MANIPAL
ACADEMY of HIGHER EDUCATION
(Deemed to be University under Section 3 of the UGC Act, 1956)

DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL

CERTIFICATE

This is to certify that Ms./Mr. ……...…… Reg. No. …..…… Section:…… Roll No: …..…… has satisfactorily completed the lab exercises prescribed for **Network Simulation Lab [ICT 3269]** of Third Year **B. Tech. (CCE)** Degree at MIT, Manipal, in the academic year 2020 - 2021.

Date: …….....

Signature of the faculty

CONTENTS

Course Objectives

- ⊗ To learn the usage of network simulator for wired and wireless network topologies
- ⊗ To understand NetAnim tool
- ⊗ To learn the usage of SUMO/MOVE tool

Course Outcomes

At the end of this course, students will be able to

- ⊗ Implement wired and wireless network topologies using NS3.
- ⊗ Visualize network simulation using NetAnim tool.
- ⊗ Model intermodal traffic using SUMO/MOVE

Evaluation plan

Split up of 60 marks for Regular Lab Evaluation
Execution: 2 Marks Record: 4 Marks Evaluation: 4 Marks
End Semester Lab evaluation: 40 marks (Duration 2 hrs)
Program write up: 20 Marks Program execution: 20 Marks

Instructions to the students

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

1. Follow the instructions on the allotted exercises
2. Show the program and results to the instructors on completion of experiments
3. Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:

  - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
  - Comments should be used to give the statement of the problem.
  - The statements within the program should be properly indented.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- In case a student misses a lab, he/ she must ensure that the experiment is completed before the next evaluation with the permission of the faculty concerned.
- Students missing out the lab for genuine reasons like conference, sports or activities assigned by the Department or Institute will have to take **prior permission** from the HOD to attend **additional lab** (with another batch) and complete it **before** the student goes on leave. The student could be awarded marks for the write up for that day provided he submits it during the **immediate** next lab.
- Students who feel sick should get permission from the HOD for evaluating the lab records. However, attendance will not be given for that lab.
- Students will be evaluated only by the faculty with whom they are registered even though they carry out additional experiments in another batch.
- The presence of the student during the lab end semester exams is mandatory even if the student assumes he has scored enough to pass the examination
- Minimum attendance of 75% is mandatory to write the final exam.
- If the student loses his book, he/she will have to rewrite all the lab details in the lab record.
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

The students should NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

Introduction to Network Simulator -3

Objectives:

- Begin hands on experimentation with NS-3
- Learn node creation, link setup and queueing.

Brief Bio about NS-3

Network simulator is a tool used for simulating the real world network on one computer by writing scripts in C++ or Python. Normally, if we want to perform experiments, to see how our network works using various parameters, we don't have required number of computers and routers for making different topologies. Even if we have these resources it is very expensive to build such a network for experiment purposes. So to overcome these drawbacks we use simulators like NS-3, NS-2 etc.

NS-3 helps to create various virtual nodes (i.e., computers in real life) and with the help of various Helper classes it allows us to install devices, internet stacks, application, etc. to our nodes. Using NS-3 we can create PointToPoint, Wireless, CSMA, etc. connections between nodes. PointToPoint connection is same as a LAN connected between two computers. Wireless connection is same as WiFi connection between various computers and routers. CSMA connection is same as bus topology between computers. After building connections we try to install NIC to every node to enable network connectivity. When network cards are enabled in the devices, we add different parameters in the channels (i.e., real world path used to send data) which are data-rate, packet size, etc. Now we use Application to generate traffic and send the packets using these applications.

So to summarize NS-3 is

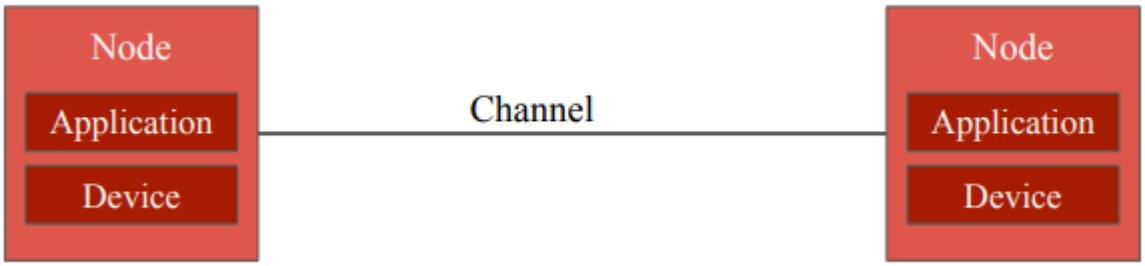
- Discrete event network simulator
- Open Source
- Collection of C++ libraries, not a program
- Supported under Linux, FreeBSD and Cygwin

Ns-3 gives us special features which can be used for real life integrations. Some of these features are:

- Tracing of the nodes:
  NS3 allows us to trace the routes of the nodes which helps us to know how much data is send or received. Trace files are generated to monitor these activities.
- NetAnim:
  It stands for Network Animator. It is an animated version of how network will look in real and how data will be transferred from one node to other.
- Pcap file:
  NS3 helps to generate pcap file which can be used to get all information of the packets (e.g., Sequence number, Source IP, destination IP, etc). These pcaps can be seen using a software tool known as wireshark.
- gnuPlot:
  GnuPlot is used to plot graphs from the data which we get from trace file of NS3. Gnuplot gives more accurate graph compare to other graph making tools and also it is less complex than other tools.

This is a brief introduction to NS-3. Basically NS-3 can perform most of the activities which are performed in the network in reality.

Conceptual overview
Key Abstractions



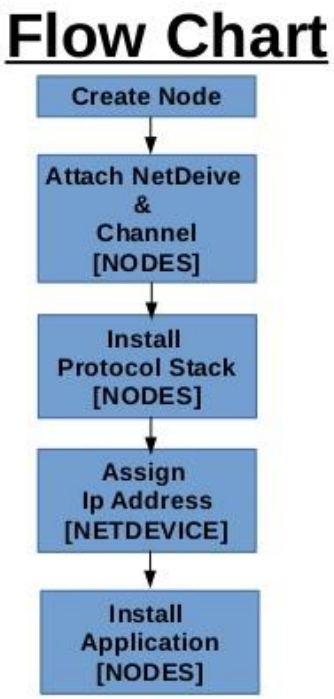
- Node
  - In Internet jargon, a computing device that connects to a network is called a host or sometimes an end system. In ns-3 the basic computing device abstraction is called the node. This abstraction is represented in C++ by the class Node. The Node class provides methods for managing the representations of computing devices in simulations.
- Application
  - Typically, computer software is divided into two broad classes. System Software organizes various computer resources such as memory, processor cycles, disk, network, etc., according to some computing model. System software usually does not use those resources to complete tasks that directly benefit a user. A user would typically run an application that acquires and uses the resources controlled by the system software to accomplish some goal.
- Channel
  - In the real world, one can connect a computer to a network. Often the media over which data flows in these networks are called channels. When you connect your Ethernet cable to the plug in the wall, you are connecting your computer to an Ethernet communication channel. In the simulated world of ns-3, one connects a Node to an object representing a communication channel. Here the basic communication subnetwork abstraction is called the channel and is represented in C++ by the class Channel.
  - The Channel class provides methods for managing communication subnetwork objects and connecting nodes to them. Channels may also be specialized by developers in the object oriented programming sense. A Channel specialization may model something as simple as a wire. The specialized Channel can also model things as complicated as a large Ethernet switch, or three-dimensional space full of obstructions in the case of wireless networks.
- Net Device
  - It used to be the case that if you wanted to connect a computers to a network, you had to buy a specific kind of network cable and a hardware device called (in PC terminology) a peripheral card that needed to be installed in your computer. If the peripheral card implemented some networking function, they were called Network Interface Cards, or NICs. Today most computers come with the network interface hardware built in and users don’t see these building blocks.
  - A NIC will not work without a software driver to control the hardware. In Unix (or Linux), a piece of peripheral hardware is classified as a device. Devices are controlled

using device drivers, and network devices (NICs) are controlled using network device drivers collectively known as net devices. In Unix and Linux you refer to these net devices by names such as eth0.

- In ns-3 the net device abstraction covers both the software driver and the simulated hardware. A net device is “installed” in a Node in order to enable the Node to communicate with other Nodes in the simulation via Channels. Just as in a real computer, a Node may be connected to more than one Channel via multiple NetDevices.

- Topology Helpers
  - In a real network, you will find host computers with added (or built-in) NICs. In ns-3 we would say that you will find Nodes with attached NetDevices. In a large simulated network you will need to arrange many connections between Nodes, NetDevices and Channels.
  - Since connecting NetDevices to Nodes, NetDevices to Channels, assigning IP addresses, etc., are such common tasks in ns-3, we provide what we call topology helpers to make this as easy as possible. For example, it may take many distinct ns-3 core operations to create a NetDevice, add a MAC address, install that net device on a Node, configure the node’s protocol stack, and then connect the NetDevice to a Channel. Even more operations would be required to connect multiple devices onto multipoint channels and then to connect individual networks together into internetworks.

Sample Code :



```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/* Emacs Code line conveys the formatting conventions */

/* Each of the ns-3 include files is placed in a directory called ns3 (under the build directory) during the build process to help avoid include file name collisions.*/
 #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"

 // Default Network Topology
 //
 //       10.1.1.0
 // n0 ---------- n1
 //    point-to-point
 //

 /* Namespace declaration- after this declaration, you will not have to type ns3:: scope resolution operator before all of the ns-3 code in order to use it.*/
 using namespace ns3;

 /*Similar to the define statement*/
 NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

 /*Main Program*/
 int
 main (int argc, char *argv[])
 {
   CommandLine cmd (__FILE__);
   cmd.Parse (argc, argv);
     Time::SetResolution (Time::NS);

/* These two lines of code enable debug logging at the INFO level for echo clients and servers. This will result in the application printing out messages as packets are sent and received during the simulation.*/
   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

 /* The NodeContainer topology helper provides a convenient way to create, manage and access any Node objects that we create in order to run a simulation. The first line above just declares a NodeContainer which we call nodes. The second line calls the Create method on the nodes object and asks the container to create two nodes.*/
   NodeContainer nodes;
   nodes.Create (2);

 /* Here a single PointToPointHelper is used to configure and connect ns-3 PointToPointNetDevice and PointToPointChannel objects in this script. The first line, instantiates a PointToPointHelper object
```

on the stack. From a high-level perspective the next line, tells the PointToPointHelper object to use the value “5Mbps” (five megabits per second) as the “DataRate” when it creates a PointToPointNetDevice object. The last line, tells the PointToPointHelper to use the value “2ms” (two milliseconds) as the value of the transmission delay of every point to point channel it subsequently creates.*/

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

/*We have a PointToPointHelper that is primed and ready to make PointToPointNetDevices and wire PointToPointChannel objects between them. The Install method of the PointToPointHelper takes a NodeContainer as a parameter. Internally, a NetDeviceContainer is created. For each node in the NodeContainer (there must be exactly two for a point-to-point link) a PointToPointNetDevice is created and saved in the device container. A PointToPointChannel is created and the two PointToPointNetDevices are attached. When objects are created by the PointToPointHelper, the Attributes previously set in the helper are used to initialize the corresponding Attributes in the created objects.

After executing the pointToPoint.Install (nodes) call we will have two nodes, each with an installed point-to-point net device and a single point-to-point channel between them. Both devices will be configured to transmit data at five megabits per second over the channel which has a two millisecond transmission delay. */

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

/* The InternetStackHelper is a topology helper that is to internet stacks what the PointToPointHelper is to point-to-point net devices. The Install method takes a NodeContainer as a parameter. When it is executed, it will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.*/

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

/* The lines below declare an address helper object and tell it that it should begin allocating IP addresses from the network 10.1.1.0 using the mask 255.255.255.0 to define the allocatable bits. By default the addresses allocated will start at one and increase monotonically, so the first address allocated from this base will be 10.1.1.1, followed by 10.1.1.2, etc. Subsequent line actually performs the address allotment*/

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

/* The first line of code in the above snippet declares the UdpEchoServerHelper with port number as parameter. echoServer.Install is going to install a UdpEchoServerApplication on the node found at index number one of the NodeContainer we used to manage our nodes. Applications require a time to “start” generating traffic and may take an optional time to “stop”. We provide both. These times are set using the ApplicationContainer methods Start and Stop. These methods take Time parameters in seconds.*/

```
UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
```

```
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

 /* Recall that we used an Ipv4InterfaceContainer to keep track of the IP addresses we assigned to our devices. The zeroth interface in the interfaces container is going to correspond to the IP address of the zeroth node in the nodes container. The first interface in the interfaces container corresponds to the IP address of the first node in the nodes container. So, in the first line of code (from above), we are creating the helper and telling it so set the remote address of the client to be the IP address assigned to the node on which the server resides. We also tell it to arrange to send packets to port nine.

The "MaxPackets" Attribute tells the client the maximum number of packets we allow it to send during the simulation. The "Interval" Attribute tells the client how long to wait between packets, and the "PacketSize" Attribute tells the client how large its packet payloads should be. With this particular combination of Attributes, we are telling the client to send one 1024-byte packet.

Just as in the case of the echo server, we tell the echo client to Start and Stop, but here we start the client one second after the server is enabled (at two seconds into the simulation).
*/
  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));
 /* When Simulator::Run is called, the system will begin looking through the list of scheduled events and executing them.
All that remains is to clean up. This is done by calling the global function Simulator::Destroy.*/
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
 }
```

Queing in NS-3

Queue is an abstract base class and is subclassed for specific scheduling and drop policies. Subclasses need to define the following public methods:

- bool Enqueue (Ptr<Item> item): Enqueue a packet
- Ptr<Item> Dequeue (void): Dequeue a packet
- Ptr<Item> Remove (void): Remove a packet
- Ptr<const Item> Peek (void): Peek a packet

```
/* DropTail :This is a basic first-in-first-out (FIFO) queue that performs a tail drop when the queue is full.
The DropTailQueue class defines one attribute: MaxSize: the maximum queue size*/
p2p.SetQueue ("ns3::DropTailQueue",
        "MaxSize", StringValue ("50p"));
p2p.SetDeviceAttribute ("DataRate", StringValue (linkDataRate));
p2p.SetChannelAttribute ("Delay", StringValue (linkDelay));
NetDeviceContainer devn2n3 = p2p.Install (n2n3);
```

Running the Script

```
1. Copy Files to :
ns-allinone-3.26/ns-3.26/scratch
2. From the ns-3.26 directory (Change to the directory “ns-3.25” $ cd ns-allinone-3.25/ns-3.25/ ) build the simulation with
./waf
3. Run the simulation using waf (remember to remove the .cc from the filename)
a. If no command line arguments are required
./waf --run scratch/scriptname
4. If command line arguments are required
./waf –-run “scratch/scriptname –-argument=value”
/*Example*/
/*If we were to provide a new DataRate using the command line, we could speed our simulation up again. We do this in the following way, according to the formula implied by the help item:*/
./waf --run "scratch/myfirst --ns3::PointToPointNetDevice::DataRate=5Mbps"
/* Let’s use the command line argument parser to take a look at the Attributes of the PointToPointNetDevice. It can be done by the following command */
./waf --run "scratch/myfirst --PrintAttributes=ns3::PointToPointNetDevice"
```

Lab Exercises:

1. Consider a source node, a destination node, and an intermediate router (respectively as shown in Fig.1.1). The link between nodes S and R (Link-1) has a bandwidth of 1Mbps and 50ms latency. The link between nodes R and D (Link-2) has a bandwidth of 100kbps and 5ms latency. Vary the Max Packet as 6, Interval as 1.0 and packet size as 1024. Show the results of the simulation on the console.

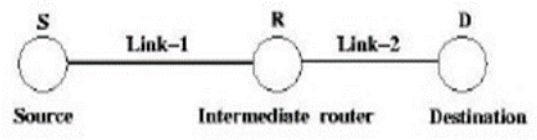


Fig. 1.1: Topology

2. Create a 4 node topology with the following configurations:
   - Between n0 and n2 : data rate is 10 Mbps, queue type is droptail, delay is 2ms.
   - Between n1 and n2 : data rate is 8 Mbps, queue type is droptail, delay is 3ms.
   - Between n2 and n3 : data rate is 10 Mbps, queue type is droptail, delay is 4ms, Maximum queue size as 3 packets.

     Show the results of the simulation on the console. Use command line arguments to vary the Max Packet as 16, Interval as 5.0 and packet size as 1024.

Observation Space

LAB NO: 2 **Date:**

Different Topologies and Introduction to NetAnim

Objectives:

- Begin hands on experimentation with NetAnim
- Learn how to configure various network topologies

What is Topology?

Topology defines the structure of the network of how all the components are interconnected to each other. There are two types of topology: physical and logical topology.

Common Physical topologies are the following:

- Bus :
  - The bus topology is designed in such a way that all the stations are connected through a single cable known as a backbone cable.
  - When a node wants to send a message over the network, it puts a message over the network. All the stations available in the network will receive the message whether it has been addressed or not.
  - The most common access method of the bus topologies is CSMA (Carrier Sense Multiple Access).
- Ring:
  - Ring topology is like a bus topology, but with connected ends.
  - The node that receives the message from the previous computer will retransmit to the next node.
  - The data flows in one direction, i.e., it is unidirectional.
  - The data flows in a single loop continuously known as an endless loop.
  - It has no terminated ends, i.e., each node is connected to other node and having no termination point.
  - The data in a ring topology flow in a clockwise direction.
  - The most common access method of the ring topology is token passing.
    - Token passing: It is a network access method in which token is passed from one node to another node.
    - Token: It is a frame that circulates around the network.
- Star:
  - Star topology is an arrangement of the network in which every node is connected to the central hub, switch or a central computer.
  - The central computer is known as a server, and the peripheral devices attached to the server are known as clients.
  - Hubs or Switches are mainly used as connection devices in a physical star topology.
- Mesh:
  - Mesh technology is an arrangement of the network in which computers are interconnected with each other through various redundant connections.
  - There are multiple paths from one computer to another computer.
  - It does not contain the switch, hub or any central computer which acts as a central point of communication.

