LAB 0 (Introduction to NS-3)

Fatema Mirza

D7030E Advanced Wireless Networks

LAB 0 (Introduction to NS-3)

by

Fatema Mirza

Submission Deadline:	September 6, 2021	
Teachers:	Dr. Evgeny Osipov,	LULEÅ TEKNISKA UNIVERSITET
	Ahmed Afif Monrat,	LULEÅ TEKNISKA UNIVERSITET

Contents

1

Questions and Answers

1.1. What are alternatives for installing NS3 under different operating systems?

The minimal prerequisites for ns-3 include 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. However, for proper functioning of ns-3, (https://www.nsnam.org/wiki/Installation#Prerequisites) could be referred to see the full list of prerequisites under different operating systems.

1.1.1. Linux

Although the prerequisites may differ for different Linux based operating system, the installation process is the same for all. Installation can be done via:

1. Bake: Bake is a recently developed tool for easy installation, building and finding out the missing requirements in the environment which can be downloaded from gitlab. The specific path for bake should be added. Subsequently, bake can be utilized to inspect for missing packages, download build and install ns-3 and its modules.The relevant commands and troubleshooting options are available in the official installation guide.

2. Manual installation

   (a) Using Git: Create a repository and fetch copy of ns-3-allinone using git commands. Download the latest version of ns-3 (ns3.34) inside the ns-3-allinone directory. Configure with waf and build to complete the installation. The commands for downloading and configuring waf can be found in the official installation guide.

   (b) Using a Tarball: This is the recommended method of installation. Create a repository and fetch a copy of the latest release using the relevant commands and build to complete the installation.

1.1.2. MacOS

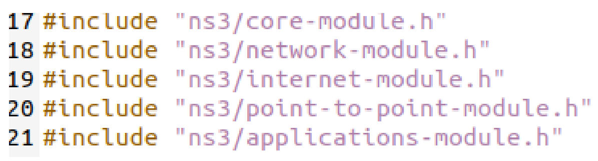
NS-3 can be installed on Catalina using the Xcode of Apple and clang/llvm compiler.

1.1.3. Windows

There are two main ways to install ns-3 in Windows, that is either install a virtual machine and create a linux instance or install Windows Subsystem for Linux and subsequently follow the steps for linux installation.

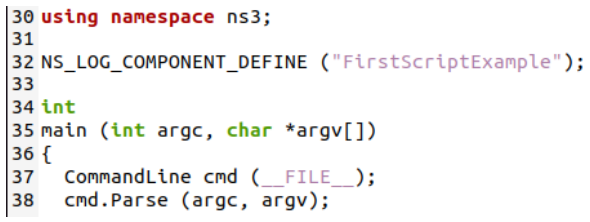
1.2. Write a step-by-step instruction for creating a simulation scenario, i.e. first we create ....

1. Import all relevant and required header files.



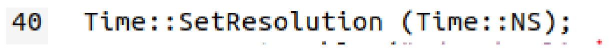
```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
```

2. Define a particular namespace for ns3 instance to avoid name clash issues. Use CommandLine class to enable argument passing during run time.



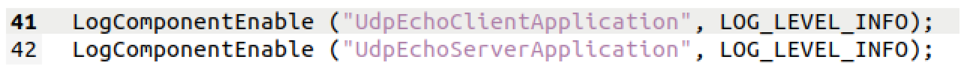
```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  CommandLine cmd (__FILE__);
  cmd.Parse (argc, argv);
```

3. Set simulation time.



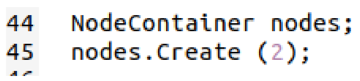
```
  Time::SetResolution (Time::NS);
```

4. Enable terminal logging.



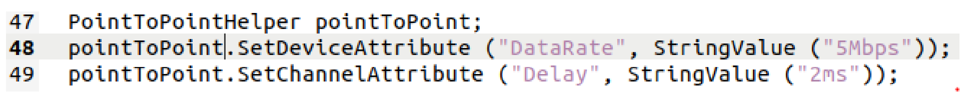
```
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

5. Create nodes (in this instance a server node and a client node).



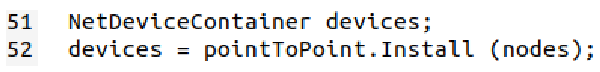
```
  NodeContainer nodes;
  nodes.Create (2);
```

6. Create channels with bandwidth and propagation delay (for this use case, point to point channel with data rate as 5 MBPS and propagation delay of 2 ms.)



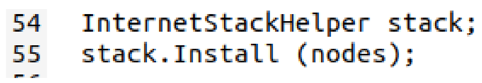
```
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

7. Configure the net devices to the channel.



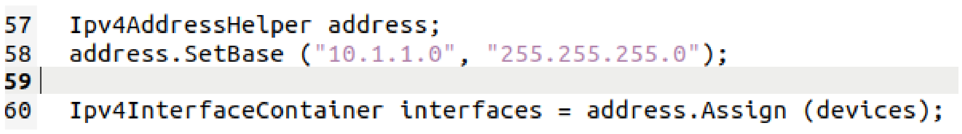
```
  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);
```

