.0-37
```

To use Qtenv with 3D visualization support, install the development packages for OpenSceneGraph (3.2 or later) and the osgEarth (2.7 or later) packages. (You may need to enable the *Universe* software repository in Software Sources.

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

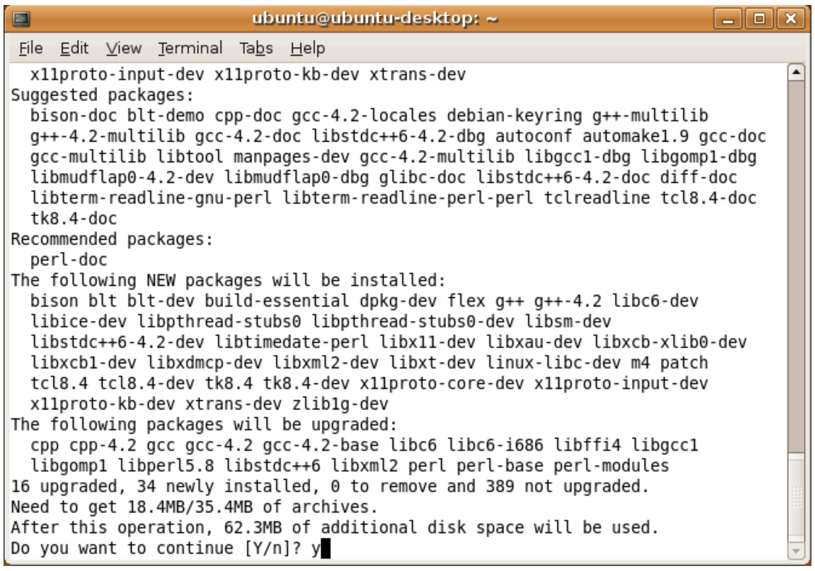
NOTE

You may opt to use gcc and g++ instead of the clang and clang++ compilers. If you do not need the 3D visualization capabilities, you can disable them in the *configure.user* file.

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

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

Figure 5.1. Command-Line Package Installation

5.3.2. Graphical Installation

Open the dash and type *Synaptic*.

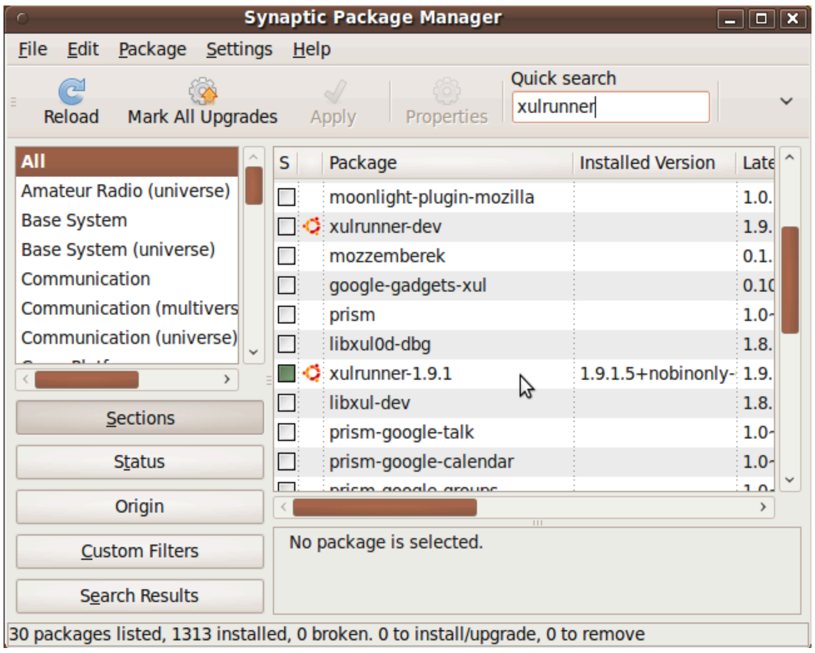
Since software installation requires root permissions, Synaptic will ask you to type your password.

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, clang, lld, bison, flex, perl, qt5-default, tcl-dev, tk-dev, libxml2-dev, zlib1g-dev, doxygen, graphviz, libwebkit2gtk-4.0-37
- recommended: libopenscenegraph-dev, openscenegraph-plugin-osgearth, libosgearth-dev, openmpi-bin, libopenmpi-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

Figure 5.2. Synaptic Package Manager

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

5.3.3. 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
```