- The Internet is an example of the mesh topology.
- Mesh topology is mainly used for WAN implementations where communication failures are a critical concern.
- Mesh topology is mainly used for wireless networks.
- Mesh topology can be formed by using the formula:
  - Number of cables = (n*(n-1))/2;
  - Where n is the number of nodes that represents the network.
- Mesh topology is divided into two categories:
  - Full Mesh Topology: In a full mesh topology, each computer is connected to all the computers available in the network.
  - Partial Mesh Topology: In a partial mesh topology, not all but certain computers are connected to those computers with which they communicate frequently.

- Tree Topology
  - Tree topology combines the characteristics of bus topology and star topology.
  - A tree topology is a type of structure in which all the computers are connected with each other in hierarchical fashion.
  - The top-most node in tree topology is known as a root node, and all other nodes are the descendants of the root node.
  - There is only one path exists between two nodes for the data transmission. Thus, it forms a parent-child hierarchy.
- Hybrid Topology
  - The combination of various different topologies is known as Hybrid topology.
  - A Hybrid topology is a connection between different links and nodes to transfer the data.
  - When two or more different topologies are combined together is termed as Hybrid topology and if similar topologies are connected with each other will not result in Hybrid topology. For example, if there exist a ring topology in one branch of ICICI bank and bus topology in another branch of ICICI bank, connecting these two topologies will result in Hybrid topology.

Code Snippets:

```
/*The topology below creates a 5 node hybrid topology involving point to point nodes and CSMA nodes.*/
/*The NodeContainer is used to create two nodes that we will connect via the point-to-point link.*/
NodeContainer p2pNodes;
p2pNodes.Create (2);
/*Next, we declare another NodeContainer to hold the nodes that will be part of the bus (CSMA) network. First, we just instantiate the container object itself.*/
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
/* The next line of code Gets the first node (as in having an index of one) from the point-to-point node container and adds it to the container of nodes that will get CSMA devices. The node in question is going to end up with a point-to-point device and a CSMA device. We then create a number of “extra” nodes that compose the remainder of the CSMA network. Since we already have one node in the
```

CSMA network – the one that will have both a point-to-point and CSMA net device, the number of "extra" nodes means the number nodes you desire in the CSMA section minus one.*/
uint32_t nCsma = 3;
/* The codes below set data rate and channel delay for point to point link*/
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
/* Then instantiate a NetDeviceContainer to keep track of the point-to-point net devices and we Install devices on the point-to-point nodes.
The CsmaHelper works just like a PointToPointHelper, but it creates and connects CSMA devices and channels. In the case of a CSMA device and channel pair, notice that the data rate is specified by a channel Attribute instead of a device Attribute. */
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
/*Just as we created a NetDeviceContainer to hold the devices created by the PointToPointHelper we create a NetDeviceContainer to hold the devices created by our CsmaHelper. We call the Install method of the CsmaHelper to install the devices into the nodes of the csmaNodes NodeContainer.*/
/*The InternetStackHelper is used to install these stacks.*/
InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);
/*Recall that we took one of the nodes from the p2pNodes container and added it to the csmaNodes container. Thus we only need to install the stacks on the remaining p2pNodes node, and all of the nodes in the csmaNodes container to cover all of the nodes in the simulation.
Using Ipv4AddressHelper to assign IP addresses to our device interfaces. First we use the network 10.1.1.0 to create the two addresses needed for our two point-to-point devices.*/
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
/*The CSMA devices will be associated with IP addresses from network number 10.1.2.0 in this case, as seen below.*/
address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
/*We are going to instantiate the server on one of the nodes that has a CSMA device and the client on the node having only a point-to-point device.
First, we set up the echo server. We create an UdpEchoServerHelper and provide a required Attribute value to the constructor which is the server port number. Recall that this port can be changed later using the SetAttribute method if desired, but we require it to be provided to the constructor.*/
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));

```
serverApps.Stop (Seconds (10.0));
/*Recall that the csmaNodes NodeContainer contains one of the nodes created for the point-to-point network and nCsma “extra” nodes. What we want to get at is the last of the “extra” nodes. The zeroth entry of the csmaNodes container will be the point-to-point node. */
/*The client application is set up exactly as we did in the first.cc example script. Again, we provide required Attributes to the UdpEchoClientHelper in the constructor (in this case the remote address and port). We tell the client to send packets to the server we just installed on the last of the “extra” CSMA nodes. We install the client on the leftmost point-to-point node seen in the topology illustration.*/
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

```
/*Snippet to build star topology*/
uint32_t nSpokes = 8;
NS_LOG_INFO ("Build star topology.");
PointToPointHelperpointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
PointToPointStarHelper star (nSpokes, pointToPoint);
NS_LOG_INFO ("Install internet stack on all nodes.");
InternetStackHelper internet;
star.InstallStack (internet);
NS_LOG_INFO ("Assign IP Addresses.");
star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));
NS_LOG_INFO ("Create applications.");
/* Create a packet sink on the star "hub" to receive packets.*/
uint16_t port = 50000;
Address hubLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress);
ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());
hubApp.Start (Seconds (1.0));
hubApp.Stop (Seconds (10.0));
/* Create OnOff applications to send TCP to the hub, one on each spoke node.*/
OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
onOffHelper.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
onOffHelper.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
ApplicationContainer spokeApps;
for (uint32_t i = 0; i < star.SpokeCount (); ++i)
{
AddressValue remoteAddress (InetSocketAddress (star.GetHubIpv4Address (i), port));
onOffHelper.SetAttribute ("Remote", remoteAddress);
spokeApps.Add (onOffHelper.Install (star.GetSpokeNode (i)));
}
```

```
spokeApps.Start (Seconds (1.0));
spokeApps.Stop (Seconds (10.0));
NS_LOG_INFO ("Enable static global routing.");
/* Turn on global static routing so we can actually be routed across the star.*/
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

NetAnim

Animation is an important tool for network simulation. While ns-3 does not contain a default graphical animation tool, we currently have two ways to provide animation, namely using the PyViz method or the NetAnim method.

The NetAnim GUI provides play, pause, and record buttons. Play and pause start and stop the simulation. The record button starts a series of screenshots of the animator, which are written to the directory in which the trace file was run. Two slider bars also exist. The top slider provides a "seek" functionality, which allows a user to skip to any moment in the simulation. The bottom slider changes the granularity of the time step for the animation. Finally, there is a quit button to stop the simulation and quit the animator.

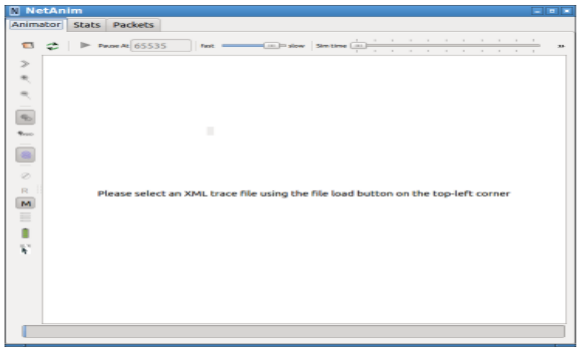
The class "AnimationInterface" under "src/netanim" uses underlying ns-3 trace sources to construct a timestamped ASCII file in XML format that can be read by a standalone animator named "NetAnim".

```
/*So you will have to make the following changes to the code, in order to view the animation on
NetAnim.*/
#include " ... "
#include "ns3/netanim−module .h" //1 Include. . .
int main ( int argc , char ∗argv [ ] )
{ std : : string animFile = "somename. xml"; //2 Name of file for animation
. . .
AnimationInterface anim (animFile ); //3 Animation interface
Simulator : : Run (); Simulator : : Destroy (); return 0;
}
```

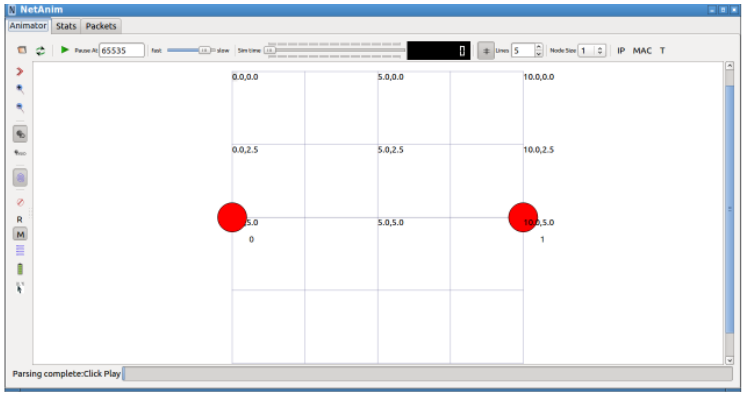
Example to run the code

1. Download the file "SimpleNS3Simulation2" and copy it to the folder ns-allinone-3.25/ns-3.25/scratch
2. Change to the directory "ns-3.25" :$ cd ns-allinone-3.25/ns-3.25/
3. And run the simulation: $ waf –run SimpleNS3Simulation2
4. After running the simulation, a XML trace file will be generated in the name "SimpleNS3Simulation_NetAnimationOutput.xml"
5. Now we have to run the NetAnim as follows (here we assume that the path to NetAnim binary is already set in the environment variable): $NetAnim

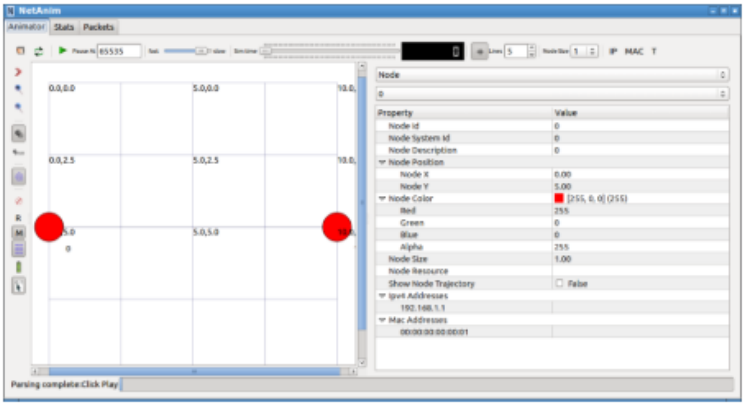
6. This command will open a blank NetAnim User Interface with a blank window



7. Now we have to open the "SimpleNS3Simulation_NetAnimationOutput.xml" file using the left-top folder icon on the toolbar.
8. This will show the nodes at positions that we set in the simulation script.



9. By using the options available in the NetAnim user interface, we can customize the properties of the Nodes and display some additional information. For example, by double-clicking on a node will open a property editor from which we can set some basic display property of a node.



Lab Exercises:

1. Configure the topology given below and visualize the same using NetAnim.

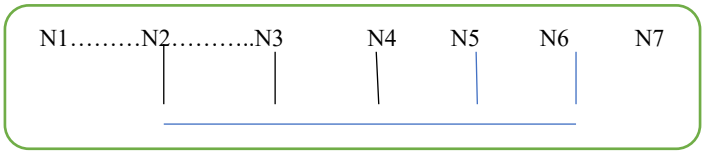


Fig. 2.1: Topology

2. Configure a star topology with 12 nodes and visualize the same using NetAnim.

Additional Exercises

1. Explore and configure the following topologies and visualize the same using NetAnim.
   i. Mesh
   ii. Dumbbell

OBSERVATION SPACE

LAB NO: 3 **Date:**

Tracing and Performance Analysis

Objectives:

- Learn to analyze the obtained results using PCAP and Wireshark
- Learn how to plot graphs using GnuPlot

Tracing Motivation

There are many ways to get information out of a program. The most straightforward way is to just directly print the information to the standard output, as in,

```
#include <iostream>
...
int main ()
{
  ...
  std::cout << "The value of x is " << x << std::endl;
  ...
}
```

This is workable in small environments, but as your simulations get more and more complicated, you end up with more and more prints and the task of parsing and performing computations on the output begins to get harder and harder.

Another thing to consider is that every time a new tidbit is needed, the software core must be edited and another print introduced. There is no standardized way to control all of this output, so the amount of output tends to grow without bounds. Eventually, the bandwidth required for simply outputting this information begins to limit the running time of the simulation. The output files grow to enormous sizes and parsing them becomes a problem.

ns-3 provides a simple mechanism for logging and providing some control over output via Log Components, but the level of control is not very fine grained at all.

It is desirable to have a facility that allows one to reach into the core system and only get the information required without having to change and recompile the core system.

The ns-3 tracing system is designed to work along those lines and is well-integrated with the Attribute and Config substems allowing for relatively simple use scenarios.

Using PCAP for tracing

Snippet

The first line of code to enable pcap tracing in the point-to-point helper

```
pointToPoint.EnablePcapAll ("second");
csma.EnablePcap ("second", csmaDevices.Get (1), true);
```

The CSMA network is a multi-point-to-point network. This means that there can (and are in this case) multiple endpoints on a shared medium. Each of these endpoints has a net device associated with it. There are two basic alternatives to gathering trace information from such a network. One way is to create a trace file for each net device and store only the packets that are emitted or consumed by that net device. Another way is to pick one of the devices and place it in promiscuous mode. That single

device then “sniffs” the network for all packets and stores them in a single pcap file. This is how tcpdump, for example, works. That final parameter tells the CSMA helper whether or not to arrange to capture packets in promiscuous mode.

In this example, we are going to select one of the devices on the CSMA network and ask it to perform a promiscuous sniff of the network, thereby emulating what tcpdump would do.

For the topology given below these pcap files will be created on execution

```
Default Network Topology

       10.1.1.0
n0 -------------- n1   n2   n3   n4
   point-to-point  |    |    |    |
                   ================
                     LAN 10.1.2.0
```

If you now go and look in the top level directory, you will find three trace files:

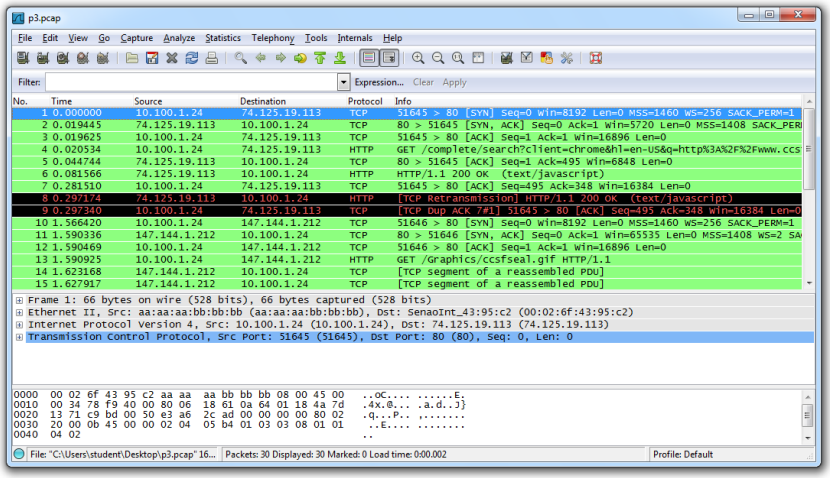
second-0-0.pcap second-1-0.pcap second-2-0.pcap

Let’s take a moment to look at the naming of these files. They all have the same form, <name>-<node>-<device>.pcap. For example, the first file in the listing is second-0-0.pcap which is the pcap trace from node zero, device zero. This is the point-to-point net device on node zero. The file second-1-0.pcap is the pcap trace for device zero on node one, also a point-to-point net device; and the file second-2-0.pcap is the pcap trace for device zero on node two.

Executing via TCP dump

tcpdump -nn -tt -r second-0-0.pcap

Alternate way is to open the pcap file using wireshark which gives the more detailed analysis



Plotting and Visualizing via GnuPlot

```
using namespace std;
string fileNameWithNoExtension = "plot-2d";
string graphicsFileName        = fileNameWithNoExtension + ".png";
string plotFileName            = fileNameWithNoExtension + ".plt";
string plotTitle               = "2-D Plot";
string dataTitle               = "2-D Data";
// Instantiate the plot and set its title.
Gnuplot plot (graphicsFileName);
plot.SetTitle (plotTitle);
// Make the graphics file, which the plot file will create when it
```

```
// is used with Gnuplot, be a PNG file.
plot.SetTerminal ("png");
// Set the labels for each axis.
plot.SetLegend ("X Values", "Y Values");
// Set the range for the x axis.
plot.AppendExtra ("set xrange [-6:+6]");
// Instantiate the dataset, set its title, and make the points be
// plotted along with connecting lines.
Gnuplot2dDataset dataset;
dataset.SetTitle (dataTitle);
dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);
double x;
double y;
// Create the 2-D dataset.
for (x = -5.0; x <= +5.0; x += 1.0)
  {
    // Calculate the 2-D curve
    y = x * x;
    // Add this point.
    dataset.Add (x, y);
  }
// Add the dataset to the plot.
plot.AddDataset (dataset);
// Open the plot file.
ofstream plotFile (plotFileName.c_str());
// Write the plot file.
plot.GenerateOutput (plotFile);
// Close the plot file.
plotFile.close ()
```

Alternatively, the following code also can be appended to generate plots

```
  // Use GnuplotHelper to plot the packet byte count over time
  GnuplotHelper plotHelper;
   // Configure the plot.  The first argument is the file name prefix
  // for the output files generated.  The second, third, and fourth
  // arguments are, respectively, the plot title, x-axis, and y-axis labels
  plotHelper.ConfigurePlot ("seventh-packet-byte-count",
                            "Packet Byte Count vs. Time",
                            "Time (Seconds)",
                            "Packet Byte Count");
   // Specify the probe type, trace source path (in configuration namespace), and
  // probe output trace source ("OutputBytes") to plot.  The fourth argument
 // specifies the name of the data series label on the plot.  The last
  // argument formats the plot by specifying where the key should be placed.
  plotHelper.PlotProbe (probeType,
                        tracePath,
                        "OutputBytes",
```

```
                "Packet Byte Count",
                GnuplotAggregator::KEY_BELOW);
 // Use FileHelper to write out the packet byte count over time
 FileHelper fileHelper;
  // Configure the file to be written, and the formatting of output data.
 fileHelper.ConfigureFile ("seventh-packet-byte-count",
                   FileAggregator::FORMATTED);
  // Set the labels for this formatted output file.
 fileHelper.Set2dFormat ("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");
  // Specify the probe type, trace source path (in configuration namespace), and
 // probe output trace source ("OutputBytes") to write.
 fileHelper.WriteProbe (probeType,
                 tracePath,
                 "OutputBytes");
```

Execution Steps

```
./waf --run src/stats/examples/gnuplot-example
This should produce the following gnuplot control files:
plot-2d.plt
In order to process these gnuplot control files, do the following:
$ gnuplot plot-2d.plt
This should produce the following graphics files:
plot-2d.png
You can view these graphics files in your favorite graphics viewer. If you have gimp installed on your
machine, for example, you can do this:
$ gimp plot-2d.png
```

Lab Exercises:

1. For the topologies given in lab 2, generate the pcap file and analyze the same using Wireshark. Calculate throughput, delay and packet delivery ratio and plot suitable graphs using GNUPLOT.

