EXPERIMENT

Aim Introduction to computer networks lab.

A Network is a set of devices (often refered to as nodes) connected by communication links. A node can be a computer, printer, or any other device capable of sending and/or receiving data generated by other nodes on the network.

Category of Networks:

LAN (Local Area Networks): are privately owned networks within a single building or campus of up to a few kilometres in size. They are used to connect personal computers and workstations in company offices and factories to share resources and exchange information.

MAN (Metropolitan Area Network): covers a city, i.e. are networks spread across an entire city. An example might be ISP providing High speed internet to customers all over the city.

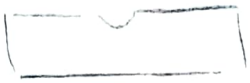
WAN (Wide Area Network): spans a wide geographical area, often a country or continent. In most wans multiple types of connections happen from wired to satellite.

Repeaters

Switches

Hubs

Routers



Bridges

Symbols for Network Devices

PAN (Personal Area Network): is a computer network for interconnecting devices centered on an individual person's workspace.

SAN

SAN (Storage Area Network) is a network that provides access to consolidated, block-level data storage.

Network Devices:

Router a type of device which acts as the central point among computers and other devices are connected to a router using network cables.

Hub: a device which connects multiple network hosts. The data that is sent to a hub is copied all across the ports and machines connected to a hub.

Switch: a device which is an intelligent hub and sends a data packet only to the intended receiver.

Bridge: a device that connects multiple networks

DELTA Pg No.
Date / /

Repeater : a device that amplifies the signal it receives, thereby increasing outreach.

Hardware Based Firewall: a device similar to a server that filters traffic to a computer. It sits between the uplink and the computer.

Networks Protocols:

TCP/IP (Transfer Control Protocol / Internet Protocol) Designed in the 1960s by the Department of Defence (DOD) in 1960s is a concise version of the OSI model. It has 4 layers application, transport, internet and network access layer. It is a loss less protocol, hence reliable.

UDP (User Datagram Protocol): a communication protocol used primarily for establishing low latency and loss tolerant connections between applications on the internet.

FTP (File Transfer Protocol): is the standard network protocol used to transfer files between a client and server on a computer network.

DELTA Pg No
Date / /

HTTP (HyperText Transfer Protocol): the transfer protocol used to transmit data on the world wide web in the form of webpages written in HTML and other web technologies

HTTPS: its the transfer protocol that forces encryption over HTTP.

Proxy & VPN:

Proxy Server is a server application that acts as an intermediary for requests from clients seeking resources from servers that provide those resources.

VPN (Virtual Private Network) extends a private network across a public network, and enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network.

Internet & WWW

Internet is a global network of networks, while the web (WWW or world wide web) is a collection of information which is accessed via the Internet.

DELTA Pg No.
Date / /

Experiment 2

Aim Introduction to discrete event simulation and installation of Network Simulator 3.

Theory

Steps to install ns3:

Packages required for installing ns3:
(i) gcc
(ii) g++
(iii) python
(iv) python-dev

Packages required for installing netanim:
(i) qt4-dev-tools

Packages required for installing PyViz:
(i) libgtk-3-dev
(ii) python-pygocanvas
(iii) python-pygraphviz

Packages required for installing Tracemetrics
(i) openjdk-8-jdk

Packages required for installing GNUplot.
(i) gnuplot

Packages required for installing wireshark

(i) WireShark

Commands to install all packages at once:

i) Ctrl + Alt + T

ii) sudo apt install gcc g++ python python-dev qt4-dev-tools libgtk-3-dev python-pygoocanvas python-pygraphite openjdk-8-jdk gnuplot wireshark

Steps to install ns3

i) Download ns-allinone-3.28.tar.bz2 and extract it

ii) CTo to ns-allinone-3.28 and give the following command in terminal

./build.py --enable-examples --enable-tests

Result NS3 is successfully installed.

DELTA Pg No.
Date / /

EXPERIMENT 3

Aim Design and implement topology of 2 nodes separated by point to point link specifying bandwidth and delay, and use UDP for transfering five packets at an interval of 1 second.

Theory

The source code implemented, firstly constituted of various header files. The core-module.h header file implements the timer functionality. Network-module.h keeps the node related data. internet-module.h manages the transmission control protocol (TCP/IP). Incorporating all the layers involved in the TCP/IP reference model. point-to-point-module.h manages the channel and application-module.h the header file only the whole experiment is based and manages User Datagram Protocol (UDP).

NS_LOG_COMPONENT_DEFINE("FirstScriptExample")
This statement declares a logging component firstly. Input-scenario creation takes place and log during runtime simulation.

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

DELTA Pg No
Date / /

NodeContainer nodes; nodes.create(2);
the node container word here provides a convenient way to create, manage and access any Node objects. The create method on the 'nodes' object and asks the container to create 2 nodes.

PointToPoint.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
PointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
The first line sets the data rate as 10 Mbps and the second line sets the delay between successive transmission as 2ms

address.setBase("10.1.1.0", "255.255.255.0");
this declares 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 allocating bits. By default addresses allocated will start at one and increase monotonically.

UdpEchoServerHelper echoServer(9);
echoServer.Install(nodes.Get(1));
It simply sets the attributes required during the execution of Helper class constructor, with port number 9.

UdpEchoClientHelper echoClient(interfaces.GetAddress(9));
This line sets the IP address and port number associated with port number 9 and so telling the helper to set the

remote address of the client to be the IP address assigned to the node on which the server resides.

echoClient.SetAttribute ("MaxPackets", UintegerValue(5));
This line defines the maximum number of packets allowed to be sent in the simulation.

