SECTION 1 COMPUTER NETWORKS LAB

Structure **Page No.**

1.0 INTRODUCTION

This is the lab course, wherein you will have the hands on experience. You have studied the support course material (MCS-218 Data Communication and Computer Networks). In this section, Computer Network Programming using an Open Source platform, Network Simpulator-3 (NS-3) is provided illustratively. A list of programming problems is also provided at the end for each session. You are also provided some example codes in the end for your reference (Appendix). Please go through the general guidelines and the program documentation guidelines carefully.

1.1 OBJECTIVES

After completing this lab course you will be able to:

- Explore and understand the features available in the NS-3 software
- Create wired and wireless connection
- Test and verify various routing/ communication protocols
- Test and verify user defined routing protocols
- Perform analysis and evaluation of various available protocols

1.2 GENERAL GUIDELINES

- You should attempt all problems/assignments given in the list session wise.
- You may seek assistance in doing the lab exercises from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning a technical problem.
- For each program you should add comments above each function in the code, including the main function. This should also include a description of the function written.

- These descriptions should be placed in the comment block immediately above the relevant function source code.
- The comment block above the main function should describe the purpose of the program. Proper comments are to be provide where and when necessary in the programming.
- The output should also include clear network topology diagram.
- If two or more submissions from different students appear to be of the same origin (i.e. are variants of essentially the same program), none of them will be counted. You are strongly advised not to copy somebody else's work.
- It is your responsibility to create a separate directory to store all the programs, so that nobody else can read or copy.
- Observation book and Lab record are compulsory
- The list of the programs (list of programs given at the end, session-wise) is available to you in this lab manual. For each session, you must come prepare with the algorithms and the programs written in the Observation Book. You should utilize the lab hours for executing the programs, testing for various desired outputs and enhancements of the programs.
- As soon as you have finished a lab exercise, contact one of the lab instructor / in-charge in order to get the exercise evaluated and also get the signature from him/her on the Observation book.
- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.
- The total no. of lab sessions (3 hours each) are 10 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

1.3 INTRODUCTION TO NS-3

The network simulator 2 or 3 is basically tools to design the network topologies of users' desire and test them. As a user, you can design various network designs and simulate them using the NS2/NS3. Various network simulators are available on the internet few are using GUI while others are purely command based. To make it more user friendly one of the open source version of the network simulator (also available in GUI form) i.e. GNS3 is used to show real like devices to realize the devices.

GNS3 is a very powerful tool for simulating network and its behaviour. As a learner you can design and extend your own designs to virtual topologies and real world networks. Using GNS-3 you can interact with your own computer network and analyse various aspects of it. This is an open source network

simulator known as GNS-3 (Graphical Network Simulator-3). This simulator uses an emulator which runs various CISCO based networking device images. This image includes switches, routers, bridges and other networking devices. The emulator is called Dynamips and helps in running standard IOS images. There are few limitations of Dynamips as this is just an emulator and cannot replace the real hardware products.

The Dynamips emulator is accessed and run with the help of a text based front end known as Dynagen. This helps in interacting with Dynamips using a OOP application programming interface (API). These APIs are used to communicate and perform various tasks in Dynamips like instance reload, reboot, loading devices etc.

However, the core NS-3 is a C++ based library and can be used using scripting also. The scripting also helps in working with the standard NS-3 tracefile format i.e. .pcap files for various applications. The new features or modules in NS-3 are implemented and complied in C++ and merged into the system. In our case the basic NS-3 distribution is more than enough to work upon and therefore we will be limiting ourselves to this only. We recommend you to install "*allinone*" distribution for simplicity in installation and can be installed with a simple *install* script.

Prerequisites: To run NS-3 you should have C++ compiler (g++ or other), Python (3.6 or above), CMake and a build system (make, xcode etc.).

1.4 BASIC FEATURES OF NS-3

There are multiple network simulation tools existing on the Internet. The outstanding features of the NS-3 are following:

1. NS-3 is modular in nature and can be combined with external libraries.
2. NS-3 can be used on Linux, MacOs, BSD and Windows (using linux virtual machines).
3. NS-3 is an open source software product and project strive to remain open source and provide researchers and developers to contribute to the project.

As mentioned this is an open source project and hence you are not required to pay any amount to use or to implement this library at your own. It is dependent on the research and community to be maintained and further developed.

