EXPERIMENT-1

Page No.
Date 23 | 1 | 20

EXPERIMENT-1

AIM-
Introduction to various devices in networks.

THEORY -

- OSI stands for Open System Interconnection is a reference model that describes how information from a software application in one computer moves through a physical medium to the software application in another computer.

OSI consists of seven layers, and each layer performs a particular network function.

OSI model was developed by ISO in 1984 and it is now considered as an architectural model for the inter-computer communications.

OSI model divides the whole task into seven smaller and manageable tasks. Each layer is assigned a particular task.

Each layer is self-contained so that the task assigned to each layer can be performed independently.

Date

Responsibility of the Host	Application
	Presentation
	Session
	Transport
Responsibility of the Network	Network
	Data Link
	Physical

The OSI model is divided into two layers : upper layers and lower layers

The upper layer of the OSI model mainly deals with the application related issues and they are implemented only in the software. The application layer is closest to the end user. Both the end user and the application layer interact with the software applications. An upper layer refers to the layer just above another layer

⇒ Physical Layer :- The main functionality of the physical layer is to transmit the individual bits from one node to another node.

⇒ Data Link layer :- This layer is responsible for the error - free transfer of data frames. It defines the format of data on the network.

⇒ Network Layer :- It is a layer 3 that manages device addressing, tracks the location of devices on the network.

⇒ Transport Layer :- It ensures that messages are transmitted in the order in which they are sent and there is no duplication of data.

⇒ Session Layer :- It is a layer 3 in OSI model. The session layer is used to establish, maintain and synchronize the interaction between communicating devices

⇒ Presentation Layer :- It is mainly concerned with the syntax and semantics of the two systems.

⇒ Application Layer :- It serves as a window for users and application processes to access network service.

Network Devices :-

Devices used to establish and to maintain a network.

REPEATER -
It is a device used to extend LAN. These devices are used to solve the signal degradation problem. It boosts

Scanned with CamScanner

Page No.
Date

or amplifies signal before passing it through the next section of network. It has two ports that connects two network at the physical level of OSI model.

Bridge -
It is used to connect two networks or LANs using same communication protocol so that information can be passed to one another. It magnifies the data transmission signal to transfer successfully. It works on datalink layer.

Hub -
It is a multiport device used to interconnect LAN devices. In case of hub each node is connected to hub by means of simple twisted pair cable hub provides a connection over high speed.

Switch -
Switches are used to create temporary point to point link between two nodes and all data along the link. It works at the data link layer of OSI model

Router -
It is a device like switch that routes data packets based on their IP addresses. Router is mainly a

or amplifies signal before passing it through the next section of network. It has two ports that connects two network at the physical level of OSI model.

Bridge -

It is used to connect two networks or LANs using same communication protocol so that information can be passed to one another. It magnifies the data transmission signal to transfer successfully. It works on datalink layer.

Hub -

It is a multiport device used to interconnect LAN devices. In case of hub each node is connected to hub by means of simple twisted pair cable hub provides a connection over high speed.

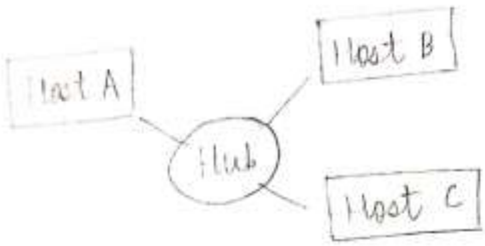
Switch -

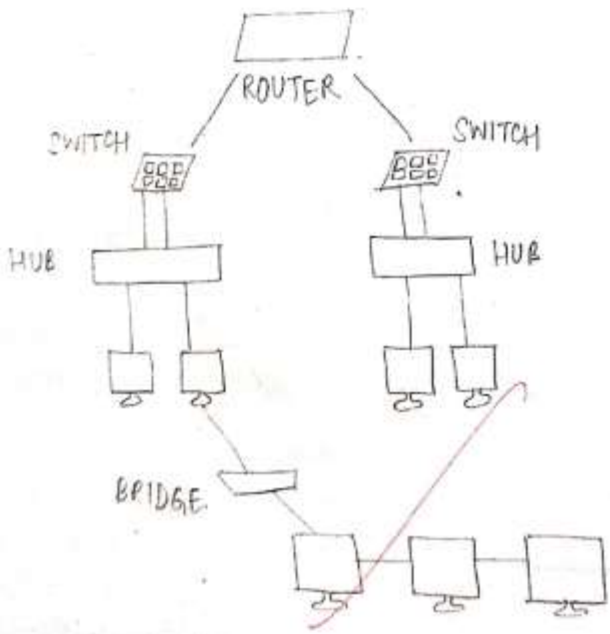
Switches are used to create temporary point to point link between two nodes and all data along the link. It works at the data link layer of OSI model

Router -

It is a device like switch that routes data packets based on their IP addresses. Router is mainly a

Scanned with CamScanner





INTERCONNECTION OF NETWORK DEVICES

Scanned with CamScanner

network layer device. It connects LANs and WANs together and have an updated routing table based on which they make decisions on routing the data packet.