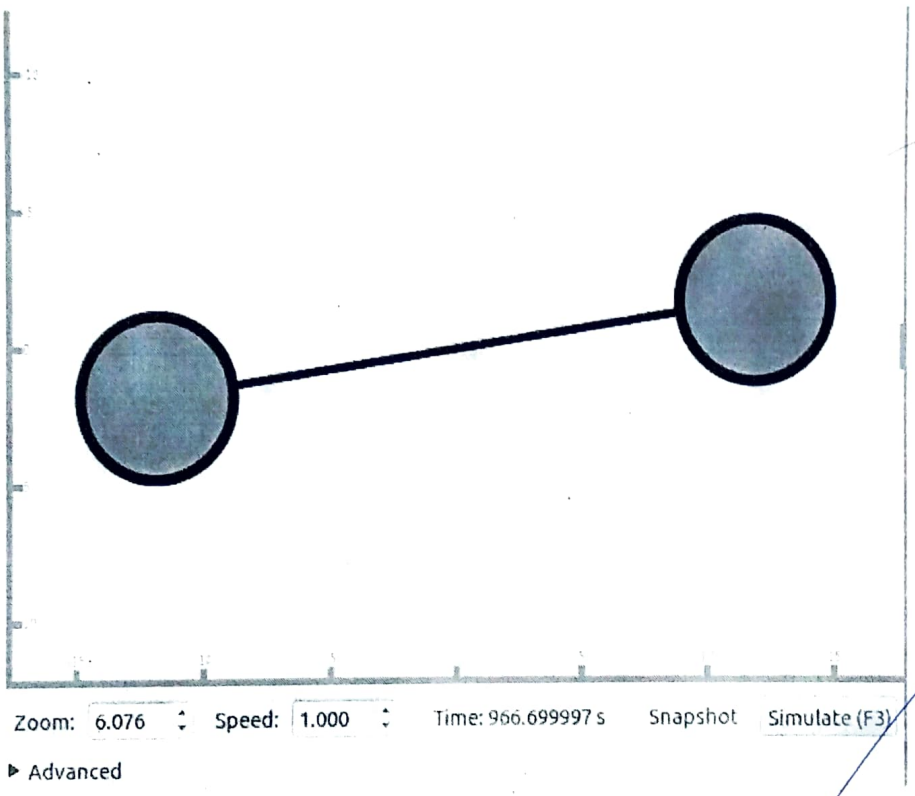
echoClient.SetAttribute ("Interval", TimeValue(Seconds(1.0)));

'Interval' attributes tells the client how long to wait between the packets to be transmitted and "PacketSize" tells the size of the packet by default 1024 byte and here five packets are sent at an interval of 1 second between 2 nodes.

Result
Packets transfered successfully with the designated time interval between the 2 nodes.

Experiment - 3

```
pc-102  pc102-ThinkCentre-M720t  ../ns-3.28.1  ./waf --run first
Entering directory `/home/pc-102/Downloads/ns-allinone-3.28.1/ns-3.28.1/build'

Leaving directory `/home/pc-102/Downloads/ns-allinone-3.28.1/ns-3.28.1/build'

Build commands will be stored in build/compile_commands.json
'build' finished successfully (4.711s)
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
```



DELTA | Pg No
Date / /

EXPERIMENT 4

Aim Design a topology of P2P (2 node) and CSMA channel (4 nodes) specifying bandwidth 5 Mbps respectively.

Theory

The ns3 CSMA device models a simple network in the spirit of Ethernet. A real Ethernet uses CSMA/CD scheme with exponentially increasing backoff to contend for the shared transmission medium. The ns3 CSMA device and channel models only a subset of this. We are going to extend our point to point example (the link between the nodes n0 and n1) by hanging a bus network off of the right side.

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");

The main program begins with a slightly different twist, we use a verbose flag to determine whether or not the UdpEchoClientApplication and UdpEchoServerApplication logging components are enabled. This flag defaults to true (the logging components are enabled) but allows us to turn off logging during regression testing of this example.

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

This step is to create 2 nodes that will connect via point to point link. The node container is used to do this just as it was done in first.cc

```
csmaNodes.Add (p2pNodes.Get(1));
CSMANodes.Create (nCsma);
```

The next line of code gets the first node from the point-to-point node container and adds it to the container of nodes that will get CSMA devices. We then create a number of "extra" nodes, that compose the remainder of the CSMA network. The number of extra nodes means the number nodes you desire in the CSMA section minus one.

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

The next bit of code is quite familiar. We instantiate a PointToPointHelper and set the associated default Attributes so that we create a 5 Mbps transmitter on devices created using the helper and a two milisecond delay on channels created by the helper.

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.10", "255.255.255.0");
IPV4InterfaceContainer p2pInterfaces;
```

We use the Ipv4AddressHelper to assign IP address to our device interface, first we use the network 10.1.10 to create the two address needed for our two point-to-point devices. We now assign IP addresses to our CSMA device interface. The operation works just as it did for the point to point case, except we now are performing the operation on a container that has a variable number of CSMA devices

Simulator::Run();
Simulator::Destroy();
return 0;

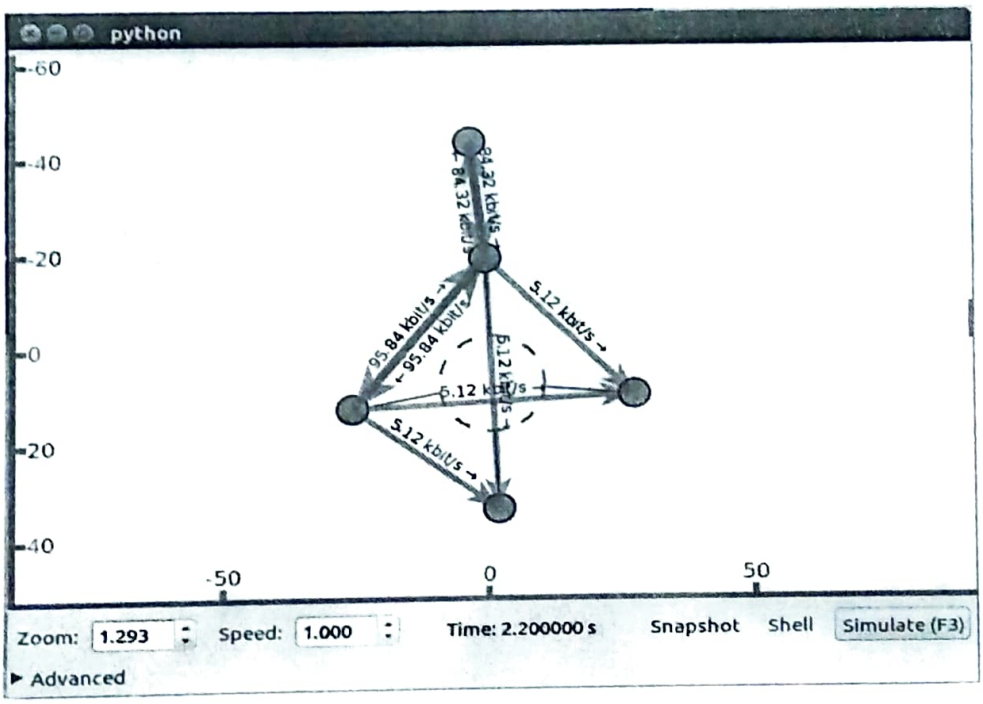
The last section of code just runs and cleans up the simulation. Just like the first cc example case we set to build just as in previous example.

Result

Topology of P2P (2 nodes) and CSMA channel (4 nodes) has been studied successfully.

EXPERIMENT - 4