Note that the default debugger for OMNeT++ just-in-time debugging is *Nemiver*, so it also needs to be installed:

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

Chapter 6. Fedora 34

6.1. 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 34

It was tested on the following architectures:

- Intel 64-bit

6.2. Opening a Terminal

Open the Search bar, and type *Terminal*.

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

To install the required packages, type in the terminal:

```
$ sudo dnf install make clang lld bison flex perl \
    python2 tcl-devel tk-devel qt5-devel libxml2-devel \
    zlib-devel doxygen graphviz webkitgtk
```

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
```

> You may opt to use gcc and g++ instead of the clang and clang++ compilers.

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`).

6.3.2. 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, clang, lld, flex, perl, tcl-devel, tk-devel, qt5-devel, libxml2-devel, zlib-devel, webkitgtk, doxygen, graphviz, openmpi-devel, OpenSceneGraph-devel, osgearth-devel

Click *Apply*, then follow the instructions.

Chapter 7. Red Hat

7.1. 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 7.x

It was tested on the following architectures:

- Intel 64-bit

7.2. Opening a Terminal

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

7.3. 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.

7.3.1. Command-Line Installation

To install the required packages, type in the terminal:

```
$ su -c 'yum install make clang lld bison flex perl \
    tcl-devel tk-devel qt-devel libxml2-devel zlib-devel \
    doxygen graphviz openmpi-devel libpcap-devel'
```

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).

NOTE

You may opt to use gcc and g++ instead of the clang and clang++ compilers.

To install additional (optional) packages for parallel simulation and packet capture support, type:

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

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`).`

7.3.2. 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, clang, lld, bison, flex, perl, tcl-devel, tk-devel, qt-devel, libxml2-devel, zlib-devel, make, doxygen, graphviz, openmpi-devel

Click *Apply*, then follow the instructions.

7.4. 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 8. OpenSUSE

8.1. 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.x

It was tested on the following architectures:

- Intel 64-bit

8.2. Opening a Terminal

Open the Search bar, and type *Terminal*.

8.3. Installing the Prerequisite Packages

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

8.3.1. Command-Line Installation

To install the required packages, type in the terminal:

```
$ sudo zypper install make clang lld bison flex perl \
    tcl-devel tk-devel libqt5-qtbase-devel libxml2-devel zlib-devel \
    doxygen graphviz libwebkitgtk-4_0-37
```

> You may opt to use gcc and g++ instead of the clang and clang++ compilers.

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:

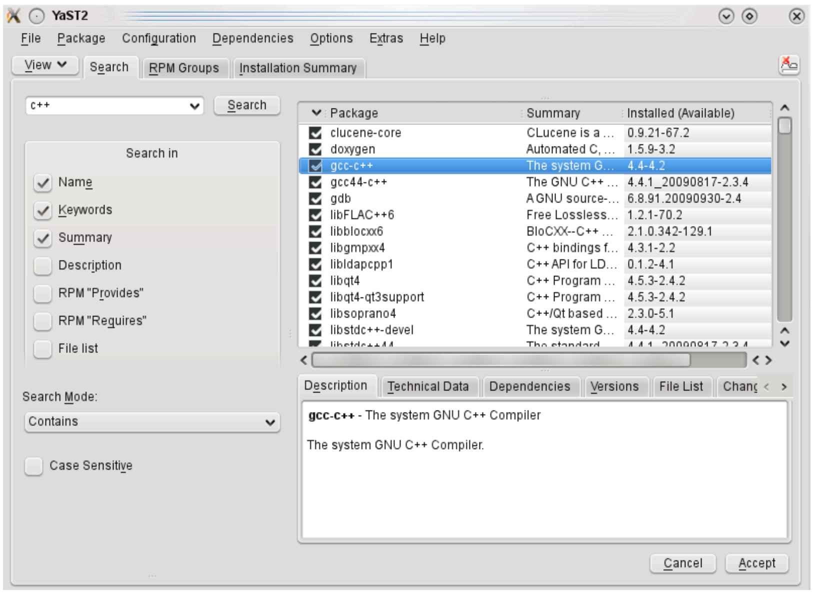
```
$ sudo zypper install openmpi-devel
```

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
```