RESULT-

The study of various network devices has been completed successfully.

9.5

14-5

Scanned with CamScanner

Date 23 | 1 | 20

EXPERIMENT-2

AIM -

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

SOFTWARE USED - Network Simulator 3

THEORY-

A discrete event simulation (DES) models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change in the system. Between consecutive events, no change in the system is assumed to occur.

In addition to next event time progression there is also an alternate approach called fixed increment time progression where time is broken up into small time slices and the system state is updated according to the set of events / activities happening in time slice. Because not every time slice has to be simulated, a next event time simulation can typically run much faster than a corresponding fixed increment time simulation.

NETWORK SIMULATOR - 3

ns is a name for series of discrete event networks simulator, ns-1, ns-2, ns-3. All are discrete event simulators, primarily used in research and teaching.

Design -

ns-3 is built using C++ and python with scripting capability. The ns library is wrapped by python thanks to the pybindgen library which delegates the parsing of the ns C++ headers to castxml and pygccxml to automatically generate the corresponding C++ binding glue. These C++ files are finally compiled into the ns python module to allow users to interact with the C++ ns models.

Simulation Workflow

Topology Definition : To ease the creation of basic facilities.

Model development :- Addition of models to simulation

Node and link configuration :- Models setting their default value

Scanned with CamScanner

Page No.
Date

4) Execution - Simulation facilities generate events

5) Performance Analysis - Analysed with tools like R.

6) Graphical Visualization - Graphs are plotted by Gnuplot or matplotlib.

c) Installation of ns-3

Command to install

1. sudo apt-get install gcc g++ python python-dev qt4-dev-tools libgtk-3-dev python-pygoocanvas python-pygraphviz openjdk-8-jdk wireshark

2. Download ns-allinone-3.28.tar.bz2 and unzip it.

3. Go to ns-allinone-3.28 and give the command ./build.py --enable-examples --enable-tests

Packages installed with ns-3

QT4 - - multiplatform GUI toolkit
- to animate simulations
- works with NetAnim

Scanned with CamScanner

Page no.
Date

gcc - GNU compiler collection
- compiler to C/C++/Java/ADA

libgtk-3-dev - developing packages

wireshark
- free, opensource packet analyzer
- for network troubleshooting
- originally named ethereal

RESULT -
Network Simulator 3 has been successfully installed

14

Scanned with CamScanner

EXPERIMENT - 3

Aim: Design and implement topology of 2 nodes(node 1, node 2) seperated by a point to point link specifying bandwidth & delay UDP echo apllication for tansferring 5 packets in an interval of 1 sec

Code:

```
#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"

using namespace ns3;

ns_log_component_define ("firstscriptexample");

int main (int argc, char *argv[])
{
 commandline cmd;
 cmd.parse (argc, argv);

 time::setresolution (time::ns);
 logcomponentenable ("udpechoclientapplication",
log_level_info);
 logcomponentenable ("udpechoserverapplication",
log_level_info);

 nodecontainer nodes;
 nodes.create (2);

 pointtopointhelper pointtopoint;
 pointtopoint.setdeviceattribute ("datarate",
stringvalue ("5mbps"));
 pointtopoint.setchannelattribute ("delay", stringvalue
("1ms"));

 netdevicecontainer devices;
 devices = pointtopoint.install (nodes);

 internetstackhelper stack;
 stack.install (nodes);

 ipv4addresshelper address;
 address.setbase ("10.1.1.0", "255.255.255.0");

 ipv4interfacecontainer interfaces = address.assign
(devices);

 udpechoserverhelper echoserver (9);

 applicationcontainer serverapps = echoserver.install
(nodes.get (1));
 serverapps.start (seconds (1.0));
 serverapps.stop (seconds (10.0));

 udpechoclienthelper echoclient
(interfaces.getaddress (1), 9);
 echoclient.setattribute ("maxpackets", uintegervalue
(5));
 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));

 simulator::run ();
 simulator::destroy ();
 return 0;
}
```