```
pc-101@pc101-ThinkCentre-M720t:~/Downloads/ns-allinone-3.28.1/ns-3.28.1$ ./waf -
-run second
Waf: Entering directory `/home/pc-101/Downloads/ns-allinone-3.28.1/ns-3.28.1/bui
ld'
[2558/2742] Linking build/examples/tutorial/ns3.28.1-first-debug
Waf: Leaving directory `/home/pc-101/Downloads/ns-allinone-3.28.1/ns-3.28.1/buil
d'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (4.864s)
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.0078s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.0078s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.01761s client received 1024 bytes from 10.1.2.4 port 9
```



DELTA Pg No
Date / /

EXPERIMENT 5

Aim: Write ns 3 simulation program to show the use of topology for creating a network.

Theory

```
#Include <iostream>
# include "ns3/core-module.h"
# include "ns3/network-module.h"
# include "ns3/internet-module.h"
# include "ns3/point-to-point-module.h"
# include "ns3/netanim-module.h"
# include "ns3/applications-module.h"
# include "ns3/point-to-point-layout-module.h"

using namespace ns3;

int main(int argc, char *argv[]){
    config::setDefault("ns3::OnOffApplication::PacketSize",
                       UintegerValue(512));
    config::SetDefault("ns3::OnOffApplication::DataSize",
                       StringValue("500kbps");
    uint32_t nLeftLeaf = 5;
    uint32_t nRightLeaf = 5;
    uint32_t nLeaf = 0
    std::string animFile = "dumbbell-animation.xml"
```

```
CommandLine cmd;
cmd.AddValue ("nleftleaf", "Number of left side leaf values,
              nLeftLeaf)
cmd.AddValue ("nRightLeaf", "Number of right side left
              nodes", nRightLeaf);
cmd.AddValue ("nLeaf", "Number of left and right leaf nodes",
              nLeaf);
cmd.AddValue ("animFile", File name , animFile);

// Assign IP address
onOff Helper.clientHelper ("ns3::UdpSocketFactory",
clientHelper s0                    Address());
clientHelper.setAttribute ("OnTime", StringValue ("ns3::
                           UniformRandomVariable"));
clientHelper.setAttribute ("OffTime", stringValue ("ns3::
                           uniform RandomVariable"));
// Set the Bounding box for animation
AnimationInterface anim(animFile);
anim.EnablePacket metadata();
//Setup Actual Simulation
Simulator::Run();
std::cout << "Animation Trace file created: " << animFile.
                                                .st() << "\n";

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

DELTA Pg No.
Date / /

COMMANDS

Location of file
ns-allinone-3.28 | ns-3.28/src | netanim/examples/
dumbbell-animation.cc.

Copy file to → ns-allinone-3.28/ns-3.28/scratch

./waf -run scratch/dumbbell animation -vis
//to visualize

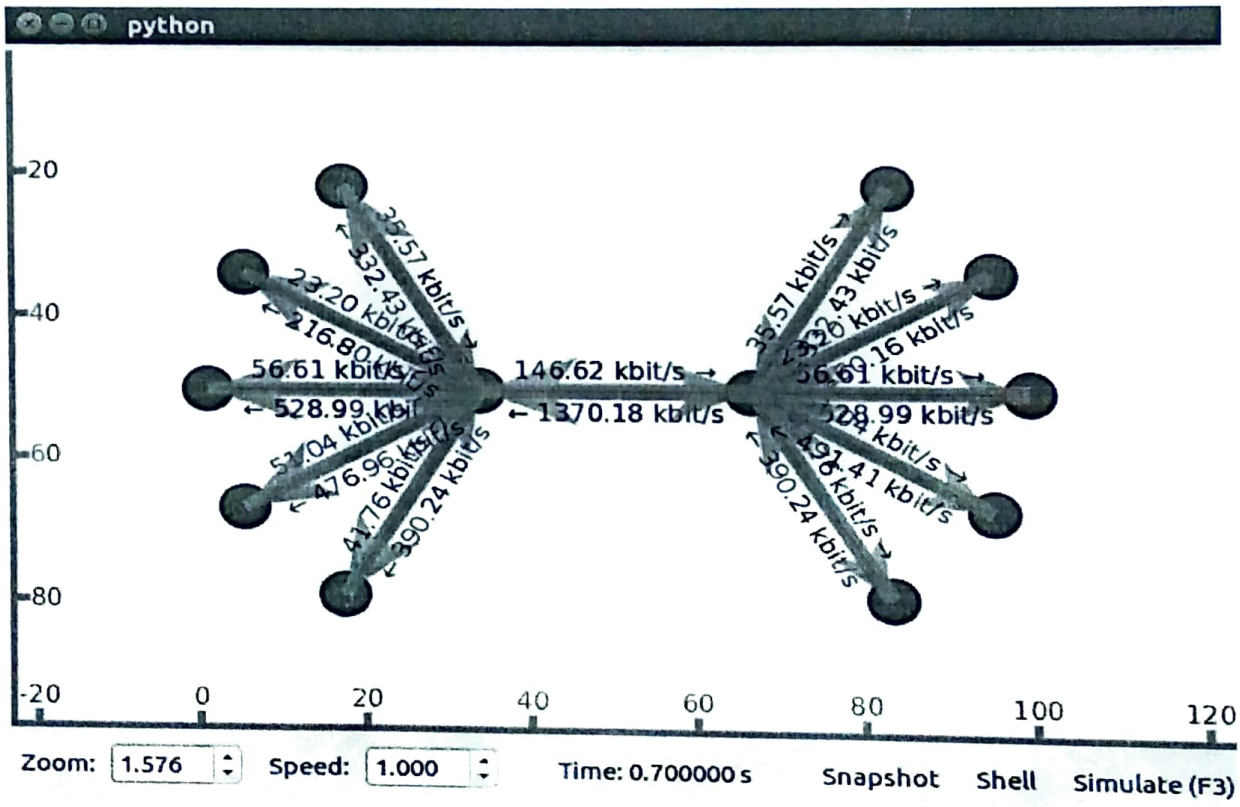
Result
An ns3 simulation simulation program was successfully created to show topology for creating a network.