OBSERVATION SPACE

LAB NO: 4 **Date:**

Multicasting and Broadcasting

Objectives:

- Simulate the concepts of broadcasting and multicasting in various topologies.

Multicast Vs Broadcast Vs Unicast

- Multicasting refers to sending a message to a select group whereas broadcasting refers to sending a message to everyone connected to a network.
- Communication that takes place over a network between a single sender and a single receiver is termed as Unicast. Example Postal Service
- A simple example of multicasting is sending an e-mail message to a mailing list. Teleconferencing and videoconferencing also use multicasting.
- TV, Radio etc. use broadcast communication mode.

Multicasting Code Snippet

```
The snippet below performs the following
Network topology
  //
  //                  Lan1
  //            ===========
  //             |   |   |
  //   n0   n1   n2   n3   n4
  //    |   |   |
  // ===========
  //         Lan0
  //
  // - Multicast source is at node n0;
  // - Multicast forwarded by node n2 onto LAN1;
  // - Nodes n0, n1, n2, n3, and n4 receive the multicast frame.
  // - Node n4 listens for the data

NS_LOG_INFO ("Configure multicasting.");
    //
    // Now we can configure multicasting.  As described above, the multicast
    // source is at node zero, which we assigned the IP address of 10.1.1.1
    // earlier.  We need to define a multicast group to send packets to.  This
    // can be any multicast address from 224.0.0.0 through 239.255.255.255
    // (avoiding the reserved routing protocol addresses).
    //

   Ipv4Address multicastSource ("10.1.1.1");
   Ipv4Address multicastGroup ("225.1.2.4");

   // Now, we will set up multicast routing.  We need to do three things:
   // 1) Configure a (static) multicast route on node n2
```

```
// 2) Set up a default multicast route on the sender n0
// 3) Have node n4 join the multicast group
// We have a helper that can help us with static multicast
Ipv4StaticRoutingHelper multicast;

// 1) Configure a (static) multicast route on node n2 (multicastRouter)
Ptr<Node> multicastRouter = c.Get (2);  // The node in question
Ptr<NetDevice> inputIf = nd0.Get (2);  // The input NetDevice
NetDeviceContainer outputDevices;  // A container of output NetDevices
outputDevices.Add (nd1.Get (0));  // (we only need one NetDevice here)

multicast.AddMulticastRoute (multicastRouter, multicastSource,
                        multicastGroup, inputIf, outputDevices);

// 2) Set up a default multicast route on the sender n0
Ptr<Node> sender = c.Get (0);
Ptr<NetDevice> senderIf = nd0.Get (0);
multicast.SetDefaultMulticastRoute (sender, senderIf);

//
// Create an OnOff application to send UDP datagrams from node zero to the
// multicast group (node four will be listening).
//
NS_LOG_INFO ("Create Applications.");
 uint16_t multicastPort = 9;   // Discard port (RFC 863)

// Configure a multicast packet generator that generates a packet
// every few seconds
OnOffHelper onoff ("ns3::UdpSocketFactory",
                Address (InetSocketAddress (multicastGroup, multicastPort)));
onoff.SetConstantRate (DataRate ("255b/s"));
onoff.SetAttribute ("PacketSize", UintegerValue (128));

ApplicationContainer srcC = onoff.Install (c0.Get (0));
//
// Tell the application when to start and stop.
//
srcC.Start (Seconds (1.));
srcC.Stop (Seconds (10.));

// Create an optional packet sink to receive these packets
PacketSinkHelper sink ("ns3::UdpSocketFactory",
                  InetSocketAddress (Ipv4Address::GetAny (), multicastPort));

ApplicationContainer sinkC = sink.Install (c1.Get (2)); // Node n4
// Start the sink
sinkC.Start (Seconds (1.0));
sinkC.Stop (Seconds (10.0));
```

Broadcasting Code Snippet

```
NodeContainer c0 = NodeContainer (c.Get (0), c.Get (1));
 NodeContainer c1 = NodeContainer (c.Get (0), c.Get (2));
 NS_LOG_INFO ("Build Topology.");
 CsmaHelper csma;
 csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));
 csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
 NetDeviceContainer n0 = csma.Install (c0);
 NetDeviceContainer n1 = csma.Install (c1);
 InternetStackHelper internet;
 internet.Install (c);
 NS_LOG_INFO ("Assign IP Addresses.");
 Ipv4AddressHelper ipv4;
 ipv4.SetBase ("10.1.0.0", "255.255.255.0");
 ipv4.Assign (n0);
 ipv4.SetBase ("192.168.1.0", "255.255.255.0");
 ipv4.Assign (n1);
 // RFC 863 discard port ("9") indicates packet should be thrown away
 // by the system.  We allow this silent discard to be overridden
 // by the PacketSink application.
 uint16_t port = 9;
 // Create the OnOff application to send UDP datagrams of size
 // 512 bytes (default) at a rate of 500 Kb/s (default) from n0
 NS_LOG_INFO ("Create Applications.");
OnOffHelper onoff ("ns3::UdpSocketFactory",
                Address (InetSocketAddress (Ipv4Address ("255.255.255.255"), port)));
 onoff.SetConstantRate (DataRate ("500kb/s"));
 ApplicationContainer app = onoff.Install (c0.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 (c0.Get (1));
app.Add (sink.Install (c1.Get (1)));
app.Start (Seconds (1.0));
app.Stop (Seconds (10.0));
```

Lab Exercises:

1. For the topology given below simulate multicasting and broadcasting using UDP protocol.

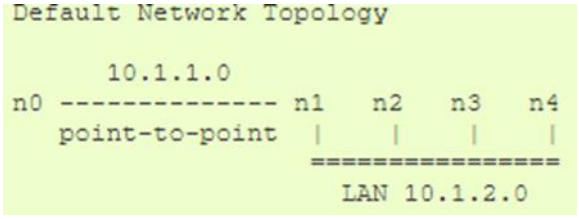


Fig. 4.1: Topology

2. Simulate the broadcasting with TCP for a star topology with 8 nodes.
3. Implement multicasting with UDP with the following configurations for the topology given below:
   - Multicast source is at node n0.
   - Multicast forwarded by node n2 onto LAN1.
   - Nodes n0, n1, n2, n3, n4 and n5 receive the multicast frame.
   - Node n5 listens for the data

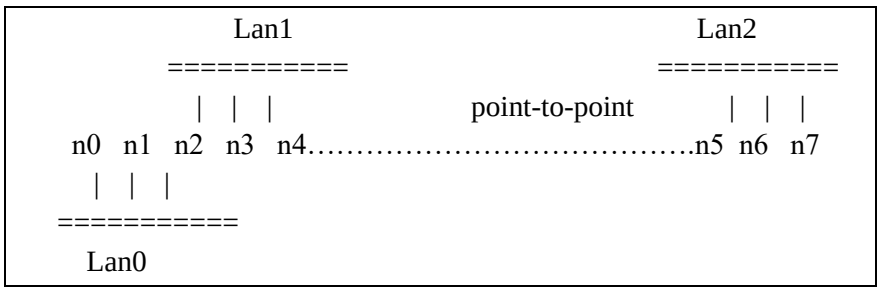


Fig. 4.2: Topology

OBSERVATION SPACE

LAB NO: 5 **Date:**

TCP and UDP configuration

Objectives:

- Configure TCP and UDP protocols
- Examine and Understand the performance differences

TCP:

ns-3 was written to support multiple TCP implementations. There are three important abstract base classes:

- class **TcpSocket**: This is defined in src/internet/model/tcp-socket.{cc,h}. This class exists for hosting TcpSocket attributes that can be reused across different implementations. For instance, the attribute InitialCwnd can be used for any of the implementations that derive from class **TcpSocket**.
- class **TcpSocketFactory**: This is used by the layer-4 protocol instance to create TCP sockets of the right type.
- class **TcpCongestionOps**: This supports different variants of congestion control– a key topic of simulation-based TCP research.

Steps to create TCP connections:

- Create a packet sink to receive

  PacketSinkHelper sink ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), servPort));

  ApplicationContainer apps = sink.Install (NodeContainerSample.Get (1));
  apps.Start (Seconds (0.0));
  apps.Stop (Seconds (3.0));

- Create a source to send packets

  // Create and bind the socket...
  Ptr<Socket> localSocket =Socket::CreateSocket (NodeContainerSample.Get (0), TcpSocketFactory::GetTypeId ());
  localSocket->Bind ();

Sample Network Topology:

```
// Network topology
//
//       n0 ----------- n1
//            500 Kbps
//             5 ms
//
// - Flow from n0 to n1 using SendApplication.
// - Tracing of queues and packet receptions to file "tcp-send.tr"
//   and pcap tracing available when tracing is turned on.
```

```
#include <string>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/network-module.h"
#include "ns3/packet-sink.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TcpSendExample");

int
main (int argc, char *argv[])
{

  bool tracing = false;
  uint32_t maxBytes = 0;

//
// Allow the user to override any of the defaults at
// run-time, via command-line arguments
//
  CommandLine cmd;
  cmd.AddValue ("tracing", "Flag to enable/disable tracing", tracing);
  cmd.AddValue ("maxBytes",
                "Total number of bytes for application to send", maxBytes);
  cmd.Parse (argc, argv);

//
// Explicitly create the nodes required by the topology (shown above).
//
  NS_LOG_INFO ("Create nodes.");
  NodeContainer nodes;
  nodes.Create (2);

  NS_LOG_INFO ("Create channels.");

//
// Explicitly create the point-to-point link required by the topology (shown above).
//
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("500Kbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("5ms"));

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

//
```

```
// Install the internet stack on the nodes
//
  InternetStackHelper internet;
  internet.Install (nodes);

//
// 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");
  Ipv4InterfaceContainer i = ipv4.Assign (devices);

  NS_LOG_INFO ("Create Applications.");

//
// Create a SendApplication and install it on node 0
//
  uint16_t port = 9;  // well-known echo port number


  BulkSendHelper source ("ns3::TcpSocketFactory",
                         InetSocketAddress (i.GetAddress (1), port));
  // Set the amount of data to send in bytes.  Zero is unlimited.
  source.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
  ApplicationContainer sourceApps = source.Install (nodes.Get (0));
  sourceApps.Start (Seconds (0.0));
  sourceApps.Stop (Seconds (10.0));

//
// Create a PacketSinkApplication and install it on node 1
//
  PacketSinkHelper sink ("ns3::TcpSocketFactory",
                         InetSocketAddress (Ipv4Address::GetAny (), port));
  ApplicationContainer sinkApps = sink.Install (nodes.Get (1));
  sinkApps.Start (Seconds (0.0));
  sinkApps.Stop (Seconds (10.0));

//
// Set up tracing if enabled
//
  if (tracing)
    {
      AsciiTraceHelper ascii;
      pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("tcp-send.tr"));
      pointToPoint.EnablePcapAll ("tcp-send", false);
    }

//
// Now, do the actual simulation.
//
```

```
 NS_LOG_INFO ("Run Simulation.");
 Simulator::Stop (Seconds (10.0));
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");

 Ptr<PacketSink> sink1 = DynamicCast<PacketSink> (sinkApps.Get (0));
 std::cout << "Total Bytes Received: " << sink1->GetTotalRx () << std::endl;
}
```

UDP:

ns-3 supports a native implementation of UDP. It provides a connectionless, unreliable datagram packet service. Packets may be reordered or duplicated before they arrive. UDP calculates and checks checksums to catch transmission errors.

Here are the important abstract base classes:

- class **UdpSocket**: This is defined in: src/internet/model/udp-socket.{cc,h} This is an abstract base class of all UDP sockets. This class exists solely for hosting UdpSocket attributes that can be reused across different implementations, and for declaring UDP-specific multicast API.
- class **UdpSocketImpl**: This class subclasses UdpSocket, and provides a socket interface to ns-3's implementation of UDP.
- class **UdpSocketFactory**: This is used by the layer-4 protocol instance to create UDP sockets.
- class **UdpSocketFactoryImpl**: This class is derived from SocketFactory and implements the API for creating UDP sockets.
- class **UdpHeader**: This class contains fields corresponding to those in a network UDP header (port numbers, payload size, checksum) as well as methods for serialization to and deserialization from a byte buffer.
- class **UdpL4Protocol**: This is a subclass of IpL4Protocol and provides an implementation of the UDP protocol.

Steps to create UDP connections:

- Create a packet sink on the receiver

```
uint16_t port = 50000;
Address sinkLocalAddress(InetSocketAddress (Ipv4Address::GetAny (), port));
PacketSinkHelper sinkHelper ("ns3::UdpSocketFactory", sinkLocalAddress);
ApplicationContainer sinkApp = sinkHelper.Install (serverNode);
sinkApp.Start (Seconds (1.0));
sinkApp.Stop (Seconds (10.0));
```

- Create the OnOff applications to send data to the UDP receiver

```
OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
clientHelper.SetAttribute ("Remote", remoteAddress);
ApplicationContainer clientApps = (clientHelper.Install (clientNode);
clientApps.Start (Seconds (2.0));
```

clientApps.Stop (Seconds (9.0));