Output:

```
[pc-106@pc106-ThinkCentre-M720t ns-3.28]$ ./waf --run scratch/first
Waf: Entering directory `/home/pc-106/Documents/ns-allinone-3.28/ns-3.28/build'
[1842/1980] Compiling scratch/first.cc
[1955/1980] Linking build/scratch/first
Waf: Leaving directory `/home/pc-106/Documents/ns-allinone-3.28/ns-3.28/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (7.367s)
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00269s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00269s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00537s client received 1024 bytes from 10.1.1.2 port 9
At time 3s client sent 1024 bytes to 10.1.1.2 port 9
At time 3.00269s server received 1024 bytes from 10.1.1.1 port 49153
At time 3.00269s server sent 1024 bytes to 10.1.1.1 port 49153
At time 3.00537s client received 1024 bytes from 10.1.1.2 port 9
At time 4s client sent 1024 bytes to 10.1.1.2 port 9
At time 4.00269s server received 1024 bytes from 10.1.1.1 port 49153
At time 4.00269s server sent 1024 bytes to 10.1.1.1 port 49153
At time 4.00537s client received 1024 bytes from 10.1.1.2 port 9
At time 5s client sent 1024 bytes to 10.1.1.2 port 9
At time 5.00269s server received 1024 bytes from 10.1.1.1 port 49153
At time 5.00269s server sent 1024 bytes to 10.1.1.1 port 49153
At time 5.00537s client received 1024 bytes from 10.1.1.2 port 9
At time 6s client sent 1024 bytes to 10.1.1.2 port 9
At time 6.00269s server received 1024 bytes from 10.1.1.1 port 49153
At time 6.00269s server sent 1024 bytes to 10.1.1.1 port 49153
At time 6.00537s client received 1024 bytes from 10.1.1.2 port 9
```

Date 30 | 01 | 20

EXPERIMENT-3

AIM-
Design and implement topology of two nodes (Node 1, Node 2) separated by a point to point link, specifying bandwidth and delay UDP Echo application for transferring 5 packets in an interval of 1 sec.

SOFTWARE USED - Network Simulator 3

THEORY -

First.cc is a script that will create a simple point to point link between two nodes and echo a single packet between the nodes. The first line in the file is an emacs mode line. This tells emacs about the formatting conventions (coding style) we use in our source code.

/*_*- Mode : C++; c-file-style : "gnu"; indent - tabs - mode : nil ; -*-*/

Module Includes

The code proper starts with a number of include statements.

Scanned with CamScanner

Page No.
Date

include "ns3/core - module.h"
include "ns3/simulator - module.h"
include "ns3/node - module.h"
include "ns3/helper - module.h"

The ns3/core- module.h file corresponds to the ns3 module you will find in the directory src/core in downloaded release distribution

ns3 Namespace

The next line in the first.cc script is a namespace declaration.

using namespace ns3;
This groups all ns-3- related declarations in a scope outside the global namespace, which we hope will have with integration with other code. The C++ using statement introduces the ns-3 namespace into the current declarative region.

Logging

The next line of the script is following,

NS_LOG_COMPONENT_DEFINE ("First Script Example");

Scanned with CamScanner

Page No
Date

This line declares a logging component called First ScriptExample that allows you to enable and disable console message logging by reference to the name.

Main function

The next lines of the script you will find are

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

This is just the declaration of the main function of program (script) The next two lines of the script are used to enable two logging components that are built into the Echo client and Echo server applications.

Udp Echo Client Helper

The echo client application is set up in a method substantially similar to that for the server. There is an underlying UdpEchoClientApplication that is managed by an UdpEchoClientHelper. The "Maxpackets" attribute tells the client the maximum number of packets which it allows during the time of simulation

Scanned with CamScanner

Page No.	
Date	

RESULT -

The point to point link between two nodes has been successfully established using ns-3.

Scanned with CamScanner

Page No.
Date 06 02 20

EXPERIMENT - 4

AIM -

Design a topology of P2P (2 node) and CSMA channel (4 nodes) specifying bandwidth 5Mbps respectively. Study IP address assignment mechanism and pcap file generation process.

SOFTWARE USED - Network Simulator 3

THEORY -

The ns-3 CSMA device models a simple network in the spirit of ethernet. A real ethernet senses CSMA / CD (Carrier Sense Multiple Access with Collision Detection) scheme with exponentially increasing back off to contend for shared transmission medium.

By default there are three 'extra' nodes as seen below

```
// Default Network Topology
//
//    10.1.1.0
//    n0 - - - - - - - -  n1   n2  n3  n4
//    point - to - point   |    |   |   |
                          - - - - - - - - -
```

Scanned with CamScanner

LAN 10·1·2·0

//

Then the ns3 namespace is used and a logging component is defined

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Second Script Example");
```

We use a verbose flag is to determine whether or not the UdpEchoClientApplication and UdpEchoServerApplication logging components are enabled. This flag defaults to true but allows us to turn off logging during regression testing

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Next, we declare another Node Container to hold on the nodes that will be part of the bus CSMA network.

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

It gets the first node from the point to point node and adds it to the container of nodes

Scanned with CamScanner

Page No.
Date

it to the container of nodes that will get CSMA devices.

```
PointToPointHelper pointToPoint;
p2pDevices = pointToPoint.Install (p2pNodes);
```

We then instantiate a NetDeviceContainer to keep track of the point-to-point net devices and we install devices on the point to point node

The CsmaHelper works just like a PointToPointHelper but it creates and connects CSMA devices and channels.