1.5 INSTALLING NS-3

Now, Let's talk about the download and installation of the network simulator software. The NS-3 is managed as a project by NS-3 consortium and available at the following link: www.nsnam.org.

You can download NS-3 freely from NS-3 website and following the installation guidelines, can easily install on different OS platforms like Mac OS, Linux and Windows. Originally, the NS-3 is based on the Linux environment, but if you have Windows environment, you can use cygwin to create a running environment for Linux based applications on windows as this provides the necessary environment for such applications. You can also use other hypervisor tools to create Linux environment in Windows and then install NS-3.

For using NS-3 all-in-one installation file GCC and other dependencies must be installed on the system. These dependencies can be found on the NS-3 official website.

Prerequisite	**Package/version**
C++ compiler	clang++ or g++ (g++ version 8 or greater)
Python	python3 version >=3.6
CMake	cmake version >=3.10
Build system	make, ninja, xcodebuild (XCode)
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)

Following are the steps involved in installation of NS-3

1. Download and unzip tar file from official website.
2. Run the *build.py* script file as

   ./buid.py --enable-examples --enable-tests

3. Check for list of Modules built and Modules not built.
4. After successfully build operation you should see different files and directories like source directory *src,* execution script *waf* and one script directory *scratch.* It should also contain the *examples* and *tutorials* files.

(A detailed description is provided on this link for installation of NS-3 on Windows https://www.nsnam.org/wiki/Installation#Windows).

At first you will see building of netanim animator build then pybindgen and finally NS-3. Now, NS-3 is installed and ready to be used you can run few examples on the system and try to learn the working of the system. Before running actual program you should run test programs to verify the running environment. To run the test programs you can run ./test.py --no-build command see if the systems pass all the tests. Once it passes all tests, you are ready to run your own code.

1.6 EXAMPLES

Let us now try to run an example on the recently installed system. Create a file with any name and save it with an extension as *.cc* or you can just go to examples/tutorials directory and see *first.cc* example file.

To run the first example, copy the *first.cc* file to scratch directory and run the following commands in the NS-2 terminal (cygwin) from the parent directory

./waf –run first

If nothing goes wrong, the program is compiled and run.

Argument Passing: NS-3 also supports the command line options and can pass the arguments to the program using command line. The following example with file *second.cc* program shows the argument value 3 given to *nCsma* variable.

./waf --run "second --nCsma=3"

Error Compilation: NS-3 by default treat the warnings as errors (enabling –Werror option) which is a good sign for professional but a little more exercise for beginners. You can disable this option by editing the *waf-tools/cflags.py* and change the following line:

self.warnings_flags = [['-Wall'], ['-Werror'], ['-Wextra']]

to the following line

self.warnings_flags = [['-Wall'], ['-Wextra']]

Now, you need to run the following commands.

./waf configure
./waf build

Once you are aware about few of the basic commands, you can see the examples given in the documentation of the NS-3. The link for the documentation is given below. A sample program is provided at the end of this file for tracing packets.

https://www.nsnam.org/doxygen/tcp-bbr-example_8cc_source.html

1.7 TERMS AND CONCEPTS

Before we proceed further, let's understand and map few terms to the actual networking concepts with the NS-3 environment to simplify the coding.

Node: A basic computing device is termed as a node (computer also) in NS-3. This is also abstracted as node class in NS-3. It also has methods for its management and representations of the device.

Application: The software with interacts with the operating systems and does the specified job is termed as an application. NS-3 refers to application as a user program which generates an activity which is further simulated. The Application class provides various associated methods to managing the user level applications.

Channel: Channels are basically the media of communication; here channel class manages the communication subnetwork objects and connecting nodes in the network. The three different channels used by us are CsmaChannel, PointToPointChannel and WifiChannel.

Net Device: A net device is usually an abstraction of both simulated hardware and software driver. After installation, the net device enables the node to communicate with other nodes using channels (in simulations).

Figure 1, shows the basic data flow model in simulator using ns-3. Applications talk with protocol stalks using APIs which are then used to communicate using NetDevices via channels.

