NS3-AI Installation Guide

Gabriel Aldair Villagrán Saucedo

Jose Saul Castillo Ipiña

Dr. Pedro David Arjona Villicaña

Dr. Juan Carlos Cuevas Tello

Introduction

This is a detailed installation guide for ns-3 and ns3-ai modules. This guide was developed for the project of Machine Learning subject imparted by Dr. Juan Carlos Cuevas Tello and supervised by Dr. Pedro David Arjona Villicaña and **using as Operating System Ubuntu 20.04.**
The ns-3 simulator is an open source networking simulation tool implemented by C++ and python and is highly used for network research and education.
ns3-ai provides a high-efficiency solution to enable the data interaction between ns-3 and other python based AI frameworks. This module does not provide any AI algorithms but instead is providing a Python module that enables AI interconnect, so the AI framework needs to be separately installed.

Installation of ns-3

ns-3 is primarily developed on GNU/Linux and macOS platforms, and the minimal requirements to run basic simulations are a C++ compiler; either g++ or clang++ compiler, and Python (version 3) interpreter, in this document you can find how to install the prerequisites and the ns-3 simulator.

Prerequisites
The core of ns-3 requires a gcc/g++ installation of 4.9 or greater (Linux), or a recent version of clang compiler (OS X, Linux, or BSD), and Python 3.5 or greater.

Ubuntu 20.04
This is the minimal set of packages needed to work with Python bindings :

```
sudo apt-get install g++ python3 python3-dev pkg-config sqlite3 cmake
```

Additional minimal requirements for Python (development):

```
sudo apt-get install python3-setuptools git
```

Netanim animator: qt5 development tools are needed for Netanim animator:

```
sudo apt-get install qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools
```

```
sudo apt-get install qt5-default
```

Support for ns-3-pyviz visualizer:

```
sudo apt-get install gir1.2-goocanvas-2.0 python3-gi
python3-gi-cairo python3-pygraphviz gir1.2-gtk-3.0
ipython3
```

Support for MPI-based distributed emulation:

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

Support for bake build tool:

```
sudo apt-get install autoconf cvs bzr unrar
```

Debugging:

```
sudo apt-get install gdb valgrind
```

Support for utils/check-style.py code style check program:

```
sudo apt-get install uncrustify
```

Doxygen and related inline documentation:

```
sudo apt-get install doxygen graphviz imagemagick
```

```
sudo apt-get install texlive texlive-extra-utils
texlive-latex-extra texlive-font-utils dvipng latexmk
```

NOTE: If you get an error in this part such as 'convert ... not authorized source-temp/figures/lena-dual-stripe.eps'**, see this post about editing ImageMagick's security policy configuration: https://cromwell-intl.com/open-source/pdf-not-authorized.html. In brief, you will want to make this kind of change to ImageMagick security policy:**

```
--- ImageMagick-6/policy.xml.bak     2020-04-28 21:10:08.564613444 -0700
+++ ImageMagick-6/policy.xml 2020-04-28 21:10:29.413438798 -0700
@@ -87,10 +87,10 @@
   <policy domain="path" rights="none" pattern="@*"/>
-  <policy domain="coder" rights="none" pattern="PS" />
+  <policy domain="coder" rights="read|write" pattern="PS" />
   <policy domain="coder" rights="none" pattern="PS2" />
   <policy domain="coder" rights="none" pattern="PS3" />
   <policy domain="coder" rights="none" pattern="EPS" />
-  <policy domain="coder" rights="none" pattern="PDF" />
+  <policy domain="coder" rights="read|write" pattern="PDF" />
   <policy domain="coder" rights="none" pattern="XPS" />
 </policymap>
```

The ns-3 manual and tutorial are written in reStructuredText for Sphinx (doc/tutorial, doc/manual, doc/models), and figures typically in dia (also needs the texlive packages above):

```
sudo apt-get install python3-sphinx dia
```

GNU Scientific Library (GSL) support for more accurate 802.11b WiFi error models (not needed for OFDM):

```
sudo apt-get instal gsl-bin libgsl-dev libgslcblas0
```

NOTE: If the above doesn't work (doesn't detect GSL on the system), consult: https://coral.ise.lehigh.edu/jild13/2016/07/11/hello/. But don't worry if you are not using 802.11b models.