```
InternetStackHelper stack;
stack.Install (p2pNodes.Get(0));
stack.Install (csmaNodes);
```

InternetStackHelper is used to install protocol stacks on the p2pNodes node.

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

Again, we provide required attributes to the UdpEchoClientHelper in the constructor. We tell the

Scanned with CamScanner

client to send packets to the server.

Since we have actually built an internetwork here, we need some form of internetwork routing.

Ipv4 Global Routing Helper : Populate Routing Tables ();

Next we enable pcap tracing. The first line of code enables pcap tracing and the second line enables pcap tracing in the CSMA helper.

point To Point . Enable Pcap All ("second");
csma . Enable Pcap ("second", csmaDevices . Get (1), true);

The last section of code just runs and cleans up the simulation.

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

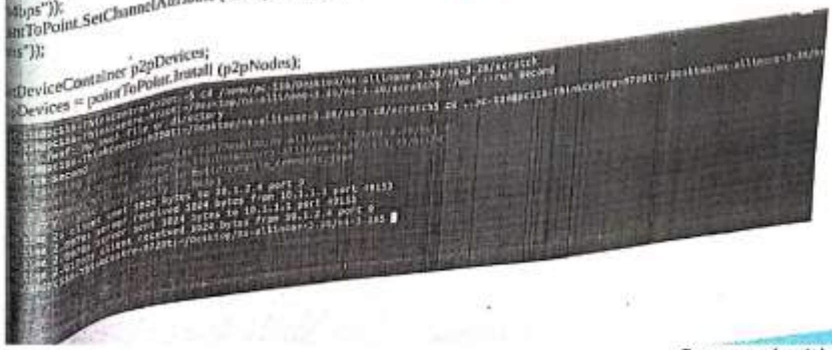
RESULT -
A topology of p2p (2 node) and CSMA channel (4 nodes) has been created.

Scanned with CamScanner

Experiment 4 - Design a topology of P2P (2 nodes) and CSMA channel (4 nodes) specifying bandwidth 5Mbps and 100Mbps respectively. Study IP address assignment mechanism and pcap file generation process.

Source Code

```
include "ns3/core-module.h"
include "ns3/network-module.h"
include "ns3/csma-module.h"
include "ns3/internet-module.h"
include "ns3/point-to-point-module.h"
include "ns3/applications-module.h"
include "ns3/ipv4-global-routing-helper.h"

Default Network Topology

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

sing namespace ns3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");
nt main (int argc, char *argv[])

bool verbose = true;
int32_t nCsma = 3;  // No. of nodes in CSMA channel (We are
dding 3 more nodes to the already existing node which is n1)

CommandLine cmd;
md.AddValue ("nCsma", "Number of \"extra\" CSMA
des/devices", nCsma);
md.AddValue ("verbose", "Tell echo applications to log if true",
rbose);

md.Parse (argc,argv);
(verbose)

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

Csma = nCsma == 0 ? 1 : nCsma;

deContainer p2pNodes;
pNodes.Create (2);
deContainer csmaNodes;
maNodes.Add (p2pNodes.Get (1)); // n1
maNodes.Create (nCsma);

ntToPointHelper pointToPoint;
ntToPoint.SetDeviceAttribute ("DataRate", StringValue
Mbps"));
ntToPoint.SetChannelAttribute ("Delay", StringValue
s"));

tDeviceContainer p2pDevices;
Devices = pointToPoint.Install (p2pNodes);

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

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

InternetStackHelper stack;
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);

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);

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
(p2pNodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

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

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



Scanned with CamScanner

EXPERIMENT – 5

Aim: Simulate the performance of wireless networks. Introduction and implementation of Wi-Fi channel in a network NS-3 Simulations.

Software used : ns3

Theory:

Characteristics of 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.

Wireless Network Construction approach in ns-3:

- Create type of nodes
  - stations
  - access points
- Create a physical layer and channel.
- Create MAC layer characteristics for node types
  - QoS or non-QoS
  - assign Service Set Identifier (SSID)
  - And other MAC related attributes.
- Install devices to nodes.
- Set-up mobility models.
- Configure Internet stack, application, and routing models.

§ Script structure: The C++ script for the network simulation has the following components:

- boilerplate: important for documentation
- module includes: include header files
- ns-3 namespace: global declaration
- logging: optional
- main function: declare main function
- topology helpers: objects to combine distinct operations
- applications: on/off application, UdpEchoClient/Server
- tracing: .tr and/or .pcap files

- simulator: start/end simulator, cleanup

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

EXPERIMENT-6

AIM: Create a topology to show udp transfer by setting up udp client and a udp server nodes

Software used: Network simulator3

Theory: The *ns-3* CSMA device models a simple network in the spirit of Ethernet. A real Ethernet uses CSMA/CD (Carrier Sense Multiple Access with Collision Detection) scheme with exponentially increasing backoff to contend for the shared transmission medium. The *ns-3* CSMA device and channel models only a subset of this.

Just as we have seen point-to-point topology helper objects when constructing point-to-point topologies, we will see equivalent CSMA topology helpers in this section. The appearance and operation of these helpers should look quite familiar to you.