There are many other terms and concepts which you can refer online on at the following link:

https://www.nsnam.org/docs/release/3.35/tutorial/html/conceptual-overview.html#key-abstractions

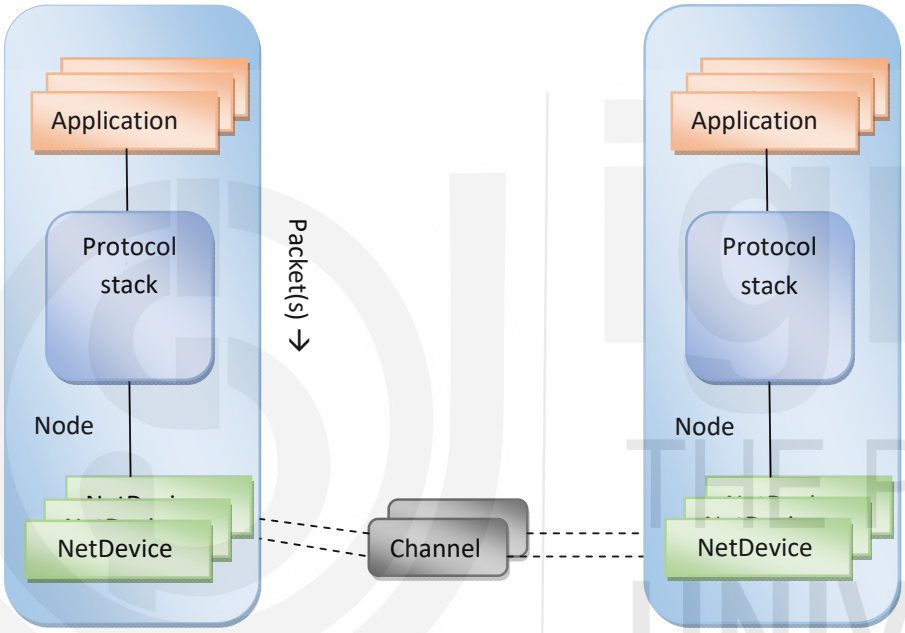


Figure 1: Basic data flow model in NS-3

1.8 LIST OF LAB ASSIGNMENTS – SESSION WISE

Let us try solving a different kind of a problem now. See the following given problems and try to implement these with the help of NS-3 simulator.

Session-1

1. Create a simple point to point network topology using two nodes.
2. Create a UdpClient and UdpServer nodes and communicate at a fixed data rate.

Session - 2

3. Measure the throughput (end to end) while varying latency in the network created in Session -1.

4. Create a simple network topology having two client node on left side and two server nodes on the right side. Both clients are connected with another node n1. Similarly both server node connecting to node n2. Also connect node n1 and n2 thus forming a dumbbell shape topology. Use point to point link only.

Session - 3

5. Install a TCP socket instance connecting either of the client node with either of the server node in session 2's network topology.

6. Install a TCP socket instance connecting other remaining client node with the remaining server node in session 2's network topology.

7. Start TCP application and monitor the packet flow.

Session - 4

8. Take three nodes n1, n2 and n3 and create a wireless mobile ad-hoc network.
9. Install the optimized Link State Routing protocol on these nodes.

Session – 5

10. Create a UDP client on a node n1 and a UDP server on a node n2.
11. Send packets to node n2 from node n1 and plot the number of bytes received with respect to time at node n2.
12. Show the pcap traces at node n2's WiFi interface.

Session – 6

13. Use 2 nodes to setup a wireless ad-hoc network where nodes are placed at a fixed distance in a 3D plane.

14. Install UDP server and Client at these two nodes.

15. Setup a CBR transmission between these nodes.

Session – 7

16. Setup 4 nodes, two TCP client and server pair and two UDP client and server pair.
17. Send packets to respective clients from both the servers.
18. Monitor the traffic for both the pair and plot the no. of bytes received.

Session – 8

19. Use the setup made in session 2 and monitor the traffic flow, plot the packets received.
20. Start the TCP application at Time 1 second.
21. After 20 seconds, start the UDP application at Rate1 which clogs the half of the dumbbell bridge capacity.

22. Using ns-3 tracing mechanism, plot the changes in the TCP window size over the time.

Session – 9

23. In the last session 8, Increase the UDP rate at 30 second to Rate2 such that it clogs whole of the dumbbell bridge capacity.
24. Use MatPlotlLib or GNUPlot to visualize cwnd vs time, also mention Rate1 and Rate2.