8. Install internet stack protocols to be used in nodes.



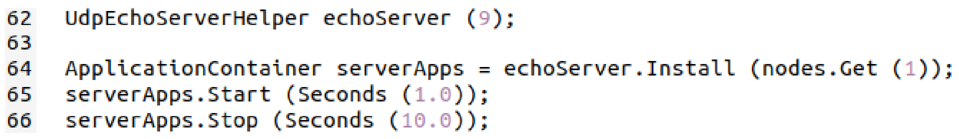
```
  InternetStackHelper stack;
  stack.Install (nodes);
```

9. Set the IP domain from the network 10.1.1.0 (base IP address so the first usable address will be 10.1.1.1) using the mask 255.255.255.0 and assign the IP addresses to the devices.



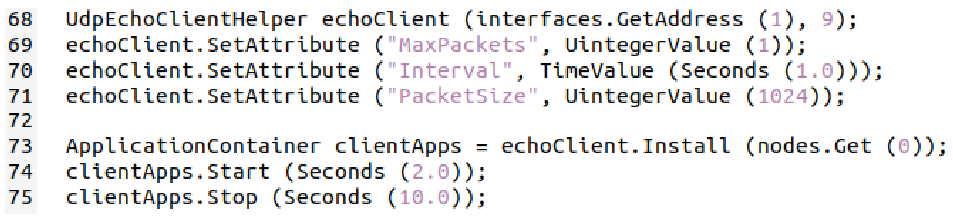
```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

10. Create UDP Server by assigning an IP address and port number. Specify for how long it will be active (atleast 10 seconds in this case).



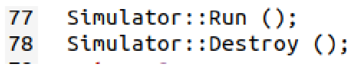
```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

11. Create UDP Client similarly to the server setting maximum packets, intervals and packet size attributes.



```
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

12. Begin simulation and destroy on completion.



```
Simulator::Run ();
Simulator::Destroy ();
```

1.3. If you would need to simulate a protocol which is not inside the NS3 library, what would you need to do?

To simulate a protocol which is not present or implemented inside the NS3 library, the protocol logic requires to be written in C++ and included in the library adhering to the rules of emacs formatting convention (for publishing). To call this subsequently for use in other scripts, it can be included in the header section. The protocol can also be added as a module and constructed resembling a module file structure in which source files, public header files, tests and examples are declared and following that configured and built. Python bindings are optional in this instance. Detailed step by step instruction is available at: https://www.nsnam.org/docs/manual/html/new-modules.html

1.4. For compiling an ns-3 executable a special build system is used. What is this system?

Waf is a python based build system used for compiling an ns-3 executable which makes certain that the shared library paths are available correctly and that the libraries can be accessed at run time.

1.5. Describe the purpose of different (only main) folders of the ns3 distribution.

The main folders are as follows:

1. bindings: contains the python scripts necessary for wrapping the ns-3.
2. build: contains the compiled scripts and libraries which are created once the build command is executed. On subsequent use, the compiled script are called from this folder.
3. doc: contains the user manual, documentation and further tutorials on NS-3.
4. examples: contains the default implementation of numerous networking protocols for ns-3.
5. scratch: contains the script files which are to be built and executed.
6. src folder contains the full source code of ns-3 implementation.
7. utils folder contains code coverage, test suites, style checking, and benchmarking related files.

1.6. Which folder should contain your simulation scripts?

All the simulation scripts must be placed in the **scratch** folder.

1.7. Write a step-by step instruction for executing an ns-3 simulation.

1. Ensure that the script to be executed is placed in the scratch folder.
2. From the root directory, run the following commands in the terminal:
   (a) For c++ files: ./waf - -run script_name
   To the previous command - - vis can be appended to view the script file graphically.
   (b) For python files: ./waf - -pyrun scratch/script_name.py

1.8. In how many formats does ns-3 saves the results (traces) of a simulation? Name them. What are the major differences?

NS-3 saves the result of the simulation traces in **two** formats, namely **ASCII tracing** with the extension format of script_name.tr and **PCAP tracing** with the extension format of script_name.pcap. The output of ASCII tracing contains a condensed textual format of various information - such as IP addresses, IP headers, Transport layer headers and so on. On the otherhand, PCAP tracing results in log files which can be viewed via tcpdump. PCAP tracing files can be analyzed and displayed using Wireshark whereas ASCII tracing files do not have this property.

1.9. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.

1.10. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.

1.11. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.

1.12. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.

1.13. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.

1.14. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.

1.15. When you run your simulation in which folder you will find the simulation traces.

The simulation file is created at the top-level directory of the repository which is this case the ns-3.34 directory unless explicitly changed with the - - cwd option of Waf to specify where to save the trace files.