EXPERIMENT NO. 5

Aim: To Design and implement LAN with various topologies and to Evaluate Network Performance. Write ns3 simulation program to show the use of topology helpers for creating a network.

Output:

```
pc-101@pc101-ThinkCentre-M720t:~/Desktop/ns-allinone-3.28/ns-3.28$ ./waf --run d
umbbell-animation
Waf: Entering directory `/home/pc-101/Desktop/ns-allinone-3.28/ns-3.28/build'
Waf: Leaving directory `/home/pc-101/Desktop/ns-allinone-3.28/ns-3.28/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (3.990s)
Animation Trace file created:dumbbell-animation.xml
```



Experiment -6

Aim : Simulate the performance of wireless networks. Introduction and implementation of wifi channel in a network using NS3 simulations.

Theory :

Multi-Channel Wi-Fi experiments are quickly becoming more common and relevant as Wi-Fi deployments have become extremely popular and dense in recent years. There are many simulation tools available for Wi-Fi simulation, but few are widely used, and some are not designed specifically with wireless in mind. In this paper, the NS3 environment is recognized as a promising tool for this type of work. Limitations in certain simulation scenarios are identified with respect to the existing NS3 wireless modules. A novel NS3 simulation module is proposed, which provides support for multi-channel Wi-Fi AP selection, so that user devices may scan several non-interfering channels and select the best AP according to IEEE 802.11 criteria. The simulation module presented is carefully studied, evaluated and validated, and is ready for use.

Wireless Networks:

• Network links are constructed on different mediums: wired and wireless.

• Wireless nodes operate untethered, assuming they have power.

• Wireless nodes may be mobile, raising a need to choose a mobility model for the simulation needs.

• Currently 802.11 based wireless models are supported.

Code :

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
// Default Network Topology
//
// Wifi 10.1.3.0
// AP
// * * * *
// | | | | 10.1.1.0
// n5 n6 n7 n0 -------------- n1 n2 n3 n4
```

```
// point-to-point | | | |
// ================
// LAN 10.1.2.0
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("ThirdScriptExample");
int main (int argc, char *argv[])
{
bool verbose = true;
uint32_t nCsma = 3;
uint32_t nWifi = 3;
bool tracing = false;
 CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
cmd.AddValue  ("verbose",  "Tell  echo  applications  to  log  if  true",
verbose);
cmd.AddValue ("tracing", "Enable pcap tracing", tracing);
cmd.Parse (argc,argv);
// The underlying restriction of 18 is due to the grid position
// allocator's configuration; the grid layout will exceed the
// bounding box if more than 18 nodes are provided.
if (nWifi > 18)
{
std::cout << "nWifi  should  be  18  or  less;  otherwise  grid  layout  exceeds
the bounding box" << std::endl;
return 1;
}

if (verbose)
{
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}

NodeContainer p2pNodes;
p2pNodes.Create (2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
```

```
NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
"Ssid", SsidValue (ssid),
"ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac",
"Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
"MinX", DoubleValue (0.0),
"MinY", DoubleValue (0.0),
"DeltaX", DoubleValue (5.0),
"DeltaY", DoubleValue (10.0),
"GridWidth", UintegerValue (3),
"LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
"Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);

InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

Ipv4AddressHelper address;
```

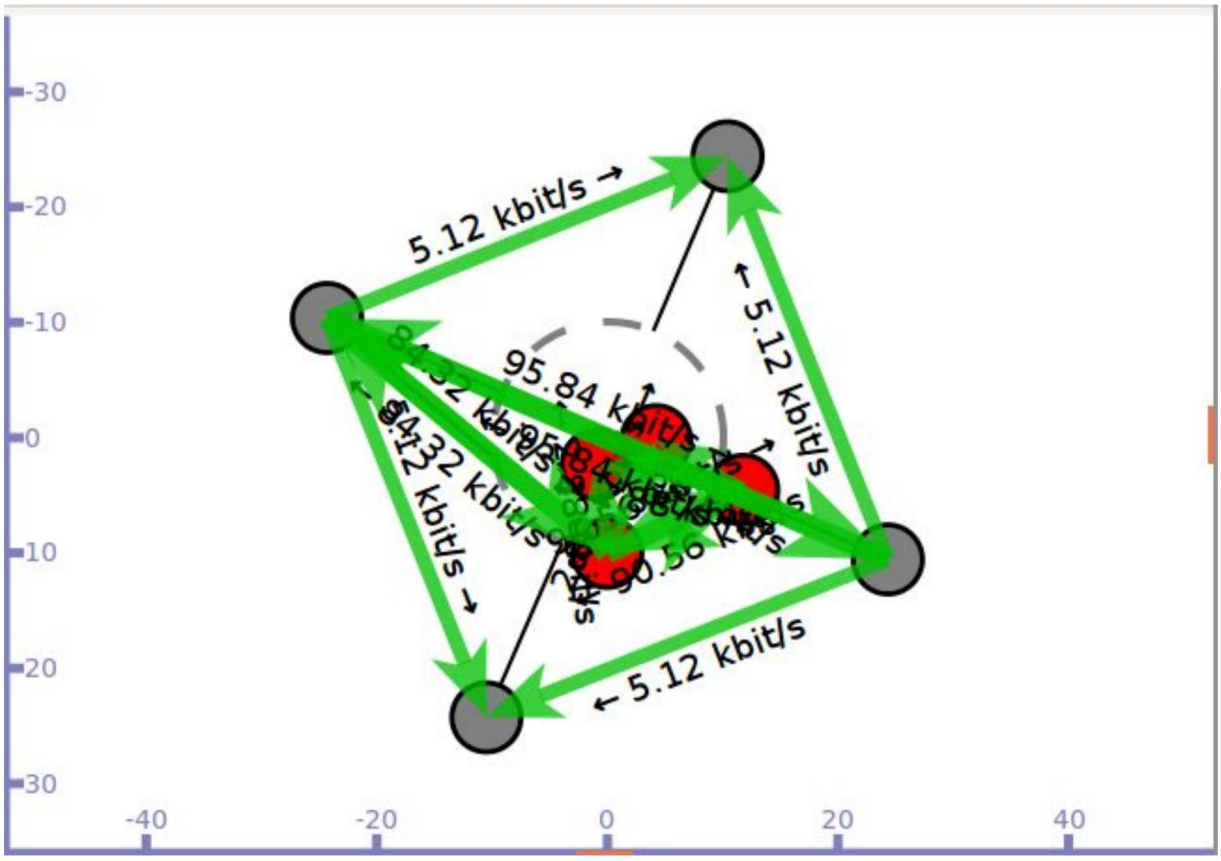
```
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);

UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get
(nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

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

ApplicationContainer clientApps =
echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

if (tracing == true)
{
pointToPoint.EnablePcapAll ("third");
phy.EnablePcap ("third", apDevices.Get (0));
csma.EnablePcap ("third", csmaDevices.Get (0), true);
}

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

Output:

Sent 1024 bytes to 10.1.2.4
Received 1024 bytes from 10.1.3.3
Received 1024 bytes from 10.1.2.4

RESULT: Successfully performed the experiment.

```
At time 2s client sent 1024 bytes to 10.1.2.4 port 9
At time 2.01794s server received 1024 bytes from 10.1.3.3 port 49153
At time 2.01794s server sent 1024 bytes to 10.1.3.3 port 49153
At time 2.03371s client received 1024 bytes from 10.1.2.4 port 9
```



EXPERIMENT - 7

Aim- Create a topology to show UDP transfer from by setting up UDPclient and UDPserver.

Software Used: NS3 Simulator

Theory:

TCP is a connection-oriented protocol layered on the top of IP of the TCP/IP stack with the ability to acknowledge receipt of packets at both ends. Acknowledgement ensures that the lost/corrupt packets can be retransmitted upon request. It also maintains a sequence in the sense that packets can be put back in the same order at the receiving end as they were transmitted.TCP can be a real bottleneck. UDP is an unreliable connectionless protocol that neither guarantees that the packets will ever reach the destination nor that they will arrive in the same order they were sent. But, it works and surprisingly reaches the destination, without the slightest aura of "guarantee" or "reliability." TCP can be best suited for file transfer or the like where loss of bits is unacceptable. UDP, on the other hand, is best suited where a little loss in the transmission bits does not matter.

- **IP Address (Who):** Every machine in the network has an IP address. This is a number of the format of something like 192.168.0.12 that uniquely identifies a host. This number gives information about ***whom*** to connect. The loopback address (denoted by 127.0.0.1) refers to oneself (same machine).
- **Port(Where):** If we know whom to connect, the port defines ***where*** to connect. It is denoted by a number ranging from 1 to 65,535. Each port provides a significant service. For example, HTTP service usually runs on port 80. When programming, care should be taken to allot a port that collides with already allocated ports.
- **Socket (How):** Socket represents the connection between two hosts and defines ***how*** to connect.

Code:

```
#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 ("UdpClientServerExample");
int
main (int argc, char *argv[])
{
//
// Enable logging for UdpClient and
//
 LogComponentEnable ("UdpClient", LOG_LEVEL_INFO);
 LogComponentEnable ("UdpServer", LOG_LEVEL_INFO);
 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 (2);
 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 one udpServer applications on node one.
//
 uint16_t port = 4000;
 UdpServerHelper server (port);
 ApplicationContainer apps = server.Install (n.Get (1));
 apps.Start (Seconds (1.0));
 apps.Stop (Seconds (10.0));
//
```

```
// Create one UdpClient application to send UDP datagrams from node zero
to
// node one.
//
 uint32_t MaxPacketSize = 1024;
 Time interPacketInterval = Seconds (0.05);
 uint32_t maxPacketCount = 320;
 UdpClientHelper client (serverAddress, port);
 client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
 client.SetAttribute ("Interval", TimeValue (interPacketInterval));
 client.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));
 apps = client.Install (n.Get (0));
 apps.Start (Seconds (2.0));
 apps.Stop (Seconds (10.0));
//
// Now, do the actual simulation.
//
 NS_LOG_INFO ("Run Simulation.");
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");
}
```

RESULT: Successfully performed the experiment.

EXPERIMENT 8

Aim: Simulating the effect of queuing disciplines on network performance - Random Early Detection.

Software Used: NS3 Simulator

Theory:

RED is a buffer management technique that attempts to provide equal access to the FIFO system by randomly dropping arriving packets before the buffer overflows. Idea is that when congestion is imminent, notify sources they should reduce their sending rates An approach to preventing unfair buffer hogging by detecting congestion when a buffer begins to reach a certain level and it notifies the source to reduce the rate at which they send packets. A dropped packet provides feedback information to the source and informs the source to reduce its transmission rate. Early drop: discard packets before buffers are full. Random drop causes some sources to reduce rate before others, causing gradual reduction in aggregate input rate.