Sample Network Topology:

```
// Network topology
//
//       n0    n1   n2   n3
//       |    |   |   |
//       ================
//              LAN
//
// - UDP flows from n0 to n1 and back
// - DropTail queues
// - Tracing of queues and packet receptions to file "udp.tr"

#include <fstream>
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("UdpExample");

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 ("UdpExample", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpClientApplication", LOG_LEVEL_ALL);
  LogComponentEnable ("UdpServerApplication", LOG_LEVEL_ALL);
#endif
//
// Allow the user to override any of the defaults and the above Bind() at
// run-time, via command-line arguments
//
  bool useV6 = false;
  Address serverAddress;

  CommandLine cmd;
  cmd.AddValue ("useIpv6", "Use Ipv6", useV6);
  cmd.Parse (argc, argv);
//
// Explicitly create the nodes required by the topology (shown above).
//
  NS_LOG_INFO ("Create nodes.");
```

```
  NodeContainer n;
  n.Create (4);

  InternetStackHelper internet;
  internet.Install (n);

  NS_LOG_INFO ("Create channels.");
//
// Explicitly create the channels required by the topology (shown above).
//
  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate (5000000)));
  csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
  csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));
  NetDeviceContainer d = csma.Install (n);

//
// We've got the "hardware" in place.  Now we need to add IP addresses.
//
  NS_LOG_INFO ("Assign IP Addresses.");
  if (useV6 == false)
    {
      Ipv4AddressHelper ipv4;
      ipv4.SetBase ("10.1.1.0", "255.255.255.0");
      Ipv4InterfaceContainer i = ipv4.Assign (d);
      serverAddress = Address(i.GetAddress (1));
    }
  else
    {
      Ipv6AddressHelper ipv6;
      ipv6.SetBase ("2001:0000:f00d:cafe::", Ipv6Prefix (64));
      Ipv6InterfaceContainer i6 = ipv6.Assign (d);
      serverAddress = Address(i6.GetAddress (1,1));
    }

  NS_LOG_INFO ("Create Applications.");
//
// Create a UdpEchoServer application on node one.
//
  uint16_t port = 9;  // well-known echo port number
  UdpEchoServerHelper server (port);
  ApplicationContainer apps = server.Install (n.Get (1));
  apps.Start (Seconds (1.0));
  apps.Stop (Seconds (10.0));

//
// Create a UdpEchoClient application to send UDP datagrams from node zero to
// node one.
//
  uint32_t packetSize = 1024;
  uint32_t maxPacketCount = 1;
```

```
  Time interPacketInterval = Seconds (1.);
  UdpEchoClientHelper client (serverAddress, port);
  client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  client.SetAttribute ("Interval", TimeValue (interPacketInterval));
  client.SetAttribute ("PacketSize", UintegerValue (packetSize));
  apps = client.Install (n.Get (0));
  apps.Start (Seconds (2.0));
  apps.Stop (Seconds (10.0));

#if 0
//
// Users may find it convenient to initialize echo packets with actual data;
// the below lines suggest how to do this
//
  client.SetFill (apps.Get (0), "Hello World");

  client.SetFill (apps.Get (0), 0xa5, 1024);

  uint8_t fill[] = { 0, 1, 2, 3, 4, 5, 6};
  client.SetFill (apps.Get (0), fill, sizeof(fill), 1024);
#endif

  AsciiTraceHelper ascii;
  csma.EnableAsciiAll (ascii.CreateFileStream ("udp.tr"));
  csma.EnablePcapAll ("udp", false);

//
// Now, do the actual simulation.
//
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

Lab Exercises:

1. Create a bus topology which has 5 nodes and sends large TCP files. Display the network statics and analyze performances for different bandwidth (0.5, 1, 4, 10, 24, and 100 Mbps) and delay (5, 10, 15, 20 and 25 ms).
2. Create a topology with 10 nodes which forms a mesh topology and communicates using UDP.

Additional Exercises:

1. Create a network topology and vary the MTU at MAC level for TCP and UDP communication and analyze the performances of the performances for various network parameters such as bandwidth, delay, packet size and packet intervals.

LAB NO: 6 **Date:**

Wired Routing – RIP

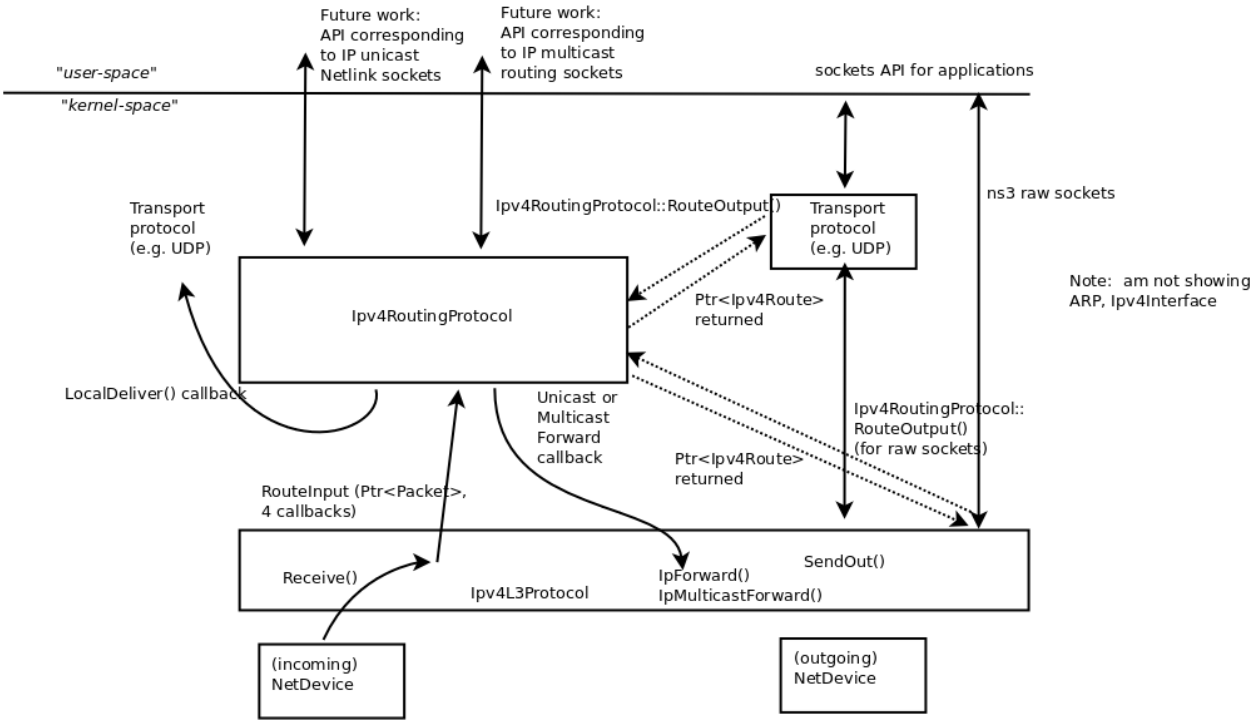
Objectives:

- To understand the routing protocols and its working
- Configure the routing protocols and compare the performances

Routing Overview:

ns-3 is intended to support traditional routing approaches and protocols, support ports of open source routing implementations, and facilitate research into unorthodox routing techniques. The overall routing architecture is described below in Routing architecture.

Routing Architecture:



The following unicast routing protocols are defined for IPv4 and IPv6:

- classes Ipv4ListRouting and Ipv6ListRouting (used to store a prioritized list of routing protocols)
- classes Ipv4StaticRouting and Ipv6StaticRouting (covering both unicast and multicast)
- class Ipv4GlobalRouting (used to store routes computed by the global route manager, if that is used)
- class Ipv4NixVectorRouting (a more efficient version of global routing that stores source routes in a packet header field)
- class Rip - the IPv4 RIPv2 protocol (RFC 2453)
- class RipNg - the IPv6 RIPng protocol (RFC 2080)
- IPv4 Optimized Link State Routing (OLSR) (a MANET protocol defined in RFC 3626)

- IPv4 Ad Hoc On Demand Distance Vector (AODV) (a MANET protocol defined in RFC 3561)
- IPv4 Destination Sequenced Distance Vector (DSDV) (a MANET protocol)
- IPv4 Dynamic Source Routing (DSR) (a MANET protocol)

RIP and RIPng:

The RIPv2 protocol for IPv4 is described in the RFC 2453, and it consolidates a number of improvements over the base protocol defined in RFC 1058.

This IPv6 routing protocol (RFC 2080) is the evolution of the well-known RIPv1 (see RFC 1058 and RFC 1723) routing protocol for IPv4.

The protocols are very simple, and are normally suitable for flat, simple network topologies.

RIPv1, RIPv2, and RIPng have the very same goals and limitations. In particular, RIP considers any route with a metric equal or greater than 16 as unreachable. As a consequence, the maximum number of hops is the network must be less than 15 (the number of routers is not set). Users are encouraged to read RFC 2080 and RFC 1058 to fully understand RIP behaviour and limitations.

RIP uses a Distance-Vector algorithm, and routes are updated according to the Bellman-Ford algorithm

Split Horizon is a strategy to prevent routing instability. Three options are possible:

- No Split Horizon
- Split Horizon
- Poison Reverse

In the first case, routes are advertised on all the router's interfaces. In the second case, routers will not advertise a route on the interface from which it was learned. Poison Reverse will advertise the route on the interface from which it was learned, but with a metric of 16 (infinity). For a full analysis of the three techniques, see RFC 1058, section 2.2.

Sample Network Topology:

```
// Network topology
//
//    SRC
//     |<=== source network
//     A-----B
//      \   / \   all networks have cost 1, except
//       \ /  |   for the direct link from C to D, which
//        C   /    has cost 10
//        | /
//        |/
//        D
//        |<=== target network
//       DST
//
//
// A, B, C and D are RIPng routers.
// A and D are configured with static addresses.
// SRC and DST will exchange packets.
//
```

```
// After about 3 seconds, the topology is built, and Echo Reply will be received.
// After 40 seconds, the link between B and D will break, causing a route failure.
// After 44 seconds from the failure, the routers will recovery from the failure.
// Split Horizoning should affect the recovery time, but it is not. See the manual
// for an explanation of this effect.
//
// If "showPings" is enabled, the user will see:
// 1) if the ping has been acknowledged
// 2) if a Destination Unreachable has been received by the sender
// 3) nothing, when the Echo Request has been received by the destination but
//    the Echo Reply is unable to reach the sender.
// Examining the .pcap files with Wireshark can confirm this effect.


#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/ipv4-routing-table-entry.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("RipSimpleRouting");

void TearDownLink (Ptr<Node> nodeA, Ptr<Node> nodeB, uint32_t interfaceA, uint32_t interfaceB)
{
  nodeA->GetObject<Ipv4> ()->SetDown (interfaceA);
  nodeB->GetObject<Ipv4> ()->SetDown (interfaceB);
}

int main (int argc, char **argv)
{
  bool verbose = false;
  bool printRoutingTables = false;
  bool showPings = false;
  std::string SplitHorizon ("PoisonReverse");

  CommandLine cmd;
  cmd.AddValue ("verbose", "turn on log components", verbose);
  cmd.AddValue ("printRoutingTables", "Print routing tables at 30, 60 and 90 seconds",
printRoutingTables);
  cmd.AddValue ("showPings", "Show Ping6 reception", showPings);
  cmd.AddValue ("splitHorizonStrategy", "Split Horizon strategy to use (NoSplitHorizon,
SplitHorizon, PoisonReverse)", SplitHorizon);
  cmd.Parse (argc, argv);

  if (verbose)
    {
      LogComponentEnableAll (LogLevel (LOG_PREFIX_TIME | LOG_PREFIX_NODE));
```

```
    LogComponentEnable ("RipSimpleRouting", LOG_LEVEL_INFO);
    LogComponentEnable ("Rip", LOG_LEVEL_ALL);
    LogComponentEnable ("Ipv4Interface", LOG_LEVEL_ALL);
    LogComponentEnable ("Icmpv4L4Protocol", LOG_LEVEL_ALL);
    LogComponentEnable ("Ipv4L3Protocol", LOG_LEVEL_ALL);
    LogComponentEnable ("ArpCache", LOG_LEVEL_ALL);
    LogComponentEnable ("V4Ping", LOG_LEVEL_ALL);
  }

if (SplitHorizon == "NoSplitHorizon")
  {
    Config::SetDefault ("ns3::Rip::SplitHorizon", EnumValue (RipNg::NO_SPLIT_HORIZON));
  }
else if (SplitHorizon == "SplitHorizon")
  {
    Config::SetDefault ("ns3::Rip::SplitHorizon", EnumValue (RipNg::SPLIT_HORIZON));
  }
else
  {
    Config::SetDefault ("ns3::Rip::SplitHorizon", EnumValue (RipNg::POISON_REVERSE));
  }

NS_LOG_INFO ("Create nodes.");
Ptr<Node> src = CreateObject<Node> ();
Names::Add ("SrcNode", src);
Ptr<Node> dst = CreateObject<Node> ();
Names::Add ("DstNode", dst);
Ptr<Node> a = CreateObject<Node> ();
Names::Add ("RouterA", a);
Ptr<Node> b = CreateObject<Node> ();
Names::Add ("RouterB", b);
Ptr<Node> c = CreateObject<Node> ();
Names::Add ("RouterC", c);
Ptr<Node> d = CreateObject<Node> ();
Names::Add ("RouterD", d);
NodeContainer net1 (src, a);
NodeContainer net2 (a, b);
NodeContainer net3 (a, c);
NodeContainer net4 (b, c);
NodeContainer net5 (c, d);
NodeContainer net6 (b, d);
NodeContainer net7 (d, dst);
NodeContainer routers (a, b, c, d);
NodeContainer nodes (src, dst);


NS_LOG_INFO ("Create channels.");
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", DataRateValue (5000000));
csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
NetDeviceContainer ndc1 = csma.Install (net1);
```

```
  NetDeviceContainer ndc2 = csma.Install (net2);
  NetDeviceContainer ndc3 = csma.Install (net3);
  NetDeviceContainer ndc4 = csma.Install (net4);
  NetDeviceContainer ndc5 = csma.Install (net5);
  NetDeviceContainer ndc6 = csma.Install (net6);
  NetDeviceContainer ndc7 = csma.Install (net7);

  NS_LOG_INFO ("Create IPv4 and routing");
  RipHelper ripRouting;

  // Rule of thumb:
  // Interfaces are added sequentially, starting from 0
  // However, interface 0 is always the loopback...
  ripRouting.ExcludeInterface (a, 1);
  ripRouting.ExcludeInterface (d, 3);

  ripRouting.SetInterfaceMetric (c, 3, 10);
  ripRouting.SetInterfaceMetric (d, 1, 10);

  Ipv4ListRoutingHelper listRH;
  listRH.Add (ripRouting, 0);
//  Ipv4StaticRoutingHelper staticRh;
//  listRH.Add (staticRh, 5);

  InternetStackHelper internet;
  internet.SetIpv6StackInstall (false);
  internet.SetRoutingHelper (listRH);
  internet.Install (routers);

  InternetStackHelper internetNodes;
  internetNodes.SetIpv6StackInstall (false);
  internetNodes.Install (nodes);

  // Assign addresses.
  // The source and destination networks have global addresses
  // The "core" network just needs link-local addresses for routing.
  // We assign global addresses to the routers as well to receive
  // ICMPv6 errors.
  NS_LOG_INFO ("Assign IPv4 Addresses.");
  Ipv4AddressHelper ipv4;

  ipv4.SetBase (Ipv4Address ("10.0.0.0"), Ipv4Mask ("255.255.255.0"));
  Ipv4InterfaceContainer iic1 = ipv4.Assign (ndc1);

  ipv4.SetBase (Ipv4Address ("10.0.1.0"), Ipv4Mask ("255.255.255.0"));
  Ipv4InterfaceContainer iic2 = ipv4.Assign (ndc2);

  ipv4.SetBase (Ipv4Address ("10.0.2.0"), Ipv4Mask ("255.255.255.0"));
  Ipv4InterfaceContainer iic3 = ipv4.Assign (ndc3);

  ipv4.SetBase (Ipv4Address ("10.0.3.0"), Ipv4Mask ("255.255.255.0"));
```

```
 Ipv4InterfaceContainer iic4 = ipv4.Assign (ndc4);

 ipv4.SetBase (Ipv4Address ("10.0.4.0"), Ipv4Mask ("255.255.255.0"));
 Ipv4InterfaceContainer iic5 = ipv4.Assign (ndc5);

 ipv4.SetBase (Ipv4Address ("10.0.5.0"), Ipv4Mask ("255.255.255.0"));
 Ipv4InterfaceContainer iic6 = ipv4.Assign (ndc6);

 ipv4.SetBase (Ipv4Address ("10.0.6.0"), Ipv4Mask ("255.255.255.0"));
 Ipv4InterfaceContainer iic7 = ipv4.Assign (ndc7);

 Ptr<Ipv4StaticRouting> staticRouting;
 staticRouting = Ipv4RoutingHelper::GetRouting <Ipv4StaticRouting> (src->GetObject<Ipv4> ()-
>GetRoutingProtocol ());
 staticRouting->SetDefaultRoute ("10.0.0.2", 1 );
 staticRouting = Ipv4RoutingHelper::GetRouting <Ipv4StaticRouting> (dst->GetObject<Ipv4> ()-
>GetRoutingProtocol ());
 staticRouting->SetDefaultRoute ("10.0.6.1", 1 );

 if (printRoutingTables)
   {
     RipHelper routingHelper;

     Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> (&std::cout);

     routingHelper.PrintRoutingTableAt (Seconds (30.0), a, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (30.0), b, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (30.0), c, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (30.0), d, routingStream);

     routingHelper.PrintRoutingTableAt (Seconds (60.0), a, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (60.0), b, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (60.0), c, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (60.0), d, routingStream);

     routingHelper.PrintRoutingTableAt (Seconds (90.0), a, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (90.0), b, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (90.0), c, routingStream);
     routingHelper.PrintRoutingTableAt (Seconds (90.0), d, routingStream);
   }

 NS_LOG_INFO ("Create Applications.");
 uint32_t packetSize = 1024;
 Time interPacketInterval = Seconds (1.0);
 V4PingHelper ping ("10.0.6.2");

 ping.SetAttribute ("Interval", TimeValue (interPacketInterval));
 ping.SetAttribute ("Size", UintegerValue (packetSize));
 if (showPings)
   {
     ping.SetAttribute ("Verbose", BooleanValue (true));
```

```
  }
 ApplicationContainer apps = ping.Install (src);
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (110.0));

 AsciiTraceHelper ascii;
 csma.EnableAsciiAll (ascii.CreateFileStream ("rip-simple-routing.tr"));
 csma.EnablePcapAll ("rip-simple-routing", true);

 Simulator::Schedule (Seconds (40), &TearDownLink, b, d, 3, 2);

 /* Now, do the actual simulation. */
 NS_LOG_INFO ("Run Simulation.");
 Simulator::Stop (Seconds (131.0));
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");
}
```

Lab Exercises:

1. Implement RIPng for the below given scenario.

```
   SRC
    |<=== source network
    A-----B
     \   / \  |      all networks have cost 1, except
      \ /  |  |      for the direct link from C to D, which
       C  /   |     has cost 10
       | /    |
       |/     |
       D ------E
       |<=== target network
      DST
```

2. Compare the performances of RIP and RIPng for network parameters.

Additional Exercises:

1. Create topology and assign different cost for links. Then, compare the performances for different conditions.

OBSERVATION SPACE

LAB NO: 7 **Date:**

Wireless Network and Communication

Objective:

- To create wireless nodes and enable different features of wireless devices
- To understand the performances of wireless communication

Wireless Network Topology:

ns-3 provides a set of 802.11 models that attempt to provide an accurate MAC-level implementation of the 802.11 specification and a “not-so-slow” PHY-level model of the 802.11a specification.

There are a couple of new includes corresponding to the wifi module and the mobility module which we will discuss below.