8.3.2. Graphical Installation

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



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.

Figure 8.1. 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, clang, lld,, bison, flex, perl, tcl-devel, tk-devel, libqt5-qtbase-devel, libxml2-devel, zlib-devel, doxygen, graphviz, libwebkitgtk-4_0-37, openmpi-devel

Click *Accept*, then follow the instructions.

Chapter 9. Generic Unix

9.1. 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 64-bit platforms. Other operating systems (FreeBSD, Solaris, etc.) and architectures may still be used as simulation platforms, without the IDE.

9.2. Dependencies

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

build-essential, GNU make, gcc, g++, bison (3.0+), flex, perl	These packages are needed for compiling OMNeT++ and simulation models, and also for certain OMNeT++ tools to work. C++ compilers other than g++, will also be accepted.

NOTE You may opt to use clang and clang++ instead of the gcc and g++ compilers.

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

Tcl/Tk 8.5 or later	Required by the Tkenv simulation runtime environment. You need the *devel* packages that include the C header files as well. It is also possible to compile OMNeT++ without Tcl/Tk (and Tkenv), by setting then `WITH_TKENV=no` variable in configure.user.
LibXML2 or Expat	Either one of these XML parsers are needed for OMNeT++ to be able to read XML files. The *devel* packages (that include the header files) are needed. LibXML2 is the preferred one.
Qt 5.4 or later	Required by the Qtenv simulation runtime environment. You need the *devel* packages that include header files as well.
OpenSceneGraph (3.2+) and osgEarth (2.7+)	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:

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.

9.3. 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.

9.3.1. Tcl/Tk

Tcl/Tk may be present as separate packages (`tcl` and `tk`), or in one package (`tcltk`). The version number (e.g. 8.5) is usually part of the name in some form (`85`, `8.5`, etc). You will need the development packages, which are usually denoted with the `-dev` or `-devel` name suffix.

Troubleshooting:

If your platform does not have suitable Tcl/Tk and/or QT packages, you may still use OMNeT++ to run simulations from the command line. To disable the graphical runtime environment use:

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

This will prevent the build system to link with Tcl/Tk or QT libraries. This is required also if you are installing OMNeT++ from a remote terminal session.

By default, the `configure` script expects to find the Tcl/Tk libraries in the standard linker path (without any `-Ldirectory` linker option) and under the standard names (i.e. with the `-ltcl8.5` or `-ltcl85` linker option). If you have them in different places or under different names, you have to edit `configure.user` and explicitly set `TK_LIBS` there (see the Build Options chapter for further details).

If you get the error *no display and DISPLAY environment variable not set*, then you're either not running X (the `wish` command, and thus `./configure` won't work just in the console) or you really need to set the `DISPLAY` variable (`export DISPLAY=:0.0` usually does it).

If you get the error: *Tcl_Init failed: Can't find a usable init.tcl...*

The `TCL_LIBRARY` environment variable should point to the directory which contains `init.tcl`. That is, you probably want to put a line like

```
export TCL_LIBRARY=/usr/lib/tcl8.5
```

into your `~/.bashrc`.