Code:

```
/** Network topology
*
*    10Mb/s, 2ms                                10Mb/s, 4ms
* n0--------------|                          |---------------n4
*                 |   1.5Mbps/s, 20ms        |
*                n2-------------------------n3
*    10Mb/s, 3ms  |                          |    10Mb/s, 5ms
* n1--------------|                          |---------------n5
*
*
*/

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/traffic-control-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("RedTests");

uint32_t checkTimes;
double avgQueueSize;

// The times
double global_start_time;
double global_stop_time;
double sink_start_time;
```

```
double sink_stop_time;
double client_start_time;
double client_stop_time;

NodeContainer n0n2;
NodeContainer n1n2;
NodeContainer n2n3;
NodeContainer n3n4;
NodeContainer n3n5;

Ipv4InterfaceContainer i0i2;
Ipv4InterfaceContainer i1i2;
Ipv4InterfaceContainer i2i3;
Ipv4InterfaceContainer i3i4;
Ipv4InterfaceContainer i3i5;

std::stringstream filePlotQueue;
std::stringstream filePlotQueueAvg;

void
CheckQueueSize (Ptr<QueueDisc> queue)
{
 uint32_t qSize = queue->GetCurrentSize ().GetValue ();

 avgQueueSize += qSize;
 checkTimes++;

 // check queue size every 1/100 of a second
 Simulator::Schedule (Seconds (0.01), &CheckQueueSize, queue);

      std::ofstream      fPlotQueue      (filePlotQueue.str      ().c_str      (),
std::ios::out|std::ios::app);
 fPlotQueue << Simulator::Now ().GetSeconds () << " " << qSize << std::endl;
 fPlotQueue.close ();

     std::ofstream     fPlotQueueAvg     (filePlotQueueAvg.str     ().c_str     (),
std::ios::out|std::ios::app);
  fPlotQueueAvg  <<  Simulator::Now  ().GetSeconds  ()  <<  "  "  <<  avgQueueSize  /
checkTimes << std::endl;
 fPlotQueueAvg.close ();
}

void
BuildAppsTest (uint32_t test)
{
 if ( (test == 1) || (test == 3) )
   {
     // SINK is in the right side
     uint16_t port = 50000;
```

```
          Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (),
port));
     PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", sinkLocalAddress);
     ApplicationContainer sinkApp = sinkHelper.Install (n3n4.Get (1));
     sinkApp.Start (Seconds (sink_start_time));
     sinkApp.Stop (Seconds (sink_stop_time));

     // Connection one
     // Clients are in left side
     /*
      * Create the OnOff applications to send TCP to the server
      * onoffhelper is a client that send data to TCP destination
      */
     OnOffHelper clientHelper1 ("ns3::TcpSocketFactory", Address ());
                          clientHelper1.SetAttribute     ("OnTime",     StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
                         clientHelper1.SetAttribute     ("OffTime",     StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
     clientHelper1.SetAttribute
       ("DataRate", DataRateValue (DataRate ("10Mb/s")));
     clientHelper1.SetAttribute
       ("PacketSize", UintegerValue (1000));

     ApplicationContainer clientApps1;
     AddressValue remoteAddress
       (InetSocketAddress (i3i4.GetAddress (1), port));
     clientHelper1.SetAttribute ("Remote", remoteAddress);
     clientApps1.Add (clientHelper1.Install (n0n2.Get (0)));
     clientApps1.Start (Seconds (client_start_time));
     clientApps1.Stop (Seconds (client_stop_time));

     // Connection two
     OnOffHelper clientHelper2 ("ns3::TcpSocketFactory", Address ());
                          clientHelper2.SetAttribute     ("OnTime",     StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
                         clientHelper2.SetAttribute     ("OffTime",     StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
     clientHelper2.SetAttribute
       ("DataRate", DataRateValue (DataRate ("10Mb/s")));
     clientHelper2.SetAttribute
       ("PacketSize", UintegerValue (1000));

     ApplicationContainer clientApps2;
     clientHelper2.SetAttribute ("Remote", remoteAddress);
     clientApps2.Add (clientHelper2.Install (n1n2.Get (0)));
     clientApps2.Start (Seconds (3.0));
     clientApps2.Stop (Seconds (client_stop_time));
   }
 else // 4 or 5
   {
```

```
      // SINKs
      // #1
      uint16_t port1 = 50001;
          Address sinkLocalAddress1 (InetSocketAddress (Ipv4Address::GetAny (),
port1));
      PacketSinkHelper sinkHelper1 ("ns3::TcpSocketFactory", sinkLocalAddress1);
      ApplicationContainer sinkApp1 = sinkHelper1.Install (n3n4.Get (1));
      sinkApp1.Start (Seconds (sink_start_time));
      sinkApp1.Stop (Seconds (sink_stop_time));
      // #2
      uint16_t port2 = 50002;
          Address sinkLocalAddress2 (InetSocketAddress (Ipv4Address::GetAny (),
port2));
      PacketSinkHelper sinkHelper2 ("ns3::TcpSocketFactory", sinkLocalAddress2);
      ApplicationContainer sinkApp2 = sinkHelper2.Install (n3n5.Get (1));
      sinkApp2.Start (Seconds (sink_start_time));
      sinkApp2.Stop (Seconds (sink_stop_time));
      // #3
      uint16_t port3 = 50003;
          Address sinkLocalAddress3 (InetSocketAddress (Ipv4Address::GetAny (),
port3));
      PacketSinkHelper sinkHelper3 ("ns3::TcpSocketFactory", sinkLocalAddress3);
      ApplicationContainer sinkApp3 = sinkHelper3.Install (n0n2.Get (0));
      sinkApp3.Start (Seconds (sink_start_time));
      sinkApp3.Stop (Seconds (sink_stop_time));
      // #4
      uint16_t port4 = 50004;
          Address sinkLocalAddress4 (InetSocketAddress (Ipv4Address::GetAny (),
port4));
      PacketSinkHelper sinkHelper4 ("ns3::TcpSocketFactory", sinkLocalAddress4);
      ApplicationContainer sinkApp4 = sinkHelper4.Install (n1n2.Get (0));
      sinkApp4.Start (Seconds (sink_start_time));