Session -10

25. Create a point to pint network between two nodes with the following parameters.

    Link bandwidth between the two nodes. Default is 5 Mbps.
    One way delay of the link. Default is 5 milliseconds.
    Loss rate of packets on the link. Default is 0.000001. (This covers losses other than those that occur due to buffer drops at node0.)
    Queue size of the buffer at node 0. Default is 10 packets.
    Simulation time. Default is 10 seconds.

    Calculate the average TCP throughput at the receiver using Wireshark application for packet capturing.

1.9 WEB REFERENCES

2. https://www.cse.iitb.ac.in/~mythili/teaching/cs224m_autumn2017/tcpsimpa/index.html
3. http://intronetworks.cs.luc.edu/current/html/ns3.html
4. https://www.nsnam.org/doxygen
5. https://www.nsnam.org/doxygen/csma-bridge_8cc_source.html
6. https://www.nsnam.org/docs/release/3.35/tutorial/html/conceptual-overview.html#key-abstractions
7. https://www.nsnam.org/wiki/Installation#Windows
8. https://www.wireshark.org/docs/wsug_html_chunked/ChapterIO.html

Appendix

Sample programs available online for different applications using NS-3

CODE-1: Source- https://www.nsnam.org/doxygen/traceroute-example_8cc_source.html

```
/* Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; */
/*
* Copyright (c) 2019 Ritsumeikan University, Shiga, Japan
*
* This program is free software; you can redistribute it *and/or modify it under the terms of the GNU General Public *License version 2 as published by the Free Software *Foundation;
*
* This program is distributed in the hope that it will be *useful, but WITHOUT ANY WARRANTY; without even the implied *warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *PURPOSE. See the GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public *License along with this program; if not, write to the Free *Software Foundation, Inc., 59 Temple Place, Suite 330, *Boston, MA 02111-1307 USA
*
* Author: Alberto Gallegos Ramonet <ramonet@fc.ritsumei.ac.jp>
*
*
* TraceRoute application example using AODV routing protocol.
*
*
*/
#include "ns3/aodv-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/v4traceroute-helper.h"
#include <iostream>
#include <cmath>
using namespace ns3;
class TracerouteExample
{
public:
TracerouteExample ();
bool Configure (int argc, char **argv);
void Run ();
void Report (std::ostream & os);
private:
// parameters
uint32_t size;
double step;
double totalTime;
bool pcap;
```

```
bool printRoutes;
NodeContainer nodes;
NetDeviceContainer devices;
Ipv4InterfaceContainer interfaces;
private:
void CreateNodes ();
void CreateDevices ();
void InstallInternetStack ();
void InstallApplications ();
};
int main (int argc, char **argv)
{
TracerouteExample test;
if (!test.Configure (argc, argv))
{
NS_FATAL_ERROR ("Configuration failed. Aborted.");
}
test.Run ();
test.Report (std::cout);
return 0;
}
//-----------------------------------------------------------------------------
TracerouteExample::TracerouteExample ()
: size (10),
step (50),
totalTime (100),
pcap (false),
printRoutes (false)
{
}
bool
TracerouteExample::Configure (int argc, char **argv)
{
// Enable AODV logs by default. Comment this if too noisy
// LogComponentEnable("AodvRoutingProtocol", LOG_LEVEL_ALL);
SeedManager::SetSeed (12345);
CommandLine cmd (__FILE__);
cmd.AddValue ("pcap", "Write PCAP traces.", pcap);
cmd.AddValue ("printRoutes", "Print routing table dumps.", printRoutes);
cmd.AddValue ("size", "Number of nodes.", size);
cmd.AddValue ("time", "Simulation time, s.", totalTime);
cmd.AddValue ("step", "Grid step, m", step);
cmd.Parse (argc, argv);
return true;
}
void
TracerouteExample::Run ()
{
CreateNodes ();
CreateDevices ();
InstallInternetStack ();
InstallApplications ();
std::cout << "Starting simulation for " << totalTime << " s ...\n";
Simulator::Stop (Seconds (totalTime));
Simulator::Run ();
Simulator::Destroy ();
```

```
}
void
TracerouteExample::Report (std::ostream &)
{
}
void
TracerouteExample::CreateNodes ()
{
std::cout << "Creating " << (unsigned)size << " nodes " << step << " m apart.\n";
nodes.Create (size);
// Name nodes
for (uint32_t i = 0; i < size; ++i)
{
std::ostringstream os;
os << "node-" << i;
Names::Add (os.str (), nodes.Get (i));
}
// Create static grid
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
"MinX", DoubleValue (0.0),
"MinY", DoubleValue (0.0),
"DeltaX", DoubleValue (step),
"DeltaY", DoubleValue (0),
"GridWidth", UintegerValue (size),
"LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
}
void
TracerouteExample::CreateDevices ()
{
WifiMacHelper wifiMac;
wifiMac.SetType ("ns3::AdhocWifiMac");
YansWifiPhyHelper wifiPhy;
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
wifiPhy.SetChannel (wifiChannel.Create ());
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode",
StringValue ("OfdmRate6Mbps"), "RtsCtsThreshold", UintegerValue (0));
devices = wifi.Install (wifiPhy, wifiMac, nodes);
if (pcap)
{
wifiPhy.EnablePcapAll (std::string ("aodv"));
}
}
void
TracerouteExample::InstallInternetStack ()
{
AodvHelper aodv;
// you can configure AODV attributes here using aodv.Set(name, value)
InternetStackHelper stack;
stack.SetRoutingHelper (aodv); // has effect on the next Install ()
stack.Install (nodes);
Ipv4AddressHelper address;
address.SetBase ("10.0.0.0", "255.0.0.0");
```

```
interfaces = address.Assign (devices);
if (printRoutes)
{
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>
("aodv.routes", std::ios::out);
aodv.PrintRoutingTableAllAt (Seconds (8), routingStream);
}
}
void
TracerouteExample::InstallApplications ()
{
V4TraceRouteHelper traceroute (Ipv4Address ("10.0.0.10")); //size - 1
traceroute.SetAttribute ("Verbose", BooleanValue (true));
ApplicationContainer p = traceroute.Install (nodes.Get (0));
// Used when we wish to dump the traceroute results into a file
//Ptr<OutputStreamWrapper> printstrm = Create<OutputStreamWrapper> ("mytrace",
std::ios::out);
//traceroute.PrintTraceRouteAt(nodes.Get(0),printstrm);
p.Start (Seconds (0));
p.Stop (Seconds (totalTime) - Seconds (0.001));
}
```