```
#include "ns3/command-line.h"
#include "ns3/config.h"
#include "ns3/string.h"
#include "ns3/log.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/mobility-helper.h"
#include "ns3/on-off-helper.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/mobility-model.h"
#include "ns3/packet-sink.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/tcp-westwood.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/ipv4-global-routing-helper.h"
```

Setup legacy Channel:

ns3::WifiChannel is an abstract base class that allows different channel implementations to be connected. At present, there is only one such channel (the ns3::YansWifiChannel). The class works in tandem with the ns3::WifiPhy class; if you want to provide a new physical layer model, you must subclass both ns3::WifiChannel and ns3::WifiPhy. The WifiChannel model exists to interconnect WifiPhy objects so that packets sent by one Phy are received by some or all other Phys on the channel.

The following two methods are useful when configuring YansWifiChannelHelper:

- YansWifiChannelHelper::AddPropagationLoss adds a PropagationLossModel to a chain of PropagationLossModel
- YansWifiChannelHelper::SetPropagationDelay sets a PropagationDelayModel

The next bit of code constructs the wifi devices and the interconnection channel between these wifi nodes. First, we configure the PHY and channel helpers:

```
YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel", "Frequency",
DoubleValue (5e9));
```

Setup Physical Layer:

Physical devices (base class ns3::WifiPhy) connect to ns3::WifiChannel models in ns-3. We need to create WifiPhy objects appropriate for the YansWifiChannel; here the YansWifiPhyHelper will do the work. The YansWifiPhyHelper class configures an object factory to create instances of a YansWifiPhy and adds some other objects to it, including possibly a supplemental ErrorRateModel and a pointer to a MobilityModel. The user code is typically:

```
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.SetChannel (wifiChannel.Create ());
wifiPhy.Set ("TxPowerStart", DoubleValue (10.0));
wifiPhy.Set ("TxPowerEnd", DoubleValue (10.0));
wifiPhy.Set ("TxPowerLevels", UintegerValue (1));
wifiPhy.Set ("TxGain", DoubleValue (0));
wifiPhy.Set ("RxGain", DoubleValue (0));
wifiPhy.Set ("RxNoiseFigure", DoubleValue (10));
wifiPhy.Set ("CcaMode1Threshold", DoubleValue (-79));
wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue (-79 + 3));
wifiPhy.SetErrorRateModel ("ns3::YansErrorRateModel");
wifiHelper.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode", StringValue (phyRate), "ControlMode", StringValue
("HtMcs0"));
```

Create Nodes and enable wireless features:

```
NodeContainer networkNodes;
networkNodes.Create (2);
Ptr<Node> apWifiNode = networkNodes.Get (0);
Ptr<Node> staWifiNode = networkNodes.Get (1);

/* Configure AP */
Ssid ssid = Ssid ("network");
wifiMac.SetType ("ns3::ApWifiMac",
    "Ssid", SsidValue (ssid));

NetDeviceContainer apDevice;
apDevice = wifiHelper.Install (wifiPhy, wifiMac, apWifiNode);

/* Configure STA */
WifiMac.SetType ("ns3::StaWifiMac",
    "Ssid", SsidValue (ssid));

NetDeviceContainer staDevices;
staDevices = wifiHelper.Install (wifiPhy, wifiMac, staWifiNode);
```

Simple Network Topology:

```
/* Network topology:
 *
 *   Ap    STA
 *   *     *
 *   |     |
 *   n1    n2
 *
 * In this example, an HT station sends TCP packets to the access point.
 * We report the total throughput received during a window of 100ms.
 * The user can specify the application data rate and choose the variant
 * of TCP i.e. congestion control algorithm to use.
 */

#include "ns3/command-line.h"
#include "ns3/config.h"
#include "ns3/string.h"
#include "ns3/log.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/mobility-helper.h"
#include "ns3/on-off-helper.h"
#include "ns3/yans-wifi-channel.h"
#include "ns3/mobility-model.h"
#include "ns3/packet-sink.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/tcp-westwood.h"
#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/ipv4-global-routing-helper.h"

NS_LOG_COMPONENT_DEFINE ("wifi-tcp");

using namespace ns3;

Ptr<PacketSink> sink; /* Pointer to the packet sink application */
uint64_t lastTotalRx = 0; /* The value of the last total received bytes */

void
CalculateThroughput ()
{
  Time now = Simulator::Now (); /* Return the simulator's virtual time. */
  double cur = (sink->GetTotalRx () - lastTotalRx) * (double) 8 / 1e5;     /* Convert Application RX
Packets to MBits. */
  std::cout << now.GetSeconds () << "s: \t" << cur << " Mbit/s" << std::endl;
  lastTotalRx = sink->GetTotalRx ();
  Simulator::Schedule (MilliSeconds (100), &CalculateThroughput);
}

int
```

```
main (int argc, char *argv[])
{
  uint32_t payloadSize = 1472; /* Transport layer payload size in bytes. */
  std::string dataRate = "100Mbps"; /* Application layer datarate. */
  std::string tcpVariant = "TcpNewReno"; /* TCP variant type. */
  std::string phyRate = "HtMcs7"; /* Physical layer bitrate. */
  double simulationTime = 10; /* Simulation time in seconds. */
  bool pcapTracing = false; /* PCAP Tracing is enabled or not. */

  /* Command line argument parser setup. */
  CommandLine cmd;
  cmd.AddValue ("payloadSize", "Payload size in bytes", payloadSize);
  cmd.AddValue ("dataRate", "Application data ate", dataRate);
  cmd.AddValue ("tcpVariant", "Transport protocol to use: TcpNewReno, "
                "TcpHybla, TcpHighSpeed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno, "
                "TcpBic, TcpYeah, TcpIllinois, TcpWestwood, TcpWestwoodPlus, TcpLedbat ",
tcpVariant);
  cmd.AddValue ("phyRate", "Physical layer bitrate", phyRate);
  cmd.AddValue ("simulationTime", "Simulation time in seconds", simulationTime);
  cmd.AddValue ("pcap", "Enable/disable PCAP Tracing", pcapTracing);
  cmd.Parse (argc, argv);

  tcpVariant = std::string ("ns3::") + tcpVariant;
  // Select TCP variant
  if (tcpVariant.compare ("ns3::TcpWestwoodPlus") == 0)
    {
      // TcpWestwoodPlus is not an actual TypeId name; we need TcpWestwood here
      Config::SetDefault ("ns3::TcpL4Protocol::SocketType", TypeIdValue (TcpWestwood::GetTypeId
()));
      // the default protocol type in ns3::TcpWestwood is WESTWOOD
      Config::SetDefault ("ns3::TcpWestwood::ProtocolType", EnumValue
(TcpWestwood::WESTWOODPLUS));
    }
  else
    {
      TypeId tcpTid;
      NS_ABORT_MSG_UNLESS (TypeId::LookupByNameFailSafe (tcpVariant, &tcpTid), "TypeId
" << tcpVariant << " not found");
      Config::SetDefault ("ns3::TcpL4Protocol::SocketType", TypeIdValue (TypeId::LookupByName
(tcpVariant)));
    }

  /* Configure TCP Options */
  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

  WifiMacHelper wifiMac;
  WifiHelper wifiHelper;
  wifiHelper.SetStandard (WIFI_PHY_STANDARD_80211n_5GHZ);

  /* Set up Legacy Channel */
  YansWifiChannelHelper wifiChannel;
```

```
 wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
 wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel", "Frequency", DoubleValue
(5e9));

 /* Setup Physical Layer */
 YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
 wifiPhy.SetChannel (wifiChannel.Create ());
 wifiPhy.Set ("TxPowerStart", DoubleValue (10.0));
 wifiPhy.Set ("TxPowerEnd", DoubleValue (10.0));
 wifiPhy.Set ("TxPowerLevels", UintegerValue (1));
 wifiPhy.Set ("TxGain", DoubleValue (0));
 wifiPhy.Set ("RxGain", DoubleValue (0));
 wifiPhy.Set ("RxNoiseFigure", DoubleValue (10));
 wifiPhy.Set ("CcaMode1Threshold", DoubleValue (-79));
 wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue (-79 + 3));
 wifiPhy.SetErrorRateModel ("ns3::YansErrorRateModel");
 wifiHelper.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode",StringValue(phyRate),                    "ControlMode", StringValue ("HtMcs0"));

 NodeContainer networkNodes;
 networkNodes.Create (2);
 Ptr<Node> apWifiNode = networkNodes.Get (0);
 Ptr<Node> staWifiNode = networkNodes.Get (1);

 /* Configure AP */
 Ssid ssid = Ssid ("network");
 wifiMac.SetType ("ns3::ApWifiMac",
                  "Ssid", SsidValue (ssid));

 NetDeviceContainer apDevice;
 apDevice = wifiHelper.Install (wifiPhy, wifiMac, apWifiNode);

 /* Configure STA */
 wifiMac.SetType ("ns3::StaWifiMac",
                  "Ssid", SsidValue (ssid));

 NetDeviceContainer staDevices;
 staDevices = wifiHelper.Install (wifiPhy, wifiMac, staWifiNode);

 /* Mobility model */
 MobilityHelper mobility;
 Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
 positionAlloc->Add (Vector (0.0, 0.0, 0.0));
 positionAlloc->Add (Vector (1.0, 1.0, 0.0));

 mobility.SetPositionAllocator (positionAlloc);
 mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
 mobility.Install (apWifiNode);
 mobility.Install (staWifiNode);

 /* Internet stack */
```

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

 Ipv4AddressHelper address;
 address.SetBase ("10.0.0.0", "255.255.255.0");
 Ipv4InterfaceContainer apInterface;
 apInterface = address.Assign (apDevice);
 Ipv4InterfaceContainer staInterface;
 staInterface = address.Assign (staDevices);

 /* Populate routing table */
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

 /* Install TCP Receiver on the access point */
 PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny
(), 9));
 ApplicationContainer sinkApp = sinkHelper.Install (apWifiNode);
 sink = StaticCast<PacketSink> (sinkApp.Get (0));

 /* Install TCP/UDP Transmitter on the station */
 OnOffHelper server ("ns3::TcpSocketFactory", (InetSocketAddress (apInterface.GetAddress (0),
9)));
 server.SetAttribute ("PacketSize", UintegerValue (payloadSize));
 server.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));
 server.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));
 server.SetAttribute ("DataRate", DataRateValue (DataRate (dataRate)));
 ApplicationContainer serverApp = server.Install (staWifiNode);

 /* Start Applications */
 sinkApp.Start (Seconds (0.0));
 serverApp.Start (Seconds (1.0));
 Simulator::Schedule (Seconds (1.1), &CalculateThroughput);

 /* Enable Traces */
 if (pcapTracing)
   {
    wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
    wifiPhy.EnablePcap ("AccessPoint", apDevice);
    wifiPhy.EnablePcap ("Station", staDevices);
   }

 /* Start Simulation */
 Simulator::Stop (Seconds (simulationTime + 1));
 Simulator::Run ();

 double averageThroughput = ((sink->GetTotalRx () * 8) / (1e6 * simulationTime));

 Simulator::Destroy ();

 if (averageThroughput < 50)
   {
```

```
    NS_LOG_ERROR ("Obtained throughput is not in the expected boundaries!");
    exit (1);
   }
 std::cout << "\nAverage throughput: " << averageThroughput << " Mbit/s" << std::endl;
 return 0;
}
```

Lab Exercise:

1. Create a wireless network with 10 nodes and establish TCP and UDP communication. Compare the performances of the communication for varied bandwidth and application layer datarate.

Additional Exercises:

1. Create different topology for varied number of network nodes. Compare the performances.
2. Check the network statistics for varied physical layer parameters.

OBSERVATION SPACE

LAB NO: 8 **Date:**

Mobility, Energy configuration

Objectives:

- To configure various mobility models
- To compare the performances of energy models
- To analyze the network performances for mobility models

Mobility Configurations:

A mobility model must be configured on each node with Wi-Fi device. Mobility model is used for calculating propagation loss and propagation delay.

AdHoc WifiNetDevice configuration:

In this example, we create two ad-hoc nodes equipped with 802.11a Wi-Fi devices. We use the ns3::ConstantSpeedPropagationDelayModel as the propagation delay model and ns3::LogDistancePropagationLossModel with the exponent of 3.0 as the propagation loss model. Both devices are configured with ConstantRateWifiManager at the fixed rate of 12Mbps. Finally, we manually place them by using the ns3::ListPositionAllocator:

```
std::string phyMode ("OfdmRate12Mbps");

NodeContainer c;
c.Create (2);

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
// ns-3 supports RadioTap and Prism tracing extensions for 802.11
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel",
"Exponent", DoubleValue (3.0));
wifiPhy.SetChannel (wifiChannel.Create ());

// Add a non-QoS upper mac, and disable rate control (i.e. ConstantRateWifiManager)
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode",StringValue (phyMode),
"ControlMode",StringValue (phyMode));

// Set it to adhoc mode
wifiMac.SetType ("ns3::AdhocWifiMac");
```

```
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

// Configure mobility
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (0.0, 0.0, 0.0));
positionAlloc->Add (Vector (5.0, 0.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (c);

// other set up (e.g. InternetStack, Application)
```

Infrastructure (access point and clients) WifiNetDevice configuration

This is a typical example of how a user might configure an access point and a set of clients. In this example, we create one access point and two clients. Each node is equipped with 802.11b Wi-Fi device:

```
std::string phyMode ("DsssRate1Mbps");

NodeContainer ap;
ap.Create (1);
NodeContainer sta;
sta.Create (2);

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
// ns-3 supports RadioTap and Prism tracing extensions for 802.11
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

YansWifiChannelHelper wifiChannel;
// reference loss must be changed since 802.11b is operating at 2.4GHz
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel",
"Exponent", DoubleValue (3.0),
"ReferenceLoss", DoubleValue (40.0459));
wifiPhy.SetChannel (wifiChannel.Create ());

// Add a non-QoS upper mac, and disable rate control
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
"DataMode",StringValue (phyMode),
"ControlMode",StringValue (phyMode));

// Setup the rest of the upper mac
Ssid ssid = Ssid ("wifi-default");
// setup ap.
wifiMac.SetType ("ns3::ApWifiMac",
"Ssid", SsidValue (ssid));
```

```
NetDeviceContainer apDevice = wifi.Install (wifiPhy, wifiMac, ap);
NetDeviceContainer devices = apDevice;

// setup sta.
wifiMac.SetType ("ns3::StaWifiMac",
"Ssid", SsidValue (ssid),
"ActiveProbing", BooleanValue (false));
NetDeviceContainer staDevice = wifi.Install (wifiPhy, wifiMac, sta);
devices.Add (staDevice);

// Configure mobility
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (0.0, 0.0, 0.0));
positionAlloc->Add (Vector (5.0, 0.0, 0.0));
positionAlloc->Add (Vector (0.0, 5.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (ap);
mobility.Install (sta);

// other set up (e.g. InternetStack, Application)
```

Energy Framework:

Energy consumption is a key issue for wireless devices, and wireless network researchers often need to investigate the energy consumption at a node or in the overall network while running network simulations in ns-3. This requires ns-3 to support energy consumption modeling. Further, as concepts such as fuel cells and energy scavenging are becoming viable for low power wireless devices, incorporating the effect of these emerging technologies into simulations requires support for modeling diverse energy sources in ns-3. The ns-3 Energy Framework provides the basis for energy consumption, energy source and energy harvesting modeling.

The ns-3 Energy Framework is composed of 3 parts: Energy Source, Device Energy Model and Energy Harvester. The framework is implemented into the src/energy/models folder.

WiFi Radio Energy Model

The WiFi Radio Energy Model is the energy consumption model of a Wifi net device. It provides a state for each of the available states of the PHY layer: Idle, CcaBusy, Tx, Rx, ChannelSwitch, Sleep, Off. Each of such states is associated with a value (in Ampere) of the current draw (see below for the corresponding attribute names). A Wifi Radio Energy Model PHY Listener is registered to the Wifi PHY in order to be notified of every Wifi PHY state transition. At every transition, the energy consumed in the previous state is computed and the energy source is notified in order to update its remaining energy.

The Wifi Tx Current Model gives the possibility to compute the current draw in the transmit state as a function of the nominal tx power (in dBm), as observed in several experimental measurements. To this purpose, the Wifi Radio Energy Model PHY Listener is notified of the nominal tx power used to transmit the current frame and passes such a value to the Wifi Tx Current Model which takes care of updating the current draw in the Tx state. Hence, the energy consumption is correctly computed even if the Wifi Remote Station Manager performs per-frame power control. Currently, a Linear Wifi Tx

Current Model is implemented which computes the tx current as a linear function (according to parameters that can be specified by the user) of the nominal tx power in dBm.

The Wifi Radio Energy Model offers the possibility to specify a callback that is invoked when the energy source is depleted. If such a callback is not specified when the Wifi Radio Energy Model Helper is used to install the model on a device, a callback is implicitly made so that the Wifi PHY is put in the OFF mode (hence no frame is transmitted nor received afterwards) when the energy source is depleted. Likewise, it is possible to specify a callback that is invoked when the energy source is recharged (which might occur in case an energy harvester is connected to the energy source). If such a callback is not specified when the Wifi Radio Energy Model Helper is used to install the model on a device, a callback is implicitly made so that the Wifi PHY is resumed from the OFF mode when the energy source is recharged.

