H2020 i**P** **O**ver **I**c**N**– the bet**T**er IP (POINT)

How to configure NS3 with Click Modular Router and Blackadder

Author:

Mohammed Qasim Al-Khalidi

<u>Contents:</u>

1- <u>Introduction:</u>

This manual describes the steps to configure, compile and run NS3 with Blackadder and Click modular router. It also shows how to setup a virtual testbed simulation environment in NS3 for running Blackadder. This process has been tested with NS3.24 released on 15 September 2015 and Linux Ubuntu 15.04 operating system and all the installation and configuration steps that follow will be oriented towards this environment even if not explicitly mentioned throughout the manual. Here we assume that Click and Blackadder are already installed and compiled on your OS, if not please refer to the Blackadder HowTo manual available in the master directory of your Blackadder installation. The NS3 folder in the master Blackadder directory includes two variations of NS3 versions supported. The first is ns3.15 and the other is the ns3.24.

```
<BLACKADDER PREFIX>/ns3/ns3.15
<BLACKADDER PREFIX>/ns3/ns3.24
```

Please refer to the relevant path during the installation steps in this manual according to the version of NS3 used on your machine. Use the former path if you are using NS3.15 or below, and use the later path if you are using any NS3 version above NS3.15.

2- Installing NS3:

Below is a summarization of the steps needed to install NS3 as part of the tested environment described in the introduction section above. If you have previously installed NS3, you can skip this section and directly proceed with configuring click and NS3 in section 3. For any further information, please refer to the URL https://www.nsnam.org/wiki/Installation which includes a complete installation guide for NS3 on all dominant operating systems.

2.1 NS3 Prerequisites:

The following list of packages is necessary for Linux Ubuntu operating systems to support different ns-3 options:

- Minimal requirements for C++ (release): This is the minimal set of packages needed to run ns-3 from a released tarball.

```
apt-get install gcc g++ python
```

- Minimal requirements for Python (release): This is the minimal set of packages needed to work with Python bindings from a released tarball.

```
apt-get install gcc g++ python python-dev
```

- qt4 development tools (Note: qt4, not qt5) needed for NetAnim animator

```
apt-get install qt4-dev-tools libqt4-dev
```

- Mercurial is needed to work with ns-3 development repositories.

```
apt-get install mercurial
```

- Running python bindings from the ns-3 development tree (ns-3-dev) requires bazaar

```
apt-get install bzr
```

- Support for generating modified python bindings

```
 apt-get install cmake libc6-dev libc6-dev-i386 g++-
multilib
```

- Debugging:

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

- GNU Scientific Library (GSL) support for more accurate WiFi error models

```
 apt-get install gsl-bin libgsl0-dev libgsl0ldbl
```

- The Network Simulation Cradle (nsc) requires the flex lexical analyzer and bison parser generator:

```
 apt-get install flex bison libfl-dev
```

- To read pcap packet traces

```
apt-get install tcpdump
```

- Database support for statistics framework

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

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

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

- A GTK-based configuration system

```
 apt-get install libgtk2.0-0 libgtk2.0-dev
```

- To experiment with virtual machines and ns-3

```
 apt-get install vtun lxc
```

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

```
apt-get install uncrustify
```

- Doxygen and related inline documentation:

```
apt-get install doxygen graphviz imagemagick
apt-get install texlive texlive-extra-utils texlive-
latex-extra texlive-font-utils texlive-lang-portuguese
dvipng
```

- 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):

```
 apt-get install python-sphinx dia
```

Note: Sphinx version >= 1.12 required. To check your version, type "sphinx-build". To fetch this package alone, outside of the Ubuntu package system, try "sudo easy_install -U Sphinx".

- Support for Gustavo Carneiro's ns-3-pyviz visualizer

```
 apt-get install python-pygraphviz python-kiwi python-
pygoocanvas libgoocanvas-dev
```

- Support for openflow module (requires some boost libraries)

```
apt-get install libboost-signals-dev libboost-filesystem-
dev
```

- Support for MPI-based distributed emulation

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

2.2 Downloading NS3 Using a Tarball:

The process for downloading ns-3 via tarball is a simple process, you just have to pick a release, download it and decompress it.

```
cd
 mkdir tarballs
 cd tarballs
 tar xjf ns-allinone-3.24.tar.bz2
```

If you change into the directory ns-allinone-3.24 you should see a number of files:

```
build.py*      ns-3.24/    pybindgen-0.15.0.795/  util.py
constants.py   nsc-0.5.2/  README
```

You are now ready to build the ns-3 distribution.