9.3.2. MPI

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

9.4. Downloading and Unpacking

Download OMNeT++ from http://omnetpp.org. Make sure you select to download the generic archive, `omnetpp-5.7-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-5.7-core.tgz
```

This will create an omnetpp-5.7 subdirectory with the OMNeT++ files in it.

9.5. Environment Variables

In general OMNeT++ requires that its bin directory should be in the PATH. You should add a line something like this to your .bashrc:

```
$ export PATH=$HOME/omnetpp-5.7/bin:$PATH
```

You may also have to specify the path where shared libraries are loaded from. Use:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/omnetpp-5.7/lib
```

If configure complains about not finding the Tcl library directory, you may specify it by setting the TCL_LIBRARY environment variable.

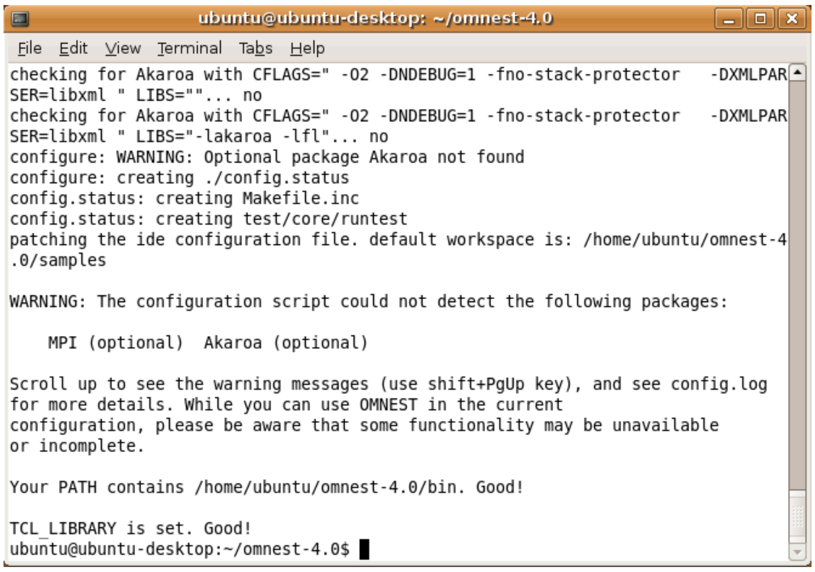
NOTE If you use a shell other than bash, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

9.6. 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   -DXMLPARSER=libxml " LIBS=""... no
checking for Akaroa with CFLAGS=" -O2 -DNDEBUG=1 -fno-stack-protector   -DXMLPARSER=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$
```

Figure 9.1. Configuring OMNeT++

Normally, the configure script needs to be running under the graphical environment (X11) in order to test for wish, the Tcl/Tk shell. If you are logged in via an ssh session, or there is some other reason why X is not running, the easiest way to work around the problem is to tell OMNeT++ to build without Tcl/Tk. To do that, use the command

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

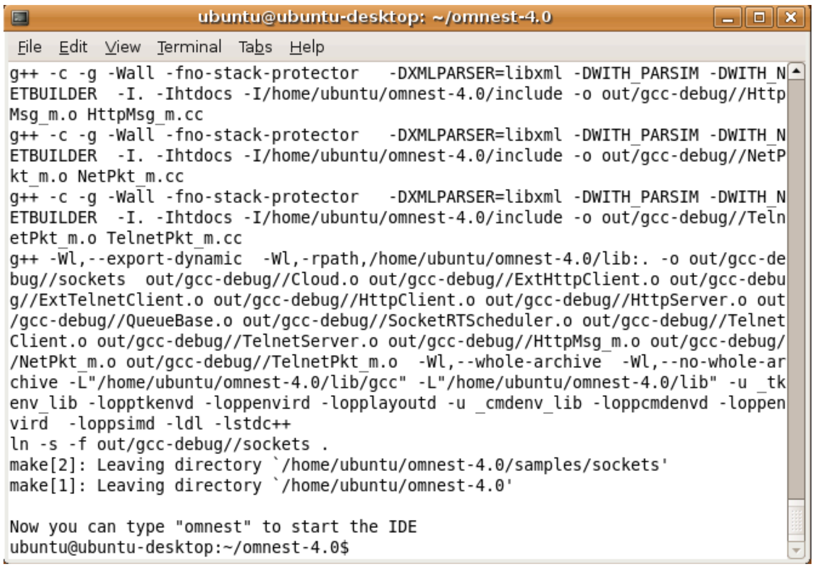
instead of plain `./configure`.

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 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.

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$
```