We provide an example script in our examples/tutorial directory. This script builds on the first.cc script and adds a CSMA network to the point-to-point simulation we've already considered. Go ahead and open examples/tutorial/second.cc in your favorite editor. You will have already seen enough *ns-3* code to understand most of what is going on in this example, but we will go over the entire script and examine some of the output.

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.");
}
```

EXPERIMENT-7

AIM: Write NS3 simulation program to show use of topology helpers for creating a network (Dumbbell- topology).

Software used : ns3

Theory :

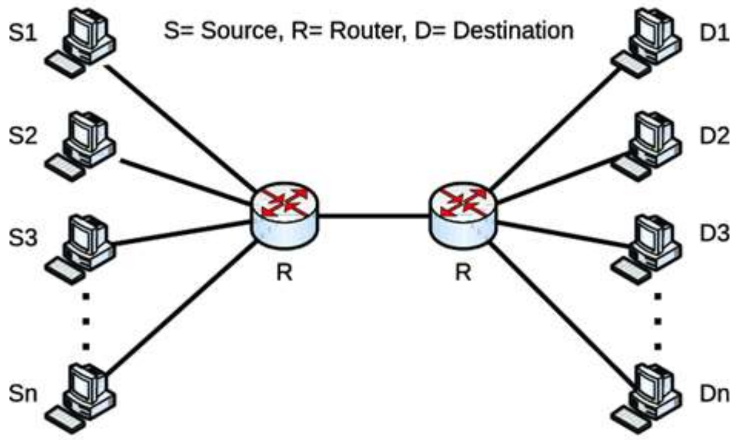
A. Topology

A Dumbbell topology is modeled with the network simulator-2 consisting of 8 hosts and 2 switches. This topology is easily scaled to different sizes. Different resources have their unique communication addresses, so here assumed that all switches has attached processor core as resources therefore treated similarly except that a traffic generator can be attached to resources. Switch, resource and link are three basic elements in the topology. Assume that the each resources has infinite buffer size but finite in switches. It means that the packet being dropped or lost cannot occur in resources but only take place in switches.

B. Communication Links
An inter-communication path between the switches is composed of links. Each node is connected with point-to-point bidirectional links. The bandwidth and latency of the link is configurable. When any link down between two nodes it implies that the packet cannot be travel between these nodes in any direction. Because bidirectional links are actually implemented using a single wire.

C. Communication traffic

The traffic flow in the dumbbell network is simulated and analyzed using different TCP variants such as: TCP Reno, TCP New Reno, HSTCP and STCP. The results and analysis are depicted in the next section.



Command line:

Copy dumbbell-animation.cc to scratch and run the following commands at the terminal.

1. Waf --run scratch/dumbbell-animation-vis

   or you can set left right nodes equal by giving following commands

2. ./waf --run “scratch/dumbbell-animation --nLeaf=10” -vis

   Or define number separately using

3. ./waf --run “scratch/dumbbell-animation --nLeftLeaf=10 --RightLeaf=5” --vis

<u>Code :</u>

```
#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::DataRate", StringValue ("500kb/s"));
```

```
uint32_t nLeftLeaf = 5;
uint32_t nRightLeaf = 5;
uint32_t nLeaf = 0; // If non-zero, number of both left and right
std::string animFile = "dumbbell-animation.xml" ; // Name of file for animation output

CommandLine cmd;
cmd.AddValue ("nLeftLeaf", "Number of left side leaf nodes", nLeftLeaf);
cmd.AddValue ("nRightLeaf","Number of right side leaf nodes", nRightLeaf);
cmd.AddValue ("nLeaf", "Number of left and right side leaf nodes", nLeaf);
cmd.AddValue ("animFile", "File Name for Animation Output", animFile);
cmd.Parse (argc,argv);
if (nLeaf > 0)
{
nLeftLeaf = nLeaf;
nRightLeaf = nLeaf;
}

// Create the point-to-point link helpers
PointToPointHelper pointToPointRouter;
pointToPointRouter.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
pointToPointRouter.SetChannelAttribute ("Delay", StringValue ("1ms"));
PointToPointHelper pointToPointLeaf;
pointToPointLeaf.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
pointToPointLeaf.SetChannelAttribute ("Delay", StringValue ("1ms"));

PointToPointDumbbellHelper d (nLeftLeaf, pointToPointLeaf,
nRightLeaf, pointToPointLeaf,
pointToPointRouter);

// Install Stack
InternetStackHelper stack;
d.InstallStack (stack);

// Assign IP Addresses
d.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"),
Ipv4AddressHelper ("10.2.1.0", "255.255.255.0"),
Ipv4AddressHelper ("10.3.1.0", "255.255.255.0"));

// Install on/off app on all right side nodes
OnOffHelper clientHelper ("ns3::UdpSocketFactory", Address ());
clientHelper.SetAttribute ("OnTime", StringValue ("ns3::UniformRandomVariable"));
```