2.3 Building NS3 with build.py:

The first time you build the ns-3 project you should build using the allinone environment. This will get the project configured for you in the most commonly useful way.

Change into the directory you created in the download section above. You should have a directory called something like ns-allinone-3.24 under your ~/tarballs directory. Type the following:

```
./build.py
```

You will see lots of typical compiler output messages displayed as the build script builds the various pieces you downloaded. Eventually you should see the following magic words:

```
Build finished successfully (00:02:37)
Leaving directory `./ns-3-dev'
```

Once the project has built you typically will not use ns-3-allinone scripts. You will now interact directly with Waf and we do it in the ns-3-dev directory and not in the ns-3-allinone directory.

3- Configuring Click and NS3:

Check that you have the most recent version of Click modular router (If you already have an older version of Click from GitHub, you can get the latest version by running `git pull` in the click directory).

To configure Click with NS3 support you must first add Blackadder Elements in Click's Elements (or create a link). Click packages are not supported when running in NS3 mode.

- Create a link for Blackadder source file in click elements directory

  ```
  ln -s <BLACKADDER PREFIX>/src <CLICK
  PREFIX>/elements/blackadder
  ```

- Compile and Install Click by running:

```
./configure --disable-linuxmodule --enable-nsclick --
enable-blackadder
make
sudo make install
```

- Copy the Blackadder model and Blackadder examples folders from <BLACKADDER PREFIX>/ns3/ns3.X into the appropriate corresponding locations in the NS3 installation directory as follows:

```
cp -r <BLACKADDER PREFIX>/ns3/ns3.X/blackadder-model <NS3
PREFIX>/ns-3.X/src/blackadder
```

```
cp -r <BLACKADDER PREFIX>/ns3/ns3.X/blackadder-examples
<NS3 PREFIX>/ns-3.X/examples/blackadder
```

- Now change directory to the newly created blackadder model directory inside NS3 :

```
cd <NS3 PREFIX>/ns-3.X/src/blackadder/model
```

and do

```
make igraph_version.h
```

This will read the igraph version used and make the appropriate changes in the blackadder model code. Do make clean if igraph is upgraded to a new version.

- Now you need to configure and build NS3 with click and the copied Blackadder folders included.

```
cd < NS3 PREFIX >/ns-3.X/
```

```
./waf configure --with-nsclick=/path/to/click/source --
enable-examples
```

After the configuration step is finished you should see a summary of optional NS3 features that includes several entries, watch out for the following:

If "NS-3 Click Integration" indicates "not enabled" then there is probably a problem with the Click modular router installation. Check that you have the latest Click version and I would suggest repeating the Click configuration and compilation steps and looking out for any error messages during the process.

Otherwise "NS-3 Click Integration" should indicate "enabled"

And "Build examples" should also indicate "enabled"

Then proceed to the build process:

```
            ./waf build
```

4- Configuring an NS3 Network:

An NS3 network is defined in the configuration file as follows:

```
network ={
        nodes =(
               { ….node1
               },
               { …node2
               }
               );
        };
```

A configuration file can currently store a single network. The above mentioned global parameters are valid for the whole network (including all nodes and all connections).

4.1 Configuring an NS3 Network Node:

A network node is defined in the configuration file as follows:

```
{
    label = "";
    role = ["",""];
    connections = (
    {
         … connection 1
     },
      {
          … connection 2
      }
      );
      applications = (
      {
         … application 1
      },
      {
         … application 2
      }
      );
 }
```

label: The Label of that network node. The label is used when sending requests to the Rendezvous Node. The Topology Manager also keeps track of the nodes in the network using their labels. The size of the label must be BLACKADDER_ID_LENGTH bytes.

role: if omitted or role[ ] then the network node has no special functionality. Use role [“RV”,”TM”] if the node is the Rendezvous Node and the Topology Manager or use the above keywords separately to place the (extra) functionalities to different nodes.