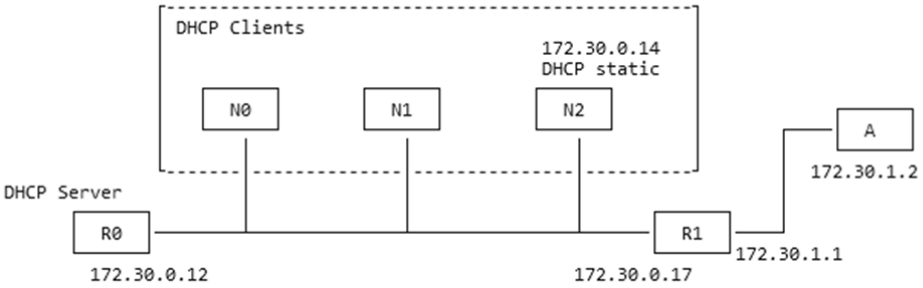
CODE-2: Source- https://www.nsnam.org/doxygen/dhcp-example_8cc_source.html

---

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
* Copyright (c) 2011 UPB
* Copyright (c) 2017 NITK Surathkal
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation;
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*
* Author: Radu Lupu <rlupu@elcom.pub.ro>
* Ankit Deepak <adadeepak8@gmail.com>
* Deepti Rajagopal <deeptir96@gmail.com>
*
*/
/*
```

```
Network layout:

R0 is a DHCP server. The DHCP server announced R1 as the default router.
Nodes N1 will send UDP Echo packets to node A.
```




```
Things to notice:
1) The routes in A are manually set to have R1 as the default router,
   just because using a dynamic outing in this example is an overkill.
2) R1's address is set statically though the DHCP server helper interface.
   This is useful to prevent address conflicts with the dynamic pool.
   Not necessary if the DHCP pool is not conflicting with static addresses.
3) N2 has a dynamically-assigned, static address (i.e., a fixed address assigned via DHCP).
```

