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

Introduction:

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. Later versions of NS3 and/or Linux operating system may still work, but have not been tested. 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 below according to the version of NS3 used on your machine. Use the former path if you are using NS3.15 or below, and the later path if you are using a version above NS3.15.

Installation Steps:

1- 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
  ```

2- Now download and install NS3. I would recommend https://www.nsnam.org/wiki/Installation , it has a complete installation guide for NS3 on all dominant operating systems and it starts from installing all the pre-requisite libraries and applications that NS3's components require. After you have downloaded and compiled NS3, you need to do the following:

- 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 go to the newly created blackadder directory inside NS3 src directory:

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

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
```

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).

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.

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

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
```

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
```

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;
    }
}
```