Create Energy Model:

```
/** Energy Model **/
/*********************************************************************
*/
/* energy source */
BasicEnergySourceHelper basicSourceHelper;
// configure energy source
basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (0.1));
// install source
EnergySourceContainer sources = basicSourceHelper.Install (c);
/* device energy model */
WifiRadioEnergyModelHelper radioEnergyHelper;
// configure radio energy model
radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0174));
// install device model
DeviceEnergyModelContainer deviceModels = radioEnergyHelper.Install (devices, sources);

…

/** connect trace sources **/

/*********************************************************************
*/
  // all sources are connected to node 1
  // energy source
  Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (sources.Get
(1));
  basicSourcePtr->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback
(&RemainingEnergy));
  // device energy model
  Ptr<DeviceEnergyModel> basicRadioModelPtr =
    basicSourcePtr->FindDeviceEnergyModels ("ns3::WifiRadioEnergyModel").Get (0);
  NS_ASSERT (basicRadioModelPtr != NULL);
  basicRadioModelPtr->TraceConnectWithoutContext ("TotalEnergyConsumption",
MakeCallback (&TotalEnergy));
```

Sample Network Topology:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store-module.h"
#include "ns3/energy-module.h"
#include "ns3/internet-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/wifi-radio-energy-model-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("EnergyExample");

static inline std::string
PrintReceivedPacket (Address& from)
{
  InetSocketAddress iaddr = InetSocketAddress::ConvertFrom (from);

  std::ostringstream oss;
  oss << "--\nReceived one packet! Socket: " << iaddr.GetIpv4 ()
      << " port: " << iaddr.GetPort ()
      << " at time = " << Simulator::Now ().GetSeconds ()
      << "\n--";

  return oss.str ();
}

/**
 * \param socket Pointer to socket.
 *
 * Packet receiving sink.
 */
void
ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  Address from;
  while ((packet = socket->RecvFrom (from)))
    {
      if (packet->GetSize () > 0)
        {
          NS_LOG_UNCOND (PrintReceivedPacket (from));
        }
    }
}
```

```
/**
 * \param socket Pointer to socket.
 * \param pktSize Packet size.
 * \param n Pointer to node.
 * \param pktCount Number of packets to generate.
 * \param pktInterval Packet sending interval.
 *
 * Traffic generator.
 */
static void
GenerateTraffic (Ptr<Socket> socket, uint32_t pktSize, Ptr<Node> n,
                 uint32_t pktCount, Time pktInterval)
{
  if (pktCount > 0)
    {
      socket->Send (Create<Packet> (pktSize));
      Simulator::Schedule (pktInterval, &GenerateTraffic, socket, pktSize, n,
                           pktCount - 1, pktInterval);
    }
  else
    {
      socket->Close ();
    }
}

/// Trace function for remaining energy at node.
void
RemainingEnergy (double oldValue, double remainingEnergy)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds ()
                 << "s Current remaining energy = " << remainingEnergy << "J");
}

/// Trace function for total energy consumption at node.
void
TotalEnergy (double oldValue, double totalEnergy)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds ()
                 << "s Total energy consumed by radio = " << totalEnergy << "J");
}

int
main (int argc, char *argv[])
{
  /*
  LogComponentEnable ("EnergySource", LOG_LEVEL_DEBUG);
  LogComponentEnable ("BasicEnergySource", LOG_LEVEL_DEBUG);
  LogComponentEnable ("DeviceEnergyModel", LOG_LEVEL_DEBUG);
  LogComponentEnable ("WifiRadioEnergyModel", LOG_LEVEL_DEBUG);
   */
```

```
 LogComponentEnable ("EnergyExample", LogLevel (LOG_PREFIX_TIME |
LOG_PREFIX_NODE | LOG_LEVEL_INFO));

 std::string phyMode ("DsssRate1Mbps");
 double Prss = -80;            // dBm
 uint32_t PpacketSize = 200;   // bytes
 bool verbose = false;

 // simulation parameters
 uint32_t numPackets = 10000;  // number of packets to send
 double interval = 1;          // seconds
 double startTime = 0.0;       // seconds
 double distanceToRx = 100.0;  // meters
 /*
  * This is a magic number used to set the transmit power, based on other
  * configuration.
  */
 double offset = 81;

 CommandLine cmd;
 cmd.AddValue ("phyMode", "Wifi Phy mode", phyMode);
 cmd.AddValue ("Prss", "Intended primary RSS (dBm)", Prss);
 cmd.AddValue ("PpacketSize", "size of application packet sent", PpacketSize);
 cmd.AddValue ("numPackets", "Total number of packets to send", numPackets);
 cmd.AddValue ("startTime", "Simulation start time", startTime);
 cmd.AddValue ("distanceToRx", "X-Axis distance between nodes", distanceToRx);
 cmd.AddValue ("verbose", "Turn on all device log components", verbose);
 cmd.Parse (argc, argv);

 // Convert to time object
 Time interPacketInterval = Seconds (interval);

 // disable fragmentation for frames below 2200 bytes
 Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
                     StringValue ("2200"));
 // turn off RTS/CTS for frames below 2200 bytes
 Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
                     StringValue ("2200"));
 // Fix non-unicast data rate to be the same as that of unicast
 Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                     StringValue (phyMode));

 NodeContainer c;
 c.Create (2);     // create 2 nodes
 NodeContainer networkNodes;
 networkNodes.Add (c.Get (0));
 networkNodes.Add (c.Get (1));

 // The below set of helpers will help us to put together the wifi NICs we want
 WifiHelper wifi;
```

```
if (verbose)
  {
    wifi.EnableLogComponents ();
  }
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

/** Wifi PHY **/
/***************************************************************************/
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.Set ("RxGain", DoubleValue (-10));
wifiPhy.Set ("TxGain", DoubleValue (offset + Prss));
wifiPhy.Set ("CcaMode1Threshold", DoubleValue (0.0));
/***************************************************************************/

/** wifi channel **/
YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
// create wifi channel
Ptr<YansWifiChannel> wifiChannelPtr = wifiChannel.Create ();
wifiPhy.SetChannel (wifiChannelPtr);

/** MAC layer **/
// Add a MAC and disable rate control
WifiMacHelper wifiMac;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode",
                              StringValue (phyMode), "ControlMode",
                              StringValue (phyMode));
// Set it to ad-hoc mode
wifiMac.SetType ("ns3::AdhocWifiMac");

/** install PHY + MAC **/
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, networkNodes);

/** mobility **/
MobilityHelper mobility;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (0.0, 0.0, 0.0));
positionAlloc->Add (Vector (2 * distanceToRx, 0.0, 0.0));
mobility.SetPositionAllocator (positionAlloc);
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (c);

/** Energy Model **/
/***************************************************************************/
/* energy source */
BasicEnergySourceHelper basicSourceHelper;
// configure energy source
basicSourceHelper.Set ("BasicEnergySourceInitialEnergyJ", DoubleValue (0.1));
// install source
EnergySourceContainer sources = basicSourceHelper.Install (c);
```

```
 /* device energy model */
 WifiRadioEnergyModelHelper radioEnergyHelper;
 // configure radio energy model
 radioEnergyHelper.Set ("TxCurrentA", DoubleValue (0.0174));
 // install device model
 DeviceEnergyModelContainer deviceModels = radioEnergyHelper.Install (devices, sources);
 /*****************************************************************************/

 /** Internet stack **/
 InternetStackHelper internet;
 internet.Install (networkNodes);

 Ipv4AddressHelper ipv4;
 NS_LOG_INFO ("Assign IP Addresses.");
 ipv4.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer i = ipv4.Assign (devices);

 TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
 Ptr<Socket> recvSink = Socket::CreateSocket (networkNodes.Get (1), tid);  // node 1, receiver
 InetSocketAddress local = InetSocketAddress (Ipv4Address::GetAny (), 80);
 recvSink->Bind (local);
 recvSink->SetRecvCallback (MakeCallback (&ReceivePacket));

 Ptr<Socket> source = Socket::CreateSocket (networkNodes.Get (0), tid);    // node 0, sender
 InetSocketAddress remote = InetSocketAddress (Ipv4Address::GetBroadcast (), 80);
 source->SetAllowBroadcast (true);
 source->Connect (remote);

 /** connect trace sources **/
 /*****************************************************************************/
 // all sources are connected to node 1
 // energy source
 Ptr<BasicEnergySource> basicSourcePtr = DynamicCast<BasicEnergySource> (sources.Get (1));
 basicSourcePtr->TraceConnectWithoutContext ("RemainingEnergy", MakeCallback
(&RemainingEnergy));
 // device energy model
 Ptr<DeviceEnergyModel> basicRadioModelPtr =
   basicSourcePtr->FindDeviceEnergyModels ("ns3::WifiRadioEnergyModel").Get (0);
 NS_ASSERT (basicRadioModelPtr != NULL);
 basicRadioModelPtr->TraceConnectWithoutContext ("TotalEnergyConsumption", MakeCallback
(&TotalEnergy));
 /*****************************************************************************/


 /** simulation setup **/
 // start traffic
 Simulator::Schedule (Seconds (startTime), &GenerateTraffic, source, PpacketSize,
                      networkNodes.Get (0), numPackets, interPacketInterval);

 Simulator::Stop (Seconds (10.0));
 Simulator::Run ();
```

```
  for (DeviceEnergyModelContainer::Iterator iter = deviceModels.Begin (); iter != deviceModels.End
();  iter ++)
    {
      double energyConsumed = (*iter)->GetTotalEnergyConsumption ();
      NS_LOG_UNCOND ("End of simulation (" << Simulator::Now ().GetSeconds ()
                     << "s) Total energy consumed by radio = " << energyConsumed << "J");
      NS_ASSERT (energyConsumed <= 0.1);
    }

  Simulator::Destroy ();

  return 0;
}
```

Lab Exercises:

1. Simulate the wireless environments for various node mobility speeds and analyze the quality of the communication in terms of throughput and Packet Delivery Ratio.
2. Create a wireless ad-hoc network scenario and check the energy consumption for varied network conditions such as node mobility, data-rate, and network coverage area.

Additional Exercises:

1. Analyze the network performances for TCP and UDP for varied node mobility models and energy models.

OBSERVATION SPACE

LAB NO: 9 **Date:**

Routing protocols – AODV, DSR, DSDV

Objectives:

- Configure AODV, DSR, and DSDV routing protocols in wireless network
- Understand the protocols and analyse the performances

Introduction

AODV

The Ad hoc On-Demand Distance Vector (AODV) routing protocol is intended for use by mobile nodes in an ad hoc network. It offers quick adaptation to dynamic link conditions, low processing and memory overhead, low network utilization, and determines unicast routes to destinations within the ad hoc network. It uses destination sequence numbers to ensure loop freedom at all times (even in the face of anomalous delivery of routing control messages), avoiding problems (such as "counting to infinity") associated with classical distance vector protocols.

Design

Class ns3::aodv::RoutingProtocol implements all functionality of service packet exchange and inherits from ns3::Ipv4RoutingProtocol. The base class defines two virtual functions for packet routing and forwarding. The first one, ns3::aodv::RouteOutput, is used for locally originated packets, and the second one, ns3::aodv::RouteInput, is used for forwarding and/or delivering received packets.

Protocol operation depends on many adjustable parameters. Parameters for this functionality are attributes of ns3::aodv::RoutingProtocol. Parameter default values are drawn from the RFC and allow the enabling/disabling protocol features, such as broadcasting HELLO messages, broadcasting data packets and so on. AODV discovers routes on demand. Therefore, the AODV model buffers all packets while a route request packet (RREQ) is disseminated. A packet queue is implemented in aodv-rqueue.cc. A smart pointer to the packet, ns3::Ipv4RoutingProtocol::ErrorCallback, ns3::Ipv4RoutingProtocol::UnicastForwardCallback, and the IP header are stored in this queue. The packet queue implements garbage collection of old packets and a queue size limit. The routing table implementation supports garbage collection of old entries and state machine, defined in the standard. It is implemented as a STL map container. The key is a destination IP address.

AODV implementation can detect the presence of unidirectional links and avoid them if necessary. If the node the model receives an RREQ for is a neighbor, the cause may be a unidirectional link. This heuristic is taken from AODV-UU implementation and can be disabled. Protocol operation strongly depends on broken link detection mechanism. The model implements two such heuristics. First, this implementation support HELLO messages. However HELLO messages are not a good way to perform neighbor sensing in a wireless environment (at least not over 802.11). Therefore, one may experience bad performance when running over wireless. There are several reasons for this: 1) HELLO messages are broadcasted. In 802.11, broadcasting is often done at a lower bit rate than unicasting, thus HELLO messages can travel further than unicast data. 2) HELLO messages are small, thus less prone to bit errors than data transmissions and 3) Broadcast transmissions are not guaranteed to be bidirectional, unlike unicast transmissions. Second, we use layer 2 feedback when possible. Link are considered to be broken if frame transmission results in a transmission failure for all retries. This mechanism is meant for active links and works faster than the first method.

DSDV

Destination-Sequenced Distance Vector (DSDV) routing protocol is a pro-active, table-driven routing protocol for MANETs developed by Charles E. Perkins and Pravin Bhagwat in 1994. It uses the hop count as metric in route selection. DSDV Routing Table: Every node will maintain a table listing all the other nodes it has known either directly or through some neighbors. Every node has a single entry in the routing table. The entry will have information about the node's IP address, last known sequence number and the hop count to reach that node. Along with these details the table also keeps track of the nexthop neighbor to reach the destination node, the timestamp of the last update received for that node.

The DSDV update message consists of three fields, Destination Address, Sequence Number and Hop Count.

Each node uses 2 mechanisms to send out the DSDV updates. They are,

1. **Periodic Updates:** are sent out after every m_periodicUpdateInterval(default:15s). In this update the node broadcasts out its entire routing table.
2. **Trigger Updates:** are small updates in-between the periodic updates. These updates are sent out whenever a node receives a DSDV packet that caused a change in its routing table. The original paper did not clearly mention when for what change in the table should a DSDV update be sent out. The current implementation sends out an update irrespective of the change in the routing table.

The updates are accepted based on the metric for a particular node. The first factor determining the acceptance of an update is the sequence number. It has to accept the update if the sequence number of the update message is higher irrespective of the metric. If the update with same sequence number is received, then the update with least metric (hopCount) is given precedence. In highly mobile scenarios, there is a high chance of route fluctuations, thus we have the concept of weighted settling time where an update with change in metric will not be advertised to neighbors. The node waits for the settling time to make sure that it did not receive the update from its old neighbor before sending out that update. The current implementation covers all the above features of DSDV. The current implementation also has a request queue to buffer packets that have no routes to destination. The default is set to buffer up to 5 packets per destination.

DSR

Dynamic Source Routing (DSR) protocol is a reactive routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes. DSR operates on a on-demand behavior. Therefore, our DSR model buffers all packets while a route request packet (RREQ) is disseminated. We implement a packet buffer in dsr-rsendbuff.cc. The packet queue implements garbage collection of old packets and a queue size limit. When the packet is sent out from the send buffer, it will be queued in maintenance buffer for next hop acknowledgment.

The maintenance buffer then buffers the already sent out packets and waits for the notification of packet delivery. Protocol operation strongly depends on broken link detection mechanism. We implement the three heuristics recommended based the RFC as follows:

- First, use link layer feedback when possible, which is also the fastest mechanism of these three to detect link errors. A link is considered to be broken if frame transmission results in a transmission failure for all retries. This mechanism is meant for active links and works much faster than in its absence. DSR is able to detect the link layer transmission failure and notify that as broken. Recalculation of routes will be triggered when needed. If user does not want to use link layer acknowledgment, it can be tuned by setting "LinkAcknowledgment" attribute to false in "dsr-routing.cc".
- Second, passive acknowledgment should be used whenever possible. The node turns on "promiscuous" receive mode, in which it can receive packets not destined for itself, and when the node assures the delivery of that data packet to its destination, it cancels the passive acknowledgment timer.

- Last, use a network layer acknowledge scheme to notify the receipt of a packet. Route request packet will not be acknowledged or retransmitted.

Sample Network Topology:

This example program allows one to run ns-3 DSDV, AODV, or OLSR under a typical random waypoint mobility model. By default, the simulation runs for 200 simulated seconds, of which the first 50 are used for start-up time. The number of nodes is 50. Nodes move according to RandomWaypointMobilityModel with a speed of 20 m/s and no pause time within a 300x1500 m region. The WiFi is in ad hoc mode with a 2 Mb/s rate (802.11b) and a Friis loss model. The transmit power is set to 7.5 dBm. It is possible to change the mobility and density of the network by directly modifying the speed and the number of nodes. It is also possible to change the characteristics of the network by changing the transmit power (as power increases, the impact of mobility decreases and the effective density increases). By default, OLSR is used, but specifying a value of 2 for the protocol will cause AODV to be used, and specifying a value of 3 will cause DSDV to be used.

By default, there are 10 source/sink data pairs sending UDP data at an application rate of 2.048 Kb/s each. This is typically done at a rate of 4 64-byte packets per second. Application data is started at a random time between 50 and 51 seconds and continues to the end of the simulation.

The program outputs a few items:

- Packet receptions are notified to stdout such as:
- <timestamp> <node-id> received one packet from <src-address>
- Each second, the data reception statistics are tabulated and output to a comma-separated value (csv) file
- Some tracing and flow monitor configuration that used to work is left commented inline in the program

Sample Code for Routing Protocols:

```
#include <fstream>
#include <iostream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/aodv-module.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include "ns3/dsr-module.h"
#include "ns3/applications-module.h"
#include "ns3/yans-wifi-helper.h"

using namespace ns3;
using namespace dsr;

NS_LOG_COMPONENT_DEFINE ("manet-routing-compare");

class RoutingExperiment
{
```

```
public:
  RoutingExperiment ();
  void Run (int nSinks, double txp, std::string CSVfileName);
  //static void SetMACParam (ns3::NetDeviceContainer & devices, int slotDistance);
  std::string CommandSetup (int argc, char **argv);

private:
  Ptr<Socket> SetupPacketReceive (Ipv4Address addr, Ptr<Node> node);
  void ReceivePacket (Ptr<Socket> socket);
  void CheckThroughput ();

  uint32_t port;
  uint32_t bytesTotal;
  uint32_t packetsReceived;

  std::string m_CSVfileName;
  int m_nSinks;
  std::string m_protocolName;
  double m_txp;
  bool m_traceMobility;
  uint32_t m_protocol;
};

RoutingExperiment::RoutingExperiment ()
  : port (9),
    bytesTotal (0),
    packetsReceived (0),
    m_CSVfileName ("manet-routing.output.csv"),
    m_traceMobility (false),
    m_protocol (2) // AODV
{
}

static inline std::string
PrintReceivedPacket (Ptr<Socket> socket, Ptr<Packet> packet, Address senderAddress)
{
  std::ostringstream oss;
  oss << Simulator::Now ().GetSeconds () << " " << socket->GetNode ()->GetId ();

  if (InetSocketAddress::IsMatchingType (senderAddress))
    {
      InetSocketAddress addr = InetSocketAddress::ConvertFrom (senderAddress);
      oss << " received one packet from " << addr.GetIpv4 ();
    }
  else
    {
      oss << " received one packet!";
    }
  return oss.str ();
}
```

```
void RoutingExperiment::ReceivePacket (Ptr<Socket> socket)
{
  Ptr<Packet> packet;
  Address senderAddress;
  while ((packet = socket->RecvFrom (senderAddress)))
    {
      bytesTotal += packet->GetSize ();
      packetsReceived += 1;
      NS_LOG_UNCOND (PrintReceivedPacket (socket, packet, senderAddress));
    }
}

void RoutingExperiment::CheckThroughput ()
{
  double kbs = (bytesTotal * 8.0) / 1000;
  bytesTotal = 0;
  std::ofstream out (m_CSVfileName.c_str (), std::ios::app);
  out << (Simulator::Now ()).GetSeconds () << ","<< kbs << ","<< packetsReceived << ","
       << m_nSinks << "," << m_protocolName << "," << m_txp << ""<< std::endl;
  out.close ();
  packetsReceived = 0;
  Simulator::Schedule (Seconds (1.0), &RoutingExperiment::CheckThroughput, this);
}