```
*
*/
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("DhcpExample");
int
main (int argc, char *argv[])
{
CommandLine cmd (__FILE__);
bool verbose = false;
bool tracing = false;
cmd.AddValue ("verbose", "turn on the logs", verbose);
cmd.AddValue ("tracing", "turn on the tracing", tracing);
cmd.Parse (argc, argv);
// GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
if (verbose)
{
LogComponentEnable ("DhcpServer", LOG_LEVEL_ALL);
LogComponentEnable ("DhcpClient", LOG_LEVEL_ALL);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
}
Time stopTime = Seconds (20);
NS_LOG_INFO ("Create nodes.");
NodeContainer nodes;
NodeContainer router;
nodes.Create (3);
router.Create (2);
NodeContainer net (nodes, router);
NS_LOG_INFO ("Create channels.");
CsmaHelper csma;
```

```
csma.SetChannelAttribute ("DataRate", StringValue ("5Mbps"));
csma.SetChannelAttribute ("Delay", StringValue ("2ms"));
csma.SetDeviceAttribute ("Mtu", UintegerValue (1500));
NetDeviceContainer devNet = csma.Install (net);
NodeContainer p2pNodes;
p2pNodes.Add (net.Get (4));
p2pNodes.Create (1);
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
InternetStackHelper tcpip;
tcpip.Install (nodes);
tcpip.Install (router);
tcpip.Install (p2pNodes.Get (1));
Ipv4AddressHelper address;
address.SetBase ("172.30.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
// manually add a routing entry because we don't want to add a dynamic routing
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4> ipv4Ptr = p2pNodes.Get (1)->GetObject<Ipv4> ();
Ptr<Ipv4StaticRouting> staticRoutingA = ipv4RoutingHelper.GetStaticRouting (ipv4Ptr);
staticRoutingA->AddNetworkRouteTo (Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"),
Ipv4Address ("172.30.1.1"), 1);
NS_LOG_INFO ("Setup the IP addresses and create DHCP applications.");
DhcpHelper dhcpHelper;
// The router must have a fixed IP.
Ipv4InterfaceContainer fixedNodes = dhcpHelper.InstallFixedAddress (devNet.Get (4),
Ipv4Address ("172.30.0.17"), Ipv4Mask ("/24"));
// Not really necessary, IP forwarding is enabled by default in IPv4.
fixedNodes.Get (0).first->SetAttribute ("IpForward", BooleanValue (true));
// DHCP server
ApplicationContainer dhcpServerApp = dhcpHelper.InstallDhcpServer (devNet.Get (3),
Ipv4Address ("172.30.0.12"),
Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"),
Ipv4Address ("172.30.0.10"), Ipv4Address ("172.30.0.15"),
Ipv4Address ("172.30.0.17"));
// This is just to show how it can be done.
DynamicCast<DhcpServer> (dhcpServerApp.Get (0))->AddStaticDhcpEntry (devNet.Get
(2)->GetAddress (), Ipv4Address ("172.30.0.14"));
dhcpServerApp.Start (Seconds (0.0));
dhcpServerApp.Stop (stopTime);
// DHCP clients
NetDeviceContainer dhcpClientNetDevs;
dhcpClientNetDevs.Add (devNet.Get (0));
dhcpClientNetDevs.Add (devNet.Get (1));
dhcpClientNetDevs.Add (devNet.Get (2));
ApplicationContainer dhcpClients = dhcpHelper.InstallDhcpClient (dhcpClientNetDevs);
dhcpClients.Start (Seconds (1.0));
dhcpClients.Stop (stopTime);
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (p2pNodes.Get (1));
serverApps.Start (Seconds (0.0));
serverApps.Stop (stopTime);
```

```
UdpEchoClientHelper echoClient (p2pInterfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (100));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (1));
clientApps.Start (Seconds (10.0));
clientApps.Stop (stopTime);
Simulator::Stop (stopTime + Seconds (10.0));
if (tracing)
{
csma.EnablePcapAll ("dhcp-csma");
pointToPoint.EnablePcapAll ("dhcp-p2p");
}
NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```