Figure 9.2. Building OMNeT++

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

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

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`:

```
$ make MODE=release
```

9.7. 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.

9.8. Starting the IDE

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

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

```
$ omnetpp
```

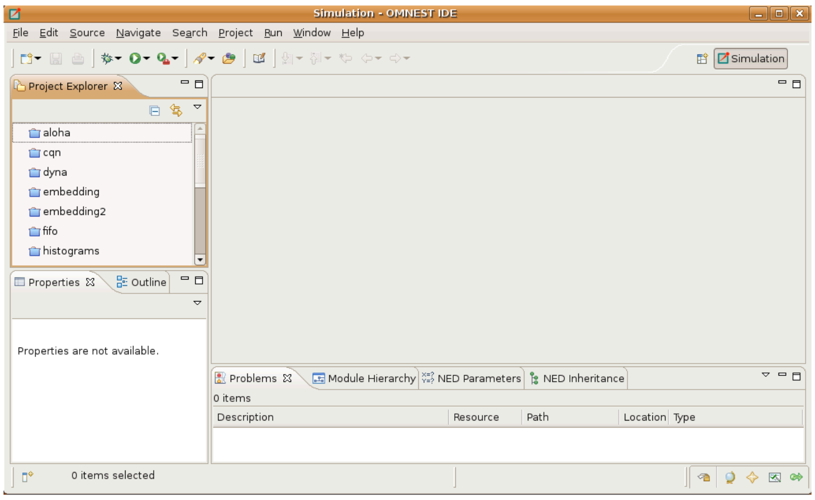


Figure 9.3. The Simulation IDE

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.

9.9. Optional Packages

9.9.1. 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 10. Build Options

10.1. 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_TKENV=no`	This will prevent the build system to link with Tcl/Tk libraries. Use this option if your platform does not have a suitable Tcl/Tk 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_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.
`PREFER_QTENV=no`	Use Tkenv as the default graphical runtime instead of Qtenv.
`EMBED_TCL_CODE=no`	Tcl/Tk is a script language and the source of the graphical runtime environment is stored as `.tcl` files in the `src/tkenv` directory. By default, these files are not used directly, but are embedded as string literals in the executable file. Setting `EMBED_TCL_CODE=yes` allows you to move the OMNeT++ installation without caring about the location of the `.tcl` files. If you want to make changes to the Tcl code, you better switch off the embedding with the `EMBED_TCL_CODE=no` option. This way you can make changes to the `.tcl` files and see the changes immediately without recompiling the OMNeT++ libraries.
`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.

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.

10.2. 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 Tkenv looks for icons. Set it to point to the `images/` subdirectory of your OMNeT++ installation.
`OMNETPP_TKENV_DIR`	This variable points to the directory that contains the Tcl script parts of Tkenv, which is by default the `src/tkenv/` subdirectory of your OMNeT++ installation. Normally you don't need to set this variable, because the Tkenv shared library contains all Tcl code compiled in as string literals. However, if you compile OMNeT++ with the `EMBED_TCL_CODE=no` setting and then you move the installation, then you need to set `OMNETPP_TKENV_DIR`, otherwise Tkenv won't start.
`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.

10.3. 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.

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.