      sinkApp4.Stop (Seconds (sink_stop_time));

      // Connection #1
      /*
       * Create the OnOff applications to send TCP to the server
       * onoffhelper is a client that send data to TCP destination
       */
      OnOffHelper clientHelper1 ("ns3::TcpSocketFactory", Address ());
                          clientHelper1.SetAttribute     ("OnTime",     StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
                         clientHelper1.SetAttribute     ("OffTime",     StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
      clientHelper1.SetAttribute
        ("DataRate", DataRateValue (DataRate ("10Mb/s")));
      clientHelper1.SetAttribute
        ("PacketSize", UintegerValue (1000));

      ApplicationContainer clientApps1;
```

```
      AddressValue remoteAddress1
        (InetSocketAddress (i3i4.GetAddress (1), port1));
      clientHelper1.SetAttribute ("Remote", remoteAddress1);
      clientApps1.Add (clientHelper1.Install (n0n2.Get (0)));
      clientApps1.Start (Seconds (client_start_time));
      clientApps1.Stop (Seconds (client_stop_time));

      // Connection #2
      OnOffHelper clientHelper2 ("ns3::TcpSocketFactory", Address ());
                          clientHelper2.SetAttribute     ("OnTime",     StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
                         clientHelper2.SetAttribute     ("OffTime",     StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
      clientHelper2.SetAttribute
        ("DataRate", DataRateValue (DataRate ("10Mb/s")));
      clientHelper2.SetAttribute
        ("PacketSize", UintegerValue (1000));

      ApplicationContainer clientApps2;
      AddressValue remoteAddress2
        (InetSocketAddress (i3i5.GetAddress (1), port2));
      clientHelper2.SetAttribute ("Remote", remoteAddress2);
      clientApps2.Add (clientHelper2.Install (n1n2.Get (0)));
      clientApps2.Start (Seconds (2.0));
      clientApps2.Stop (Seconds (client_stop_time));

      // Connection #3
      OnOffHelper clientHelper3 ("ns3::TcpSocketFactory", Address ());
                          clientHelper3.SetAttribute     ("OnTime",     StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
                         clientHelper3.SetAttribute     ("OffTime",     StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
      clientHelper3.SetAttribute
        ("DataRate", DataRateValue (DataRate ("10Mb/s")));
      clientHelper3.SetAttribute
        ("PacketSize", UintegerValue (1000));

      ApplicationContainer clientApps3;
      AddressValue remoteAddress3
        (InetSocketAddress (i0i2.GetAddress (0), port3));
      clientHelper3.SetAttribute ("Remote", remoteAddress3);
      clientApps3.Add (clientHelper3.Install (n3n4.Get (1)));
      clientApps3.Start (Seconds (3.5));
      clientApps3.Stop (Seconds (client_stop_time));

      // Connection #4
      OnOffHelper clientHelper4 ("ns3::TcpSocketFactory", Address ());
                          clientHelper4.SetAttribute     ("OnTime",     StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
```

```
                        clientHelper4.SetAttribute     ("OffTime",     StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
     clientHelper4.SetAttribute
       ("DataRate", DataRateValue (DataRate ("40b/s")));
     clientHelper4.SetAttribute
       ("PacketSize", UintegerValue (5 * 8)); // telnet

     ApplicationContainer clientApps4;
     AddressValue remoteAddress4
       (InetSocketAddress (i1i2.GetAddress (0), port4));
     clientHelper4.SetAttribute ("Remote", remoteAddress4);
     clientApps4.Add (clientHelper4.Install (n3n5.Get (1)));
     clientApps4.Start (Seconds (1.0));
     clientApps4.Stop (Seconds (client_stop_time));
   }
}

int
main (int argc, char *argv[])
{
 LogComponentEnable ("RedQueueDisc", LOG_LEVEL_INFO);

 uint32_t redTest;
 std::string redLinkDataRate = "1.5Mbps";
 std::string redLinkDelay = "20ms";

 std::string pathOut;
 bool writeForPlot = false;
 bool writePcap = false;
 bool flowMonitor = false;

 bool printRedStats = true;

 global_start_time = 0.0;
 global_stop_time = 11;
 sink_start_time = global_start_time;
 sink_stop_time = global_stop_time + 3.0;
 client_start_time = sink_start_time + 0.2;
 client_stop_time = global_stop_time - 2.0;

 // Configuration and command line parameter parsing
 redTest = 1;
 // Will only save in the directory if enable opts below
 pathOut = "."; // Current directory
 CommandLine cmd;
 cmd.AddValue ("testNumber", "Run test 1, 3, 4 or 5", redTest);
     cmd.AddValue     ("pathOut",     "Path     to     save     results     from
--writeForPlot/--writePcap/--writeFlowMonitor", pathOut);
  cmd.AddValue  ("writeForPlot",  "<0/1>  to  write  results  for  plot  (gnuplot)",
writeForPlot);
```

```
 cmd.AddValue ("writePcap", "<0/1> to write results in pcapfile", writePcap);
 cmd.AddValue ("writeFlowMonitor", "<0/1> to enable Flow Monitor and write their
results", flowMonitor);

 cmd.Parse (argc, argv);
 if ( (redTest != 1) && (redTest != 3) && (redTest != 4) && (redTest != 5) )
   {
     NS_ABORT_MSG ("Invalid test number. Supported tests are 1, 3, 4 or 5");
   }

 NS_LOG_INFO ("Create nodes");
 NodeContainer c;
 c.Create (6);
 Names::Add ( "N0", c.Get (0));
 Names::Add ( "N1", c.Get (1));
 Names::Add ( "N2", c.Get (2));
 Names::Add ( "N3", c.Get (3));
 Names::Add ( "N4", c.Get (4));
 Names::Add ( "N5", c.Get (5));
 n0n2 = NodeContainer (c.Get (0), c.Get (2));
 n1n2 = NodeContainer (c.Get (1), c.Get (2));
 n2n3 = NodeContainer (c.Get (2), c.Get (3));
 n3n4 = NodeContainer (c.Get (3), c.Get (4));
 n3n5 = NodeContainer (c.Get (3), c.Get (5));