4.2 Configuring an NS3 Network Connection:

A network connection is always unidirectional and is defined within the context of a network node as follows:

```
  {
    to = "00000002";
    Mtu = 1500;
    DataRate = "100Mbps";
    Delay = "10ms";
 }
```

to: the destination node (its label)

Mtu: The simulated MTU of this link

DataRate: The simulated Data Rate of this link

Delay: The simulated propagation delay of this link

4.3 Configuring an NS3 Application:

This block defines a set of NS3 applications that will be simulated as running in this node. Note that the name of the application must be the same as the C++ class defining the application in NS3.

```
{
  Name = "Subscriber";
  Start = "2.34";
  stop = "14.87";
}
```

name: The name of the NS3 application (same as the C++ definition)

start: The time in seconds when this application will start

stop: The time in seconds when this application will stop

A sample configuration file that includes all the settings above named (ns3_topology1.cfg) exists in:

```
<BLACKADDER_PREFIX>/ ns3/ns3.X/blackadder--model/model
```

5- Running an NS3 Simulation:

To run an NS3 simulation you have to first deploy a simulated network. A configuration file that describes the simulated network topology must be created. This file describes all network nodes and connections as well as the applications that will run in each node. Let's use the sample configuration file mentioned earlier and deploy by running:

```
./deploy -c ../ns3/ns3.X/blackadder
model/model/ns3_topology1.cfg -s
```

By using -s (--simulate) flag, the deployment tool will create all necessary click configuration files, the topology file and NS3 simulation (C++) code.

This will create these files in /tmp/:
00000001.conf, 00000002.conf, 00000003.conf, 00000004.conf,
00000005.conf, 00000006.conf, 00000007.conf, 00000008.conf
topology.graphml, topology.cpp

Leave all of them there except topology.cpp which contains the NS3 code necessary to run the sample configuration. Therefore you will need to copy topology.cpp into the appropriate location in the NS3 directory (<NS3 PREFIX>/ns-3.X/examples/blackadder/). It will be called examples/blackadder/example3 in NS3.

```
cp /tmp/topology.cpp <NS3 PREFIX>/ns-
3.X/examples/blackadder/example3.cc
```

Now edit <NS3 PREFIX>/ns-3.X/examples/blackadder/wscript to add the example by appending the following lines:

```
obj = bld.create_ns3_program('example3', ['core', 'point-
to-point', 'blackadder', 'applications'])
obj.source = ['example3.cc', 'publisher.cc' ,
'subscriber.cc',]
```

Now build NS3:

```
cd <NS3_PREFIX>
```

```
./waf build
```

And finally you can run the example simulation with

```
./waf --run examples/blackadder/example3
```

6- Writing an NS3 Application:

Publish/Subscribe applications running on top of an NS3 simulated Blackadder deployment must extend the PubSubApplication Class. Since pub/sub applications are NS3 Objects, they should implement the GetTypeId(void) method, where they can define their own attributes (see NS3 attribute system). The four essential methods below must be implemented for any NS3 pub/sub application running on top of a simulated Blackadder deployment.

Note: Object and subclass DoStart has been renamed to DoInitialize in NS3.17 and later versions. Therefore, if you are working with NS3.15 you will need to use DoStart or if you are working with the latest version you will need to use DoInitialize.

DoInitialize(void): This is a good place to define the Event Listener Callback method that will be called when a Blackadder is pushed to the application by the simulator (see Publisher and Subscriber example applications). A call to PubSubApplication::DoInitialize(); must be also made so that some housekeeping is done in the father class.

DoDispose(void): This is a good place to deregister and free any resources acquired with the DoInitialize method. A call to PubSubApplication::DoDispose(); must be also made.

StartApplication(void): This method is called by the Simulator when the time has come for the application to start (defined in the deployment configuration file). The API exported by the PubSubApplication Class can be used to access the service model which is exported by Blackadder. NS3 events can be also scheduled.

StopApplication(): This method is called by the Simulator when the time has come for the application to stop (defined in the deployment configuration file). Any pending events must be cancelled. Blackadder is notified about the application exiting, by the underlying ServiceModel Class that hides such implementation details from a NS3 application.

Finally, an event handler must be defined and assigned to the m_cb variable of PubSubApplication object. This should happen in the DoInitialize() method call, like:

```
m_cb = MakeCallback(&ApplicationClassName::EventHandler,
this);
```

The event handler's signature must be as following:

```
Void ApplicationClassName::EventHandler(Ptr<Event> ev);
```

It should usually look like this:

```
Void ApplicationClassName::EventHandler(Ptr<Event> ev) {
   Switch (ev->type) {
     case SCOPE_PUBLISHED:
     //do something – maybe now or schedule an NS3 event
     break;
     case SCOPE_UNPUBLISHED:
     //do something – maybe now or schedule an NS3 event
     break;
     case START_PUBLISH:
     //do something – maybe now or schedule an NS3 event
     break;
     case STOP_PUBLISH:
     //do something – maybe now or schedule an NS3 event
     break;
     case PUBLISHED_DATA:
     //do something – maybe now or schedule an NS3 event
     break;
   }
}
```