CODE-3: Source- https://www.nsnam.org/doxygen/csma-bridge_8cc_source.html

---

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
* This program is free software; you can redistribute it *and/or modify it under the terms of the GNU General Public *License version 2 as  published by the Free Software *Foundation;
*
* This program is distributed in the hope that it will be *useful, but WITHOUT ANY WARRANTY; without even the implied *warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *PURPOSE. See the GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public *License along with this program; if not, write to the Free *Software Foundation, Inc., 59 Temple Place, Suite 330, *Boston, MA 02111-1307 USA
*/
// Network topology
//
// n0     n1
// |      |
// ----------
// | Switch |
// ----------
// |      |
// n2     n3
//
// - CBR/UDP flows from n0 to n1 and from n3 to n0
// - DropTail queues
// - Tracing of queues and packet receptions to file "csma-bridge.tr"
#include <iostream>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/bridge-module.h"
```

```
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("CsmaBridgeExample");
int
main (int argc, char *argv[])
{
//
// Users may find it convenient to turn on explicit debugging
// for selected modules; the below lines suggest how to do this
//
#if 0
LogComponentEnable ("CsmaBridgeExample", LOG_LEVEL_INFO);
#endif
//
// Allow the user to override any of the defaults and the above Bind() at
// run-time, via command-line arguments
//
CommandLine cmd (__FILE__);
cmd.Parse (argc, argv);
//
// Explicitly create the nodes required by the topology (shown above).
//
NS_LOG_INFO ("Create nodes.");
NodeContainer terminals;
terminals.Create (4);
NodeContainer csmaSwitch;
csmaSwitch.Create (1);
NS_LOG_INFO ("Build Topology");
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", DataRateValue (5000000));
csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
// Create the csma links, from each terminal to the switch
NetDeviceContainer terminalDevices;
NetDeviceContainer switchDevices;
for (int i = 0; i < 4; i++)
{
NetDeviceContainer link = csma.Install (NodeContainer (terminals.Get (i), csmaSwitch));
terminalDevices.Add (link.Get (0));
switchDevices.Add (link.Get (1));
}
// Create the bridge netdevice, which will do the packet switching
Ptr<Node> switchNode = csmaSwitch.Get (0);
BridgeHelper bridge;
bridge.Install (switchNode, switchDevices);
// Add internet stack to the terminals
InternetStackHelper internet;
internet.Install (terminals);
// We've got the "hardware" in place. Now we need to add IP addresses.
//
NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
ipv4.Assign (terminalDevices);
//
// Create an OnOff application to send UDP datagrams from node zero to node 1.
```

```
//
NS_LOG_INFO ("Create Applications.");
uint16_t port = 9; // Discard port (RFC 863)
OnOffHelper onoff ("ns3::UdpSocketFactory",
Address (InetSocketAddress (Ipv4Address ("10.1.1.2"), port)));
onoff.SetConstantRate (DataRate ("500kb/s"));
ApplicationContainer app = onoff.Install (terminals.Get (0));
// Start the application
app.Start (Seconds (1.0));
app.Stop (Seconds (10.0));
// Create an optional packet sink to receive these packets
PacketSinkHelper sink ("ns3::UdpSocketFactory",
Address (InetSocketAddress (Ipv4Address::GetAny (), port)));
app = sink.Install (terminals.Get (1));
app.Start (Seconds (0.0));
//
// Create a similar flow from n3 to n0, starting at time 1.1 seconds
//
onoff.SetAttribute ("Remote",
AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)));
app = onoff.Install (terminals.Get (3));
app.Start (Seconds (1.1));
app.Stop (Seconds (10.0));
app = sink.Install (terminals.Get (0));
app.Start (Seconds (0.0));
NS_LOG_INFO ("Configure Tracing.");
//
// Configure tracing of all enqueue, dequeue, and NetDevice receive events.
// Trace output will be sent to the file "csma-bridge.tr"
//
AsciiTraceHelper ascii;
csma.EnableAsciiAll (ascii.CreateFileStream ("csma-bridge.tr"));
//
// Also configure some tcpdump traces; each interface will be traced.
// The output files will be named:
// csma-bridge-<nodeId>-<interfaceId>.pcap
// and can be read by the "tcpdump -r" command (use "-tt" option to
// display timestamps correctly)
//
csma.EnablePcapAll ("csma-bridge", false);
//
// Now, do the actual simulation.
//
NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```