Ptr<Socket> RoutingExperiment::SetupPacketReceive (Ipv4Address addr, Ptr<Node> node)
{
  TypeId tid = TypeId::LookupByName ("ns3::UdpSocketFactory");
  Ptr<Socket> sink = Socket::CreateSocket (node, tid);
  InetSocketAddress local = InetSocketAddress (addr, port);
  sink->Bind (local);
  sink->SetRecvCallback (MakeCallback (&RoutingExperiment::ReceivePacket, this));

  return sink;
}

std::string
RoutingExperiment::CommandSetup (int argc, char **argv)
{
  CommandLine cmd;
  cmd.AddValue ("CSVfileName","The name of the CSV output file name",      m_CSVfileName);
  cmd.AddValue ("traceMobility", "Enable mobility tracing", m_traceMobility);
  cmd.AddValue ("protocol", "1=OLSR;2=AODV;3=DSDV;4=DSR", m_protocol);
  cmd.Parse (argc, argv);
  return m_CSVfileName;
}

int main (int argc, char *argv[])
{
  RoutingExperiment experiment;
  std::string CSVfileName = experiment.CommandSetup (argc,argv);
  //blank out the last output file and write the column headers
```

```
 std::ofstream out (CSVfileName.c_str ());
 out << "SimulationSecond," <<  "ReceiveRate," <<  "PacketsReceived," <<    "NumberOfSinks," <<
    "RoutingProtocol," <<  "TransmissionPower" <<  std::endl;
 out.close ();
 int nSinks = 10;
 double txp = 7.5;
 experiment.Run (nSinks, txp, CSVfileName);
}

void RoutingExperiment::Run (int nSinks, double txp, std::string CSVfileName)
{
 Packet::EnablePrinting ();
 m_nSinks = nSinks;
 m_txp = txp;
 m_CSVfileName = CSVfileName;
 int nWifis = 50;
 double TotalTime = 200.0;
 std::string rate ("2048bps");
 std::string phyMode ("DsssRate11Mbps");
 std::string tr_name ("manet-routing-compare");
 int nodeSpeed = 20; //in m/s
 int nodePause = 0; //in s
 m_protocolName = "protocol";
 Config::SetDefault  ("ns3::OnOffApplication::PacketSize",StringValue ("64"));
 Config::SetDefault ("ns3::OnOffApplication::DataRate",  StringValue (rate));

 //Set Non-unicastMode rate to unicast mode
 Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode",StringValue(phyMode));
 NodeContainer adhocNodes;
 adhocNodes.Create (nWifis);
 // setting up wifi phy and channel using helpers
 WifiHelper wifi;
 wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
 YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
 YansWifiChannelHelper wifiChannel;
 wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
 wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
 wifiPhy.SetChannel (wifiChannel.Create ());

 // Add a mac and disable rate control
 WifiMacHelper wifiMac;
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager","DataMode",
                             StringValue(phyMode),"ControlMode",StringValue (phyMode));
 wifiPhy.Set ("TxPowerStart",DoubleValue (txp));
 wifiPhy.Set ("TxPowerEnd", DoubleValue (txp));
 wifiMac.SetType ("ns3::AdhocWifiMac");
 NetDeviceContainer adhocDevices = wifi.Install (wifiPhy, wifiMac, adhocNodes);

 MobilityHelper mobilityAdhoc;
 int64_t streamIndex = 0; // used to get consistent mobility across scenarios
```

```
ObjectFactory pos;
pos.SetTypeId ("ns3::RandomRectanglePositionAllocator");
pos.Set ("X", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=300.0]"));
pos.Set ("Y", StringValue ("ns3::UniformRandomVariable[Min=0.0|Max=1500.0]"));

Ptr<PositionAllocator> taPositionAlloc = pos.Create ()->GetObject<PositionAllocator> ();
streamIndex += taPositionAlloc->AssignStreams (streamIndex);

std::stringstream ssSpeed;
ssSpeed << "ns3::UniformRandomVariable[Min=0.0|Max=" << nodeSpeed << "]";
std::stringstream ssPause;
ssPause << "ns3::ConstantRandomVariable[Constant=" << nodePause << "]";
mobilityAdhoc.SetMobilityModel ("ns3::RandomWaypointMobilityModel",
                                "Speed", StringValue (ssSpeed.str ()),
                                "Pause", StringValue (ssPause.str ()),
                                "PositionAllocator", PointerValue (taPositionAlloc));
mobilityAdhoc.SetPositionAllocator (taPositionAlloc);
mobilityAdhoc.Install (adhocNodes);
streamIndex += mobilityAdhoc.AssignStreams (adhocNodes, streamIndex);
NS_UNUSED (streamIndex); // From this point, streamIndex is unused

AodvHelper aodv;
OlsrHelper olsr;
DsdvHelper dsdv;
DsrHelper dsr;
DsrMainHelper dsrMain;
Ipv4ListRoutingHelper list;
InternetStackHelper internet;

switch (m_protocol)
  {
  case 1:
    list.Add (olsr, 100);
    m_protocolName = "OLSR";
    break;
  case 2:
    list.Add (aodv, 100);
    m_protocolName = "AODV";
    break;
  case 3:
    list.Add (dsdv, 100);
    m_protocolName = "DSDV";
    break;
  case 4:
    m_protocolName = "DSR";
    break;
  default:
    NS_FATAL_ERROR ("No such protocol:" << m_protocol);
  }

if (m_protocol < 4)
```

```
  {
    internet.SetRoutingHelper (list);
    internet.Install (adhocNodes);
  }
else if (m_protocol == 4)
  {
    internet.Install (adhocNodes);
    dsrMain.Install (dsr, adhocNodes);
  }

NS_LOG_INFO ("assigning ip address");
Ipv4AddressHelper addressAdhoc;
addressAdhoc.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer adhocInterfaces;
adhocInterfaces = addressAdhoc.Assign (adhocDevices);

OnOffHelper onoff1 ("ns3::UdpSocketFactory",Address ());
onoff1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));
onoff1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));

for (int i = 0; i < nSinks; i++)
  {
    Ptr<Socket> sink = SetupPacketReceive (adhocInterfaces.GetAddress (i),
                                           adhocNodes.Get (i));
    AddressValue remoteAddress (InetSocketAddress (adhocInterfaces.GetAddress (i), port));
    onoff1.SetAttribute ("Remote", remoteAddress);
    Ptr<UniformRandomVariable> var = CreateObject<UniformRandomVariable> ();
    ApplicationContainer temp = onoff1.Install (adhocNodes.Get (i + nSinks));
    temp.Start (Seconds (var->GetValue (100.0,101.0)));
    temp.Stop (Seconds (TotalTime));
  }
std::stringstream ss;
ss << nWifis;
std::string nodes = ss.str ();

std::stringstream ss2;
ss2 << nodeSpeed;
std::string sNodeSpeed = ss2.str ();

std::stringstream ss3;
ss3 << nodePause;
std::string sNodePause = ss3.str ();

std::stringstream ss4;
ss4 << rate;
std::string sRate = ss4.str ();

//NS_LOG_INFO ("Configure Tracing.");
//tr_name = tr_name + "_" + m_protocolName +"_" + nodes + "nodes_" + sNodeSpeed + "speed_" +
    sNodePause + "pause_" + sRate + "rate";
```

```
 //AsciiTraceHelper ascii;
 AsciiTraceHelper ascii;
 MobilityHelper::EnableAsciiAll (ascii.CreateFileStream (tr_name + ".mob"));

 //Ptr<FlowMonitor> flowmon;
 //FlowMonitorHelper flowmonHelper;
 //flowmon = flowmonHelper.InstallAll ();

 NS_LOG_INFO ("Run Simulation.");
 CheckThroughput ();
 Simulator::Stop (Seconds (TotalTime));
 Simulator::Run ();
 //flowmon->SerializeToXmlFile ((tr_name + ".flowmon").c_str(), false, false);

 Simulator::Destroy ();
}
```

Steps to execute the program

1. Copy the manet-routing-compare.cc file from /path/ns-allinone-3.30/ns-3.30/examples/routing and paste the program file inside scratch folder of ns3
2. Open the terminal and change the directory to ns3
   root@path ~ #cd /path/ns-allinone-3.30/ns-3.30/
3. root@path ~ ns-3.30# ./waf --run "scratch/manet-routing-compare --protocol=3"
   //--protocol: 1=OLSR;2=AODV;3=DSDV;4=DSR

Lab Exercises:

1. Create a wireless ad-hoc network scenario that consists of 50 static nodes. The nodes are communicating using UDP and the size of a packet is 512 bytes. Vary the number of source nodes from 5, to 20 with increment of 5 and create newtrok scenario. Consider the various routing algorithms such as AODV, DSDV, and DSR to analyze the system performance. Plot the graph based on simulation results of different routing algorithms and analyze performances.
2. Create a wireless ad-hoc network scenario that consists of 50 mobile nodes. The nodes are communicating using TCP and the size of the packet is 250 bytes. Vary the number of source nodes from 5, to 20 with increment of 5 and create newtrok scenario. Consider the various routing algorithms such as AODV, DSDV, and DSR to analyze the system performance. Plot the graph based on simulation results of different routing algorithms and analyze its performance.

Additional Exercises:

1. Create a wireless network which has both static and mobile nodes. Assumne the network consists of 50 nodes where half the of nodes communicate using TCP and remaining nodes are using UDP for the communication. Simulate the same scenario for various routing protocols such as AODV, DSR, and DSDV. Compare the performances of routing protocols and plot the graph to demonstrate the same.

OBSERVATION SPACE

LAB NO: 10 **Date:**

Mobility Scenario Generation using SUMO/MOVE

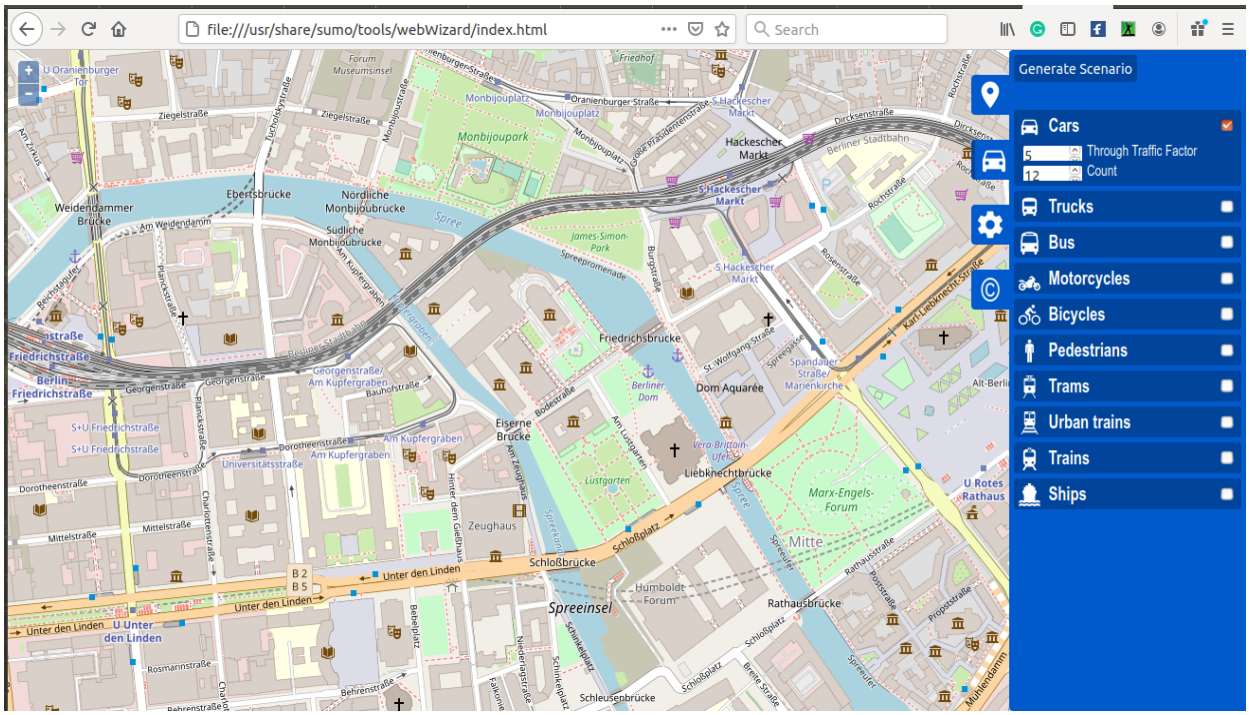
Objectives:

- Create the vehicular movement file using SUMO tool
- Configure the vehicular movement to ad-hoc nodes
- Understand the ad-hoc network and examine the performance of the network

Introduction

"Simulation of Urban MObility" (SUMO) is an open source, highly portable, microscopic and continuous traffic simulation package designed to handle large networks. It allows for intermodal simulation including pedestrians and comes with a large set of tools for scenario creation. Traffic simulation within SUMO uses software tools for simulation and analysis of road traffic and traffic management systems. New traffic strategies can be implemented via a simulation for analysis before they are used in real-world situations. SUMO has also been proposed as a tool-chain component for the development and validation of automated driving functions via various X-in-the-Loop and digital twin approaches.

SUMO is used for research purposes like traffic forecasting, evaluation of traffic lights, route selection, or in the field of vehicular communication systems. SUMO users are able to make changes to the program source code through the open-source license to experiment with new approaches.



Sample Code of Vehicular Network (Mobility):

```
#include <iostream>
#include <fstream>
#include <sstream>
#include "ns3/core-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ns2-mobility-helper.h"

using namespace ns3;

// Prints actual position and velocity when a course change event occurs
static void
CourseChange (std::ostream *os, std::string foo, Ptr<const MobilityModel> mobility)
{
  Vector pos = mobility->GetPosition (); // Get position
  Vector vel = mobility->GetVelocity (); // Get velocity

  // Prints position and velocities
  *os << Simulator::Now () << " POS: x=" << pos.x << ", y=" << pos.y
      << ", z=" << pos.z << "; VEL:" << vel.x << ", y=" << vel.y
      << ", z=" << vel.z << std::endl;
}

// Example to use ns2 traces file in ns3
int main (int argc, char *argv[])
{
  std::string traceFile;
  std::string logFile;

  int    nodeNum;
  double duration;

  // Enable logging from the ns2 helper
  LogComponentEnable ("Ns2MobilityHelper",LOG_LEVEL_DEBUG);

  // Parse command line attribute
  CommandLine cmd;
  cmd.AddValue ("traceFile", "Ns2 movement trace file", traceFile);
  cmd.AddValue ("nodeNum", "Number of nodes", nodeNum);
  cmd.AddValue ("duration", "Duration of Simulation", duration);
  cmd.AddValue ("logFile", "Log file", logFile);
  cmd.Parse (argc,argv);

  // Check command line arguments
  if (traceFile.empty () || nodeNum <= 0 || duration <= 0 || logFile.empty ())
    {
      std::cout << "Usage of " << argv[0] << " :\n\n"
```

```
    "./waf --run \"ns2-mobility-trace"
    " --traceFile=src/mobility/examples/default.ns_movements"
    " --nodeNum=2 --duration=100.0 --logFile=ns2-mob.log\" \n\n"
    "NOTE: ns2-traces-file could be an absolute or relative path. You could use the file
default.ns_movements\n"
    "      included in the same directory of this example file.\n\n"
    "NOTE 2: Number of nodes present in the trace file must match with the command line argument
and must\n"
    "       be a positive number. Note that you must know it before to be able to load it.\n\n"
    "NOTE 3: Duration must be a positive number. Note that you must know it before to be able to
load it.\n\n";

    return 0;
  }

 // Create Ns2MobilityHelper with the specified trace log file as parameter
 Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

 // open log file for output
 std::ofstream os;
 os.open (logFile.c_str ());

 // Create all nodes.
 NodeContainer stas;
 stas.Create (nodeNum);

 ns2.Install (); // configure movements for each node, while reading trace file

 // Configure callback for logging
 Config::Connect ("/NodeList/*/$ns3::MobilityModel/CourseChange",
                  MakeBoundCallback (&CourseChange, &os));