```
 clientHelper.SetAttribute ("OffTime", StringValue ("ns3::UniformRandomVariable"));
 ApplicationContainer clientApps;

for (uint32_t i = 0; i < ((d.RightCount () < d.LeftCount ()) ? d.RightCount () : d.LeftCount ()); ++i)
 {
 // Create an on/off app sending packets to the same leaf right side
 AddressValue remoteAddress (InetSocketAddress (d.GetLeftIpv4Address (i), 1000));
 clientHelper.SetAttribute ("Remote", remoteAddress);
clientApps.Add (clientHelper.Install (d.GetRight (i)));
 }

clientApps.Start (Seconds (0.0));
 clientApps.Stop (Seconds (10.0));

 // Set the bounding box for animation
 d.BoundingBox (1, 1, 100, 100);

 // Create the animation object and configure for specified output
 AnimationInterface anim (animFile);
 anim.EnablePacketMetadata (); // Optional
 anim.EnableIpv4L3ProtocolCounters (Seconds (0), Seconds (10)); // Optional

// Set up the actual simulation
 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Run ();
 std::cout << "Animation Trace file created:" << animFile.c_str ()<< std::endl;
 Simulator::Destroy ();
return 0;
}
```

EXPERIMENT-8

Aim: Write NS3 simulation program to implement Ad hoc Network.

Software used: ns3

Theory:

Code:

```
#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"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("AdhocScriptExample");
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);
  std::string animFile = "adhoc-animation1.xml" ;
  Time::SetResolution (Time::NS);
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  NodeContainer nodes;
  nodes.Create (2);
  YansWifiChannelHelper chl = YansWifiChannelHelper::Default();
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
```

```
 phy.SetChannel(chl.Create());  WifiHelper wifi;
 wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager","DataMode",StringValue(
"DsssRate2Mbps"),"ControlMode",StringValue("DsssRate1Mbps"));
 WifiMacHelper mac;
 mac.SetType("ns3::AdhocWifiMac");
 NetDeviceContainer devices;
 devices = wifi.Install (phy,mac,nodes);
 MobilityHelper mobility;
 mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
 mobility.Install (nodes);
 InternetStackHelper stack;
 stack.Install (nodes);
 Ipv4AddressHelper address;
 address.SetBase ("10.1.1.0", "255.255.255.0");
 Ipv4InterfaceContainer interfaces = address.Assign (devices);
 UdpEchoServerHelper echoServer (9);
 ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
 serverApps.Start (Seconds (1.0));
 serverApps.Stop (Seconds (10.0));
 UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
 echoClient.SetAttribute ("MaxPackets", UintegerValue (4));
 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));
 AnimationInterface anim (animFile);
 anim.EnablePacketMetadata (true);
 AnimationInterface::SetConstantPosition(nodes.Get(0),10,25);
 AnimationInterface::SetConstantPosition(nodes.Get(1),40,25);
 anim.EnableIpv4L3ProtocolCounters (Seconds (0), Seconds (10));
 Simulator::Run ();
 std::cout << "Animation Trace file created:" << animFile.c_str ()<< std::endl;
 Simulator::Destroy ();
 return 0;}
```

OUTPUT:

At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.01543s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.01543s server sent 1024 bytes to 10.1.1.1 port 49153

At time 2.02795s client received 1024 bytes from 10.1.1.2 port 9
At time 3s client sent 1024 bytes to 10.1.1.2 port 9
At time 3.00454s server received 1024 bytes from 10.1.1.1 port 49153
At time 3.00454s server sent 1024 bytes to 10.1.1.1 port 49153
At time 3.0099s client received 1024 bytes from 10.1.1.2 port 9
At time 4s client sent 1024 bytes to 10.1.1.2 port 9
At time 4.00454s server received 1024 bytes from 10.1.1.1 port 49153
At time 4.00454s server sent 1024 bytes to 10.1.1.1 port 49153
At time 4.0097s client received 1024 bytes from 10.1.1.2 port 9
At time 5s client sent 1024 bytes to 10.1.1.2 port 9

EXPERIMENT-9

Aim -To Write an NS3 simulation program to implement Mesh Topology

Software used : ns3

Theory:

Mesh topology is a type of networking where all nodes cooperate to distribute data amongst each other.Mesh systems usually rely on a *routing table*, which tells every node
(a) how to communicate with the access point, and
(b) how a node should direct traffic that is trying to go somewhere. The routing table assumes that there is not direct communication anywhere in the network except by nodes that have a route to the access point.
Routing tables are comprised of:

1. Source identifier
2. Destination identifier
3. Source sequence number
4. Destination sequence number
5. Broadcast identifier

Here each node is capable of sending messages to and receiving messages from other nodes. The nodes act as relays, passing on a message towards its final destination.
Mesh networks are becoming increasingly popular due to their efficiency.

There are two types of mesh topology:

- full mesh topology
- partial mesh topology

In full mesh, each node is directly connected to every other node. This enables a message to be sent along many individual routes.

In a partial mesh, not all nodes are connected directly to each other. A partial mesh therefore has fewer routes for a message to travel along than a full mesh but is simpler to implement.

Advantages -

1. messages can be received more quickly if the route to the intended recipient is short.
2. messages should always get through as they have many possible routes on which to travel.
3. multiple connections mean (in theory) that no node should be isolated.
4. multiple connections mean each node can transmit to and receive from more than one node at the same time.
5. new nodes can be added without interruption or interfering with other nodes.

Disadvantages –

- full mesh networks can be impractical to set up because of the high number of connections needed
- many connections require a lot of maintenance

code:

