CN ASSIGNMENT - 5

Name- Asmit Kumar Panika
Id- 2020UCP1165

1. About ns-3

ns-3 is a discrete-event simulator typically run from the command line. It is written directly in C++, not in a high-level modeling language; simulation events are simply C++ function calls, organized by a scheduler.

An ns-3 user will obtain the ns-3 source code (see below), compile it into shared (or static) libraries, and link the libraries to main() programs that he or she authors. The main() program is where the specific simulation scenario configuration is performed and where the simulator is run and stopped. Several example programs are provided, which can be modified or copied to create new simulation scenarios. Users also often edit the ns-3 library code (and rebuild the libraries) to change its behavior.

ns-3 has optional Python bindings for authoring scenario configuration programs in Python (and using a Python-based workflow).

Many simulation tools exist for network simulation studies. Below are a few distinguishing features of ns-3 in contrast to other tools.

- ns-3 is designed as a set of libraries that can be combined together and also with other external software libraries. While some simulation platforms provide users with a single, integrated graphical user interface environment in which all tasks are carried out, ns-3 is more modular in this regard. Several external animators and data analysis and visualization tools can be used with ns-3. However, users should expect to work at the command line and with C++ and/or Python software development tools.

- ns-3 is primarily used on Linux or macOS systems, although support exists for BSD systems and also for Windows frameworks that can build Linux code, such as Windows Subsystem for Linux, or Cygwin. Native Windows Visual Studio is not presently supported although a developer is working on future support. Windows users may also use a Linux virtual machine.

2. Prerequisite before installing

Prerequisite Package/version

C++ compiler :- clang++ or g++ (g++ version 4.9 or greater)
Python :- python2 version >= 2.7.10, or python3 version >=3.4
Git :- any recent version (to access ns-3 from GitLab.com)

Tar :- any recent version (to unpack an ns-3 release)
Bunzip2 :- any recent version (to uncompress an ns-3 release)

Check version of C++ using g++ --version, similarly version of python using python --version and similarly for others as well.

3. Downloading a release of ns-3 as a source archive

This option is for the new user who wishes to download and experiment with the most recently released and packaged version of ns-3. ns-3 publishes its releases as compressed source archives, sometimes referred to as a tarball. A tarball is a particular format of software archive where multiple files are bundled together and the archive is usually compressed. The process for downloading ns-3 via tarball is simple; you just have to pick a release, download it and uncompress it.

Let's assume that you, as a user, wish to build ns-3 in a local directory called workspace. If you adopt the workspace directory approach, you can get a copy of a release by typing the following into your Linux shell (substitute the appropriate version numbers, of course)

```
$ cd
$ mkdir workspace
$ cd workspace
$ wget https://www.nsnam.org/release/ns-allinone-3.36.1.tar.bz2
$ tar xjf ns-allinone-3.36.1.tar.bz2
```

Notice the use above of the wget utility, which is a command-line tool to fetch objects from the web; if you do not have this installed, you can use a browser for this step.

Following these steps, if you change into the directory ns-allinone-3.36.1, you should see a number of files and directories

```
$ cd ns-allinone-3.36.1
$ ls
bake            constants.py        ns-3.36.1           README.md
build.py        netanim-3.108       pybindgen-0.22.1    util.py
```

4. Building with build.py

Note: This build step is only available from a source archive release described above; not from downloading via git or bake.

When working from a released tarball, a convenience script available as part of ns-3-allinone can orchestrate a simple build of components. This program is called build.py. This program will get the project configured for you in the most commonly useful way. However, please note that more advanced configuration and work with ns-3 will typically involve using the native ns-3 build system, CMake, to be introduced later in this tutorial.

If you downloaded using a tarball you should have a directory called something like ns-allinone-3.36.1 under your ~/workspace directory. Type the following:

```
$ ./build.py --enable-examples --enable-tests
```

5. Testing ns-3

You can run the unit tests of the ns-3 distribution by running the ./test.py script:

```
$ ./test.py --no-build
```

These tests are run in parallel by ns3. You should eventually see a report saying that 92 of 92 tests passed (92 passed, 0 failed, 0 crashed, 0 valgrind errors)

6. Running a script

We typically run scripts under the control of Waf. To run a program, simply use the --run option in Waf. NS-3 by default provides various scripts out of which hello world program is always there.

```
$ ./waf --run hello-simulator
```

Waf first checks to make sure that the program is built correctly and executes a build if required.

Waf then executes the program, which produces the following output.

```
Hello Simulator
```

What do I do if I don't see the output?

If you see ns3 messages indicating that the build was completed successfully, but do not see the "Hello Simulator" output, chances are that you have switched your build mode to optimized in the Building with the ns3 CMake wrapper section, but have missed the change back to debug mode. All of the console output used in this tutorial uses a special ns-3 logging component that is useful for printing user messages to the console. Output from this component is automatically disabled when you compile optimized code – it is "optimized out." If you don't see the "Hello Simulator" output, type the following:

```
$ ./ns3 configure --build-profile=debug --enable-examples --enable-tests
```

to tell ns3 to build the debug versions of the ns-3 programs that includes the examples and tests. You must still build the actual debug version of the code by typing

```
$ ./ns3
```

Now, if you run the hello-simulator program, you should see the expected output