 Simulator::Stop (Seconds (duration));
 Simulator::Run ();
 Simulator::Destroy ();

 os.close (); // close log file
 return 0;
}
```

Steps to execute the program

1. Sumo software - Install this software, $sudo apt-get install sumo sumo-tools sumo-doc
2. Or Compile it from the source. Download sumo.1.2.x....tar.gz git clone command....
3. root@path ~ # cd sumo/tools
4. root@path ~ sumo/tools# python osmWebWizard.py (OSM - Open street Map)
5. Create the Sumo-gui or Sumo configuration to select the cars, buses, motorcycles, etc. and generate the scenario,
   root@path ~ sumo/tools# sumo-gui automatically pops up.

6. Create the mobility.tcl
    a. $] sumo -c osm.sumocfg --fcd-output trace.xml
    b. $] cd && cd sumo/tools
    c. $] python traceExporter.py -i trace.xml --ns2mobility-output=mobility.tcl
    d. Now check the number of nodes in the mobility.tcl file which is very important.
    e. Move the mobility.tcl in to the /home folder (/home/userfoldername/)

7. run the ns2-mobility-trace.cc file with nodeNum, duration, logFile, etc.
    a. Copy the ns2-mobility-trace.cc file from path/ns-allinone-3.30/ns-3.30/src/mobility/examples to the scratch folder.
    b. root@path ~ # cd ns-allinone-3.30/ns-3.30
    c. root@path ~ ns-allinone-3.30/ns-3.30#./waf --run "scratch/ns2-mobility-trace --traceFile=/home/userfoldername/mobility.tcl -- nodeNum=1813 --duration=100.0 --logFile=ns2-mob.log"

8. Include the NetAnim Code and run the simulation.
    a. include NetAnim Code
    b. #include "ns3/netanim-module.h" and include the following line above the Simulator::Run()
    c. AnimationInterface anim("vehicularmobility.xml");

To run the NetAnim:

root@path ~ # cd ..
root@path ~ # cd ns-allinone-3.30/netanim-3.108/
root@path ~ ns-allinone-3.30/netanim-3.108# ./NetAnim
A window will be opened and select the vehicularmobility.xml and file and run the simulation.

Lab Exercises:

1. Create a wireless ad-hoc network scenario for the Manipal location that consists of 5 bus nodes, 7 car nodes, 3 motorcycle nodes, and 4 pedestrian nodes. The nodes are using UDP with packet size 512 bytes to communicate. Display the network statistics and analyze the performances.
2. Repeate the exercise 1 for TCP.

OBSERVATION SPACE

LAB NO: 11 **Date:**

5G New Radio

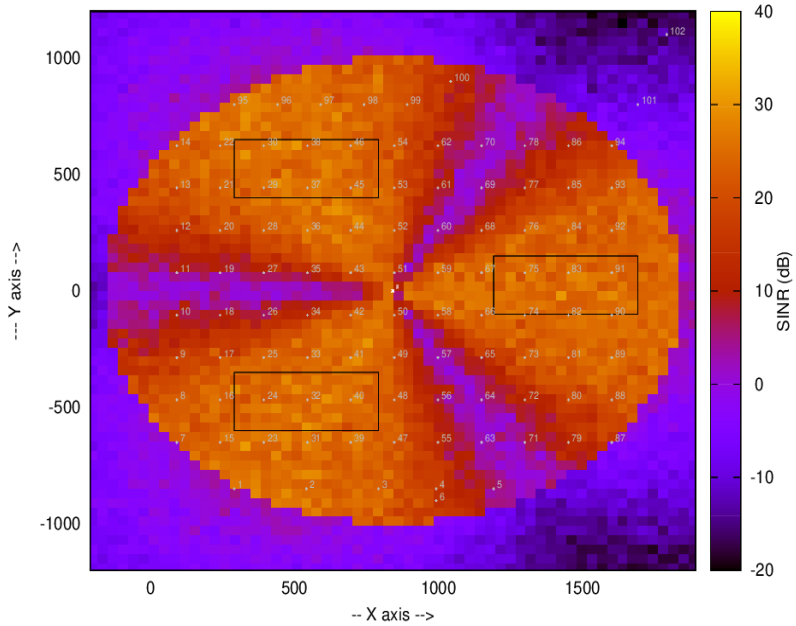
Objectives:

- Understand the 5G New Radio and its protocols
- Configure the 5G New Radio and examine the performance of the simulation scenario

Introduction

5G networks are digital cellular networks, in which the service area covered by providers is divided into small geographical areas called cells. All the 5G wireless devices in a cell communicate by radio waves with a local antenna array and low power automated transceiver (transmitter and receiver) in the cell, over frequency channels assigned by the transceiver from a pool of frequencies that are reused in other cells. The local antennas are connected with the telephone network and the Internet by a high-bandwidth optical fiber or wireless backhaul connection. As in other cell networks, a mobile device crossing from one cell to another is automatically "handed off" seamlessly to the new cell. 5G can support up to a million devices per square kilometer, while 4G supports only up to 100,000 devices per square kilometer. The new 5G wireless devices also have 4G LTE capability, as the new networks use 4G for initially establishing the connection with the cell, as well as in locations where 5G access is not available 5G will do much more than significantly improve your network connection. It provides new opportunities, enabling us to deliver groundbreaking solutions that reach across society.

Several network operators use millimeter waves for additional capacity, as well as higher throughput. Millimeter waves have a shorter range than microwaves, therefore the cells are limited to a smaller size. Millimeter waves also have more trouble passing through building walls. Millimeter wave antennas are smaller than the large antennas used in previous cellular networks. Some are only a few inches (several centimeters) long.



The NR module is a pluggable module to ns-3 that can be used to simulate 5G New Radio (NR) cellular networks. The simulator is the natural evolution of LENA, the LTE/EPC Network Simulator, but its development started from the mmWave module because it was more advanced in terms of beamforming, TDD, 3GPP channel model, and operation at FR2. It incorporates fundamental PHY-MAC NR features aligned with 3GPP NR Release-15.

Sample Code of 5G network:

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/mobility-module.h"
#include "ns3/config-store.h"
#include "ns3/mmwave-helper.h"
#include <ns3/buildings-helper.h>
#include "ns3/global-route-manager.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/log.h"

using namespace ns3;
using namespace mmwave;

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

  /* Information regarding the traces generated:
   *
   * 1. UE_1_SINR.txt : Gives the SINR for each sub-band
   *    Subframe no.  | Slot No. | Sub-band  | SINR (db)
   *
   * 2. UE_1_Tb_size.txt : Allocated transport block size
   *    Time (micro-sec)  |  Tb-size in bytes
   * */

  Ptr<MmWaveHelper> ptr_mmWave = CreateObject<MmWaveHelper> ();
  /* A configuration example.
   * ptr_mmWave->GetCcPhyParams ().at (0).GetConfigurationParameters ()-
>SetAttribute("SymbolPerSlot", UintegerValue(30)); */

  NodeContainer enbNodes;
  NodeContainer ueNodes;
  enbNodes.Create (1);
  ueNodes.Create (1);

  Ptr<ListPositionAllocator> enbPositionAlloc = CreateObject<ListPositionAllocator> ();
  enbPositionAlloc->Add (Vector (0.0, 0.0, 0.0));

  MobilityHelper enbmobility;
  enbmobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  enbmobility.SetPositionAllocator (enbPositionAlloc);
  enbmobility.Install (enbNodes);
  BuildingsHelper::Install (enbNodes);
```

```
 MobilityHelper uemobility;
 Ptr<ListPositionAllocator> uePositionAlloc = CreateObject<ListPositionAllocator> ();
 uePositionAlloc->Add (Vector (80.0, 0.0, 0.0));

 uemobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
 uemobility.SetPositionAllocator (uePositionAlloc);
 uemobility.Install (ueNodes);
 BuildingsHelper::Install (ueNodes);

 NetDeviceContainer enbNetDev = ptr_mmWave->InstallEnbDevice (enbNodes);
 NetDeviceContainer ueNetDev = ptr_mmWave->InstallUeDevice (ueNodes);

 ptr_mmWave->AttachToClosestEnb (ueNetDev, enbNetDev);
 ptr_mmWave->EnableTraces ();

 // Activate a data radio bearer
 enum EpsBearer::Qci q = EpsBearer::GBR_CONV_VOICE;
 EpsBearer bearer (q);
 ptr_mmWave->ActivateDataRadioBearer (ueNetDev, bearer);

 Simulator::Stop (Seconds (1));
 Simulator::Run ();
 Simulator::Destroy ();
 return 0;
}
```

Steps to execute the program

1. Copy the mmwave-example.cc file from /path/ns-allinone-3.30/ns-3.30/src/mmwave/example and paste the program file inside the scratch folder of ns3
2. Open the terminal and redirect the location path to ns3 folder
3. Type the command in the terminal: root@path ~ # ./waf --run scratch/mmwave-example

Lab Exercises:

1. Create a 5G network scenario that consists of one base station with three sectors. The users are placed across different scetors of the base station and vary the number of users from 10 to 40 with increment of 10 users in each scenario. Plot the radio environment map of the scenario to visualize the network.

2. Create a 5G network scenario that consists of one base station with three sectors. The 60 users are placed in one sector of the base station. Plot the radio environment map of the scenario to visualize the network.

Additional Exercises:

1. Create a 5G network scenario that consists of one base station with three sectors and 20 users. The users are moving with speed of 25 mps in their sector. Show the network animation (NetAnim) of the scenario to visualize the movement of nodes in 5G networks.

OBSERVATION SPACE

LAB NO: 12 **Date:**

5G Networks with Evolved Packet Core

Objectives:

- Understand the 5G networks and its protocols
- Configure the 5G networks and examine the performance of the simulation scenario

Introduction

The main objective of the EPC model is to provide means for the simulation of end-to-end IP connectivity over the 5G model. To this aim, it supports for the interconnection of multiple UEs to the Internet, via a radio access network of multiple eNBs connected to a single SGW/PGW node.

The following design choices have been made for the EPC model:

1. The Packet Data Network (PDN) type supported is both IPv4 and IPv6.
2. The SGW and PGW functional entities are implemented within a single node, which is hence referred to as the SGW/PGW node.
3. The scenarios with inter-SGW mobility are not of interests. Hence, a single SGW/PGW node will be present in all simulations scenarios
4. A requirement for the EPC model is that it can be used to simulate the end-to-end performance of realistic applications. Hence, it should be possible to use with the EPC model any regular ns-3 application working on top of TCP or UDP.
5. Another requirement is the possibility of simulating network topologies with the presence of multiple eNBs, some of which might be equipped with a backhaul connection with limited capabilities. In order to simulate such scenarios, the user data plane protocols being used between the eNBs and the SGW/PGW should be modeled accurately.
6. It should be possible for a single UE to use different applications with different QoS profiles. Hence, multiple EPS bearers should be supported for each UE. This includes the necessary classification of TCP/UDP traffic over IP done at the UE in the uplink and at the PGW in the downlink.
7. The focus of the EPC model is mainly on the EPC data plane. The accurate modeling of the EPC control plane is, for the time being, not a requirement; hence, the necessary control plane interactions can be modeled in a simplified way by leveraging on direct interaction among the different simulation objects via the provided helper objects.
8. The focus of the EPC model is on simulations of active users in ECM connected mode. Hence, all the functionality that is only relevant for ECM idle mode (in particular, tracking area update and paging) are not modeled at all.
9. The model should allow the possibility to perform an X2-based handover between two eNBs.

Sample Code of 5G network:

```
#include "ns3/mmwave-helper.h"
#include "ns3/epc-helper.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-helper.h"
#include "ns3/config-store.h"
#include "ns3/mmwave-point-to-point-epc-helper.h"
//#include "ns3/gtk-config-store.h"

using namespace ns3;
using namespace mmwave;

/**
 * Sample simulation script for LTE+EPC. It instantiates several eNodeB,
 * attaches one UE per eNodeB starts a flow for each UE to  and from a remote host.
 * It also  starts yet another flow between each UE pair.
 */
NS_LOG_COMPONENT_DEFINE ("EpcFirstExample");
int
main (int argc, char *argv[])
{
  uint16_t numEnb = 1;
  uint16_t numUe = 1;
  double simTime = 2.0;
  double interPacketInterval = 100;
  double minDistance = 10.0; // eNB-UE distance in meters
  double maxDistance = 150.0; // eNB-UE distance in meters
  bool harqEnabled = true;
  bool rlcAmEnabled = false;

  // Command line arguments
  CommandLine cmd;
  cmd.AddValue ("numEnb", "Number of eNBs", numEnb);
  cmd.AddValue ("numUe", "Number of UEs per eNB", numUe);
  cmd.AddValue ("simTime", "Total duration of the simulation [s])", simTime);
  cmd.AddValue ("interPacketInterval", "Inter-packet interval [us])", interPacketInterval);
  cmd.AddValue ("harq", "Enable Hybrid ARQ", harqEnabled);
  cmd.AddValue ("rlcAm", "Enable RLC-AM", rlcAmEnabled);
  cmd.Parse (argc, argv);

  Config::SetDefault ("ns3::MmWaveHelper::RlcAmEnabled", BooleanValue (rlcAmEnabled));
  Config::SetDefault ("ns3::MmWaveHelper::HarqEnabled", BooleanValue (harqEnabled));
  Config::SetDefault ("ns3::MmWaveFlexTtiMacScheduler::HarqEnabled", BooleanValue
(harqEnabled));
```

```
  Config::SetDefault ("ns3::LteRlcAm::ReportBufferStatusTimer", TimeValue (MicroSeconds
(100.0)));
  Config::SetDefault ("ns3::LteRlcUmLowLat::ReportBufferStatusTimer", TimeValue (MicroSeconds
(100.0)));

  Ptr<MmWaveHelper> mmwaveHelper = CreateObject<MmWaveHelper> ();
  mmwaveHelper->SetSchedulerType ("ns3::MmWaveFlexTtiMacScheduler");
  Ptr<MmWavePointToPointEpcHelper>  epcHelper =
CreateObject<MmWavePointToPointEpcHelper> ();
  mmwaveHelper->SetEpcHelper (epcHelper);
  mmwaveHelper->SetHarqEnabled (harqEnabled);

  ConfigStore inputConfig;
  inputConfig.ConfigureDefaults ();

  // parse again so you can override default values from the command line
  cmd.Parse (argc, argv);

  Ptr<Node> pgw = epcHelper->GetPgwNode ();

  // Create a single RemoteHost
  NodeContainer remoteHostContainer;
  remoteHostContainer.Create (1);
  Ptr<Node> remoteHost = remoteHostContainer.Get (0);
  InternetStackHelper internet;
  internet.Install (remoteHostContainer);

  // Create the Internet
  PointToPointHelper p2ph;
  p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));
  p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
  p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));
  NetDeviceContainer internetDevices = p2ph.Install (pgw, remoteHost);
  Ipv4AddressHelper ipv4h;
  ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
  Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (internetDevices);
  // interface 0 is localhost, 1 is the p2p device
  Ipv4Address remoteHostAddr = internetIpIfaces.GetAddress (1);

  Ipv4StaticRoutingHelper ipv4RoutingHelper;
  Ptr<Ipv4StaticRouting> remoteHostStaticRouting = ipv4RoutingHelper.GetStaticRouting
(remoteHost->GetObject<Ipv4> ());
  remoteHostStaticRouting->AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask ("255.0.0.0"),
1);

  NodeContainer ueNodes;
  NodeContainer enbNodes;
  enbNodes.Create (numEnb);
  ueNodes.Create (numUe);

  // Install Mobility Model
```

```
 Ptr<ListPositionAllocator> enbPositionAlloc = CreateObject<ListPositionAllocator> ();
 enbPositionAlloc->Add (Vector (0.0, 0.0, 0.0));
 MobilityHelper enbmobility;
 enbmobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
 enbmobility.SetPositionAllocator (enbPositionAlloc);
 enbmobility.Install (enbNodes);

 MobilityHelper uemobility;
 Ptr<ListPositionAllocator> uePositionAlloc = CreateObject<ListPositionAllocator> ();
 Ptr<UniformRandomVariable> distRv = CreateObject<UniformRandomVariable> ();
 for (unsigned i = 0; i < numUe; i++)
  {
   double dist = distRv->GetValue (minDistance, maxDistance);
   uePositionAlloc->Add (Vector (dist, 0.0, 0.0));
  }
 uemobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
 uemobility.SetPositionAllocator (uePositionAlloc);
 uemobility.Install (ueNodes);

 // Install mmWave Devices to the nodes
 NetDeviceContainer enbmmWaveDevs = mmwaveHelper->InstallEnbDevice (enbNodes);
 NetDeviceContainer uemmWaveDevs = mmwaveHelper->InstallUeDevice (ueNodes);

 // Install the IP stack on the UEs
 internet.Install (ueNodes);
 Ipv4InterfaceContainer ueIpIface;
 ueIpIface = epcHelper->AssignUeIpv4Address (NetDeviceContainer (uemmWaveDevs));
 // Assign IP address to UEs, and install applications
 for (uint32_t u = 0; u < ueNodes.GetN (); ++u)
  {
   Ptr<Node> ueNode = ueNodes.Get (u);
   // Set the default gateway for the UE
   Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ueNode-
>GetObject<Ipv4> ());
   ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);
  }

 mmwaveHelper->AttachToClosestEnb (uemmWaveDevs, enbmmWaveDevs);

 // Install and start applications on UEs and remote host
 uint16_t dlPort = 1234;
 uint16_t ulPort = 2000;
 uint16_t otherPort = 3000;
 ApplicationContainer clientApps;
 ApplicationContainer serverApps;
 for (uint32_t u = 0; u < ueNodes.GetN (); ++u)
  {
   ++ulPort;
   ++otherPort;
   PacketSinkHelper dlPacketSinkHelper ("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), dlPort));
```

```
    PacketSinkHelper ulPacketSinkHelper ("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), ulPort));
    PacketSinkHelper packetSinkHelper ("ns3::UdpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), otherPort));
    serverApps.Add (dlPacketSinkHelper.Install (ueNodes.Get (u)));
    serverApps.Add (ulPacketSinkHelper.Install (remoteHost));
    serverApps.Add (packetSinkHelper.Install (ueNodes.Get (u)));

    UdpClientHelper dlClient (ueIpIface.GetAddress (u), dlPort);
    dlClient.SetAttribute ("Interval", TimeValue (MicroSeconds (interPacketInterval)));
    dlClient.SetAttribute ("MaxPackets", UintegerValue (1000000));

    UdpClientHelper ulClient (remoteHostAddr, ulPort);
    ulClient.SetAttribute ("Interval", TimeValue (MicroSeconds (interPacketInterval)));
    ulClient.SetAttribute ("MaxPackets", UintegerValue (1000000));

    clientApps.Add (dlClient.Install (remoteHost));
    clientApps.Add (ulClient.Install (ueNodes.Get(u)));
  }
 serverApps.Start (Seconds (0.1));
 clientApps.Start (Seconds (0.1));
 mmwaveHelper->EnableTraces ();
 // Uncomment to enable PCAP tracing
 p2ph.EnablePcapAll ("mmwave-epc-simple");

 Simulator::Stop (Seconds (simTime));
 Simulator::Run ();
 /*GtkConfigStore config;
 config.ConfigureAttributes();*/

 Simulator::Destroy ();
 return 0;
}
```

Steps to execute the program

1. Copy the mmwave-simple-epc.cc file from /path/ns-allinone-3.30/ns-3.30/src/mmwave /example and paste the program file inside the scratch folder of ns3
2. Open the terminal and redirect the location path to ns3 folder
3. Type the command in the terminal: root@path ~ # ./waf --run scratch/ mmwave-simple-epc
4. After execution of the file, it will generate the trace file (DlPdcpStats.txt, DlRlcStats.txt)
5. Using AWK file to extract the useful information from the generated trace file to get Throughput, Delay, and Packet delivery fraction.
6. Plot the graph based on the result using GNU plot or Excel graph.

Lab Exercises:

1. Create a 5G network scenario that consists of one base station with three sectors. The users are communicating using TCP and they are placed across the sectors. Vary the number of users from 20 to 100 with increment of 20 users in each scenario to analyze the

system performance. Plot the graph based on simulation results for varied number of users.

2. Create a 5G network scenario that consists of one base station with three sectors. The users are placed across the base station sectors, and users use the UDP. Fixed the number of users to 60 users in each scenario. Vary the buffer size of each user from 10240 bytes to 51200 bytes to analyze the system performance. Plot the graph based on simulation results of different buffer sizes and analyze performance.

OBSERVATION SPACE