```
#include <iostream>
#include <sstream>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mesh-module.h"
#include "ns3/mobility-module.h"
#include "ns3/mesh-helper.h"
#include "ns3/yans-wifi-helper.h"

using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("TestMeshScript");

class MeshTest
{
 public:
  MeshTest ();
  void Configure (int argc, char ** argv);
```

```
 int Run ();
 private:
 int       m_xSize;
 int       m_ySize;
 double    m_step;
 double    m_randomStart;
 double    m_totalTime;
 double    m_packetInterval;
 uint16_t  m_packetSize;
 uint32_t  m_nIfaces;
 bool      m_chan;
 bool      m_pcap;
 bool      m_ascii;
 std::string m_stack;
 std::string m_root;
 NodeContainer nodes;
 NetDeviceContainer meshDevices;
 Ipv4InterfaceContainer interfaces;
 MeshHelper mesh;
 private:
 void CreateNodes ();
 void InstallInternetStack ();
 void InstallApplication ();
 void Report ();
};
MeshTest::MeshTest () :
 m_xSize (3),
 m_ySize (3),
 m_step (100.0),
 m_randomStart (0.1),
 m_totalTime (100.0),
 m_packetInterval (0.1),
 m_packetSize (1024),
 m_nIfaces (1),
 m_chan (true),
 m_pcap (false),
 m_ascii (false),
 m_stack ("ns3::Dot11sStack"),
 m_root ("ff:ff:ff:ff:ff:ff")
}
void
MeshTest::Configure (int argc, char *argv[])
{
 CommandLine cmd;
 cmd.AddValue ("x-size", "Number of nodes in a row grid", m_xSize);
 cmd.AddValue ("y-size", "Number of rows in a grid", m_ySize);
 cmd.AddValue ("step",   "Size of edge in our grid (meters)", m_step);
```

```
  // Avoid starting all mesh nodes at the same time (beacons may collide)
  cmd.AddValue ("start",  "Maximum random start delay for beacon jitter (sec)",  m_randomStart);
  cmd.AddValue ("time",  "Simulation time (sec)", m_totalTime);
  cmd.AddValue ("packet-interval",  "Interval between packets in UDP ping (sec)", m_packetInterval);
  cmd.AddValue ("packet-size",  "Size of packets in UDP ping (bytes)", m_packetSize);
  cmd.AddValue ("interfaces", "Number of radio interfaces used by each mesh point", m_nIfaces);
  cmd.AddValue ("channels",   "Use different frequency channels for different interfaces", m_chan);
  cmd.AddValue ("pcap",   "Enable PCAP traces on interfaces", m_pcap);
  cmd.AddValue ("ascii",   "Enable Ascii traces on interfaces", m_ascii);
  cmd.AddValue ("stack",  "Type of protocol stack. ns3::Dot11sStack by default", m_stack);
  cmd.AddValue ("root", "Mac address of root mesh point in HWMP", m_root);

  cmd.Parse (argc, argv);
  NS_LOG_DEBUG ("Grid:" << m_xSize << "*" << m_ySize);
  NS_LOG_DEBUG ("Simulation time: " << m_totalTime << " s");
  if (m_ascii)
    {
      PacketMetadata::Enable ();
    }
}
void
MeshTest::CreateNodes ()
{
  nodes.Create (m_ySize*m_xSize);
  // Configure YansWifiChannel
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  wifiPhy.SetChannel (wifiChannel.Create ());
  mesh = MeshHelper::Default ();
  if (!Mac48Address (m_root.c_str ()).IsBroadcast ())
    {
      mesh.SetStackInstaller (m_stack, "Root", Mac48AddressValue (Mac48Address (m_root.c_str ())));
    }
  else
    {
      //If root is not set, we do not use "Root" attribute, because it
      //is specified only for 11s
      mesh.SetStackInstaller (m_stack);
    }
  if (m_chan)
    {
      mesh.SetSpreadInterfaceChannels (MeshHelper::SPREAD_CHANNELS);
    }
  else
    {
      mesh.SetSpreadInterfaceChannels (MeshHelper::ZERO_CHANNEL);
```