To read pcap packet traces:

```
sudo apt-get install tcpdump
```

Database support for statistics framework:

```
sudo apt-get install sqlite sqlite3 libsqlite3-dev
```

Xml-based version of the config store (requires libxml2 >= version 2.7):

```
sudo apt-get install libxml2 libxml2-dev
```

Support for generating modified python bindings:

```
sudo apt-get install cmake libc6-dev libc6-dev-i386
libclang-dev llvm-dev automake python3-pip python3 -m pip
install --user cxxfilt
```

A GTK-based configuration system:

```
sudo apt-get install libgtk-3-dev
```

To experiment with virtual machines and ns-3

```
sudo apt-get install vtun lxc uml-utilities
```

Support for openflow module (requires libxml2-dev if not installed above) and Boost development libraries

```
sudo apt-get install libxml2 libxml2-dev libboost-all-dev
```

Installation

Installation

There are different ways to install ns-3 which are:

- Installing with bake
- Installing manually using git
- Installing manually using tarball

In this document I am showing you the installation using bake.

Installation with Bake

Bake is a new tool for installing, building and finding out the missing requirements for ns-3 in your own environment.

To use Bake you need to have at least Python (2.7 or above) and Git in your machine (see the section Prerequisites above to see how to install these).

First you need to download Bake using Git, go to where you want Bake to be installed and call.

```
git clone https://gitlab.com/nsnam/bake
```

It is advisable to add bake to your path.

```
export BAKE_HOME=`pwd`/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```

After that you can use Bake to find the missing packages, download, build and install ns-3 and its modules.

To find out what is missing in your system and may be needed for installing ns-3 you can call bake check:

```
bake.py check
```

You should have seen something like:

```
> Python - OK
   > GNU C++ compiler - OK
   > Mercurial - OK
   > CVS - OK
   > GIT - OK
   > Bazaar - OK
   > Tar tool - OK
   > Unzip tool - OK
   > Unrar tool - OK
   > 7z  data compression utility - OK
   > XZ data compression utility - OK
   > Make - OK
   > cMake - OK
   > patch tool - OK
   > autoreconf tool - OK
   > Path searched for tools: /usr/lib64/qt-3.3/bin
 /usr/lib64/ccache /usr/local/bin  /usr/bin/bin/usr/local/sbin /usr/sbin
 /sbin /user/dcamara/home/scripts/user/dcamara/home/INRIA/Programs/bin
 /user/dcamara/home/INRIA/repos/llvm/build/Debug+Asserts/bin
```

Before downloading and building ns-3 you need to configure bake to inform it which are the modules you want added to ns-3, the standard distribution for example.

```
bake.py configure -e ns-<version>
```

NOTE: In this command you have to choose the version that works with the ns3-ai module, the version that you have to choose is 3.35 (e.g):

```
bake.py configure -e ns-3.35
```

Then to see the modules it has added, and the specific system requirements for this configuration, you can call bake show:

```
bake.py show
```

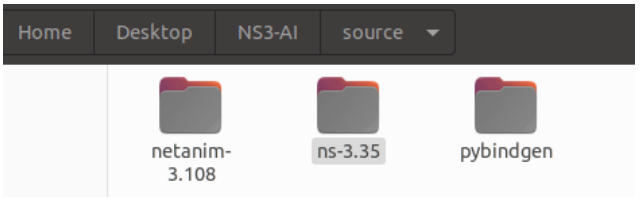
To download the modules, build and install you can call bake deploy

```
bake.py deploy
```

This will download the selected modules, all their dependencies and build ns-3 with all these independent modules, it will take you some minutes.

Configuration with waf

There are different ways to configure ns-3, for using **./waf** you need to be in the directory "**ns-3.35**" :



Once you are in you can use the command **./waf,** to see valid configure options, type

./waf --help.

NOTE: The most important option is -d <debug level>. Valid debug levels (which are listed in waf --help) are: "debug" or "optimized". It is also possible to change the flags used for compilation with.

Unlike some other build tools, to change the build target, the option must be supplied during the configure stage rather than the build stage (i.e., "./waf -d optimized" will not work; instead, do:

```
./waf -d optimized configure; ./waf
```

Of course, you can run gdb in emacs, or use your favorite debugger such as ddd or insight just as easily. In an optimized build you can find the executable for the first.cc example as build/examples/ns-<version>-first-optimized.

In order to forcibly disable python bindings, you can provide the following option:

```
./waf --disable-python configure
```

In order to tell the build system to use the sudo program to set the suid bit if required, you can provide the following option:

```
./waf --enable-sudo configure
```

To start over a configuration from scratch, type:

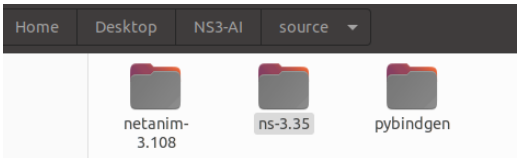
```
./waf clean
```

Or if you get stuck and all else fails:

```
rm -rf build
```

Validating

ns-3 has unit tests that can be run to verify the installation to run this tests you have to be on the ns-3.35 directory :



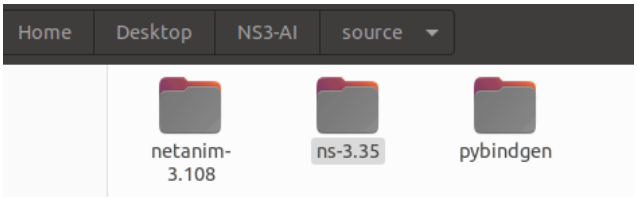

```
./waf configure --enable-tests
./test.py
```

which should produce an output like this:

```
PASS: TestSuite histogram
PASS: TestSuite ns3-wifi-interference
PASS: TestSuite ns3-tcp-cwnd
PASS: TestSuite ns3-tcp-interoperability
PASS: TestSuite sample
...
```

Running examples

for using the different examples that are already created you need to be in the directory "**ns-3.35**" :



Once you are in the directory execute:

```
./waf -d optimized configure; ./waf
```

For example, in a debug build you can find the executable for the first.cc example as build/examples/first. You can debug the executable directly by:

NOTE: You can choose the directory that you want to try

```
ndnsim@NDN:~/Desktop/NS3-AI/source/ns-3.35/build/examples$ ls
channel-models  error-model  matrix-topology  realtime  socket  tcp              tutorial  udp-client-server
energy          ipv6         naming           routing   stats   traffic-control  udp       wireless
```

in this example I am using the **tutorial** directory

```
./waf shell
cd build/examples/tutorial
./<name of the executable> (e.g.):
./ns3.35-hello-simulator-debug
```

```
ndnsim@NDN:~/Desktop/NS3-AI/source/ns-3.35$ ./ns3.35-hello-simulator-debug
bash: ./ns3.35-hello-simulator-debug: No such file or directory
ndnsim@NDN:~/Desktop/NS3-AI/source/ns-3.35$ ./waf shell
Waf: Entering directory `/home/ndnsim/Desktop/NS3-AI/source/ns-3.35/build'
Waf: Leaving directory `/home/ndnsim/Desktop/NS3-AI/source/ns-3.35/build'
Build commands will be stored in build/compile_commands.json
ndnsim@NDN:~/Desktop/NS3-AI/source/ns-3.35$ cd build/examples/tutorial
ndnsim@NDN:~/Desktop/NS3-AI/source/ns-3.35/build/examples/tutorial$ ./ns3.35-hel
lo-simulator-debug
Hello Simulator
ndnsim@NDN:~/Desktop/NS3-AI/source/ns-3.35/build/examples/tutorial$
```

Installation of ns3-ai module

The module needs to be built with ns-3, this are the instruction that I used to install the module:

```
cd ns-3.35/contrib
git clone https://github.com/hust-diangroup/ns3-ai.git
```

Once I clone the repository I have to rebuild ns-3

```
./waf configure
./waf
```

Resources

[Installation - Nsnam]

[hust-diangroup/ns3-ai: Enable the interaction between ns-3 and popular frameworks using Python, which mean you can train and test your AI]

algorithms in ns-3 without changing any frameworks you are using now! (github.com)