     Config::SetDefault      ("ns3::TcpL4Protocol::SocketType",      StringValue
("ns3::TcpNewReno"));
 // 42 = headers size
 Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (1000 - 42));
 Config::SetDefault ("ns3::TcpSocket::DelAckCount", UintegerValue (1));
 GlobalValue::Bind ("ChecksumEnabled", BooleanValue (false));

 uint32_t meanPktSize = 500;

 // RED params
 NS_LOG_INFO ("Set RED params");
 Config::SetDefault ("ns3::RedQueueDisc::MaxSize", StringValue ("1000p"));
     Config::SetDefault     ("ns3::RedQueueDisc::MeanPktSize",     UintegerValue
(meanPktSize));
 Config::SetDefault ("ns3::RedQueueDisc::Wait", BooleanValue (true));
 Config::SetDefault ("ns3::RedQueueDisc::Gentle", BooleanValue (true));
 Config::SetDefault ("ns3::RedQueueDisc::QW", DoubleValue (0.002));
 Config::SetDefault ("ns3::RedQueueDisc::MinTh", DoubleValue (5));
 Config::SetDefault ("ns3::RedQueueDisc::MaxTh", DoubleValue (15));

 if (redTest == 3) // test like 1, but with bad params
   {
     Config::SetDefault ("ns3::RedQueueDisc::MaxTh", DoubleValue (10));
     Config::SetDefault ("ns3::RedQueueDisc::QW", DoubleValue (0.003));
   }
```

```
 else if (redTest == 5) // test 5, same of test 4, but in byte mode
   {
     Config::SetDefault ("ns3::RedQueueDisc::MaxSize",
                         QueueSizeValue (QueueSize (QueueSizeUnit::BYTES, 1000 *
meanPktSize)));
     Config::SetDefault ("ns3::RedQueueDisc::Ns1Compat", BooleanValue (true));
           Config::SetDefault ("ns3::RedQueueDisc::MinTh", DoubleValue (5 *
meanPktSize));
          Config::SetDefault ("ns3::RedQueueDisc::MaxTh", DoubleValue (15 *
meanPktSize));
   }

 NS_LOG_INFO ("Install internet stack on all nodes.");
 InternetStackHelper internet;
 internet.Install (c);

 TrafficControlHelper tchPfifo;
 uint16_t handle = tchPfifo.SetRootQueueDisc ("ns3::PfifoFastQueueDisc");
  tchPfifo.AddInternalQueues (handle, 3, "ns3::DropTailQueue", "MaxSize",
StringValue ("1000p"));

 TrafficControlHelper tchRed;
  tchRed.SetRootQueueDisc ("ns3::RedQueueDisc", "LinkBandwidth", StringValue
(redLinkDataRate),
                           "LinkDelay", StringValue (redLinkDelay));

 NS_LOG_INFO ("Create channels");
 PointToPointHelper p2p;

 p2p.SetQueue ("ns3::DropTailQueue");
 p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
 p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));
 NetDeviceContainer devn0n2 = p2p.Install (n0n2);
 tchPfifo.Install (devn0n2);

 p2p.SetQueue ("ns3::DropTailQueue");
 p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
 p2p.SetChannelAttribute ("Delay", StringValue ("3ms"));
 NetDeviceContainer devn1n2 = p2p.Install (n1n2);
 tchPfifo.Install (devn1n2);

 p2p.SetQueue ("ns3::DropTailQueue");
 p2p.SetDeviceAttribute ("DataRate", StringValue (redLinkDataRate));
 p2p.SetChannelAttribute ("Delay", StringValue (redLinkDelay));
 NetDeviceContainer devn2n3 = p2p.Install (n2n3);
 // only backbone link has RED queue disc
 QueueDiscContainer queueDiscs = tchRed.Install (devn2n3);

 p2p.SetQueue ("ns3::DropTailQueue");
 p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
```

```
p2p.SetChannelAttribute ("Delay", StringValue ("4ms"));
NetDeviceContainer devn3n4 = p2p.Install (n3n4);
tchPfifo.Install (devn3n4);

p2p.SetQueue ("ns3::DropTailQueue");
p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("5ms"));
NetDeviceContainer devn3n5 = p2p.Install (n3n5);
tchPfifo.Install (devn3n5);

NS_LOG_INFO ("Assign IP Addresses");
Ipv4AddressHelper ipv4;

ipv4.SetBase ("10.1.1.0", "255.255.255.0");
i0i2 = ipv4.Assign (devn0n2);

ipv4.SetBase ("10.1.2.0", "255.255.255.0");
i1i2 = ipv4.Assign (devn1n2);

ipv4.SetBase ("10.1.3.0", "255.255.255.0");
i2i3 = ipv4.Assign (devn2n3);

ipv4.SetBase ("10.1.4.0", "255.255.255.0");
i3i4 = ipv4.Assign (devn3n4);

ipv4.SetBase ("10.1.5.0", "255.255.255.0");
i3i5 = ipv4.Assign (devn3n5);

// Set up the routing
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

if (redTest == 5)
  {
    // like in ns2 test, r2 -> r1, have a queue in packet mode
    Ptr<QueueDisc> queue = queueDiscs.Get (1);

                                    StaticCast<RedQueueDisc>       (queue)->SetMode
(RedQueueDisc::QUEUE_DISC_MODE_PACKETS);
    StaticCast<RedQueueDisc> (queue)->SetTh (5, 15);
    StaticCast<RedQueueDisc> (queue)->SetQueueLimit (1000);
  }

BuildAppsTest (redTest);

if (writePcap)
  {
    PointToPointHelper ptp;
    std::stringstream stmp;
    stmp << pathOut << "/red";
    ptp.EnablePcapAll (stmp.str ().c_str ());
```

```
  }

Ptr<FlowMonitor> flowmon;
if (flowMonitor)
  {
    FlowMonitorHelper flowmonHelper;
    flowmon = flowmonHelper.InstallAll ();
  }

if (writeForPlot)
  {
    filePlotQueue << pathOut << "/" << "red-queue.plotme";
    filePlotQueueAvg << pathOut << "/" << "red-queue_avg.plotme";

    remove (filePlotQueue.str ().c_str ());
    remove (filePlotQueueAvg.str ().c_str ());
    Ptr<QueueDisc> queue = queueDiscs.Get (0);
    Simulator::ScheduleNow (&CheckQueueSize, queue);
  }

Simulator::Stop (Seconds (sink_stop_time));
Simulator::Run ();

if (flowMonitor)
  {
    std::stringstream stmp;
    stmp << pathOut << "/red.flowmon";

    flowmon->SerializeToXmlFile (stmp.str ().c_str (), false, false);
  }

if (printRedStats)
  {
    QueueDisc::Stats st = queueDiscs.Get (0)->GetStats ();
    std::cout << "*** RED stats from Node 2 queue disc ***" << std::endl;
    std::cout << st << std::endl;

    st = queueDiscs.Get (1)->GetStats ();
    std::cout << "*** RED stats from Node 3 queue disc ***" << std::endl;
    std::cout << st << std::endl;
  }

Simulator::Destroy ();

return 0;
}
```