```
    }
  mesh.SetMacType ("RandomStart", TimeValue (Seconds (m_randomStart)));
  // Set number of interfaces - default is single-interface mesh point
  mesh.SetNumberOfInterfaces (m_nIfaces);
  // Install protocols and return container if MeshPointDevices
  meshDevices = mesh.Install (wifiPhy, nodes);
  // Setup mobility - static grid topology
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                 "MinX", DoubleValue (0.0),
                                 "MinY", DoubleValue (0.0),
                                 "DeltaX", DoubleValue (m_step),
                                 "DeltaY", DoubleValue (m_step),
                                 "GridWidth", UintegerValue (m_xSize),
                                 "LayoutType", StringValue ("RowFirst"));
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (nodes);
 if (m_pcap)
    wifiPhy.EnablePcapAll (std::string ("mp-"));
  if (m_ascii)
    {
      AsciiTraceHelper ascii;
      wifiPhy.EnableAsciiAll (ascii.CreateFileStream ("mesh.tr"));
    }
}
void
MeshTest::InstallInternetStack ()
{
  InternetStackHelper internetStack;
  internetStack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  interfaces = address.Assign (meshDevices);
}
void
MeshTest::InstallApplication ()
{
  UdpEchoServerHelper echoServer (9);
  ApplicationContainer serverApps = echoServer.Install (nodes.Get (0));
  serverApps.Start (Seconds (0.0));
  serverApps.Stop (Seconds (m_totalTime));
  UdpEchoClientHelper echoClient (interfaces.GetAddress (0), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue ((uint32_t)(m_totalTime*
(1/m_packetInterval))));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (m_packetInterval)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (m_packetSize));
  ApplicationContainer clientApps = echoClient.Install (nodes.Get (m_xSize*m_ySize-1));
```

```
  clientApps.Start (Seconds (0.0));
  clientApps.Stop (Seconds (m_totalTime));
}
int
MeshTest::Run ()
{
  CreateNodes ();
  InstallInternetStack ();
  InstallApplication ();
  Simulator::Schedule (Seconds (m_totalTime), &MeshTest::Report, this);
  Simulator::Stop (Seconds (m_totalTime));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
void
MeshTest::Report ()
{
  unsigned n (0);
  for (NetDeviceContainer::Iterator i = meshDevices.Begin (); i != meshDevices.End (); ++i, ++n)
   {
     std::ostringstream os;
     os << "mp-report-" << n << ".xml";
     std::cerr << "Printing mesh point device #" << n << " diagnostics to " << os.str () << "\n";
     std::ofstream of;
     of.open (os.str ().c_str ());
     if (!of.is_open ())
       {
         std::cerr << "Error: Can't open file " << os.str () << "\n";
         return;
       }
     mesh.Report (*i, of);
     of.close ();
    }
}
int
main (int argc, char *argv[])
{
  MeshTest t;
  t.Configure (argc, argv);
  return t.Run  ();
}
```

EXPERIMENT-10

Aim:- Write an NS3 simulation program to implement Star Topology.

Software used : ns3

Theory: A star topology is a topology for a Local Area Network (LAN) in which all nodes are individually connected to a central connection point, like a hub or a switch. A star takes more cable than e.g. a bus, but the benefit is that if a cable fails, only one node will be brought down.

All traffic emanates from the hub of the star. The central site is in control of all the nodes attached to it. The central hub is usually a fast, self contained computer and is responsible for routing all traffic to other nodes. The main advantages of a star network is that one malfunctioning node does not affect the rest of the network. However this type of network can be prone to bottleneck and failure problems at the central site.

Advantages of star topology

- Centralized management of the network, through the use of the central computer, hub, or switch.
- Easy to add another computer to the network.
- If one computer on the network fails, the rest of the network continues to function normally.

Disadvantages of star topology

- May have a higher cost to implement, especially when using a switch or router as the central network device.
- The central network device determines the performance and number of nodes the network can handle.
- If the central computer, hub, or switch fails, the entire network goes down and all computers are disconnected from the network.

Code:

```
#include "ns3/netanim-module.h"
 #include "ns3/internet-module.h"
 #include "ns3/point-to-point-module.h"
 #include "ns3/applications-module.h"
#include "ns3/point-to-point-layout-module.h"


using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("Star");
int
main (int argc, char *argv[])
{
Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (137));
 Config::SetDefault("ns3::OnOffApplication::DataRate",StringValue ("14kb/s"));
uint32_t nSpokes = 8;

 CommandLine cmd;
cmd.AddValue ("nSpokes", "Number of nodes to place in the star", nSpokes);
cmd.Parse (argc, argv);
NS_LOG_INFO ("Build star topology.");
 PointToPointHelper pointToPoint;
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.");

  uint16_t port = 50000;
 AddresshubLocalAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
 PacketSinkHelperpacketSinkHelper("ns3::TcpSocketFactory", hubLocalAddress);
ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());
hubApp.Start (Seconds (1.0));
 hubApp.Stop (Seconds (10.0));

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));

 Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
 NS_LOG_INFO ("Enable pcap tracing.");
 pointToPoint.EnablePcapAll ("star");
NS_LOG_INFO ("Run Simulation.");
 Simulator::Run ();
Simulator::Destroy ();
 NS_LOG_INFO ("Done.");
 return 0;}
```