**Practical No 5**

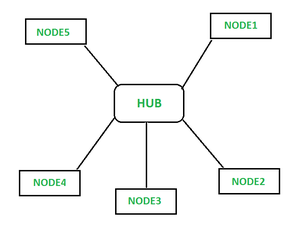
**1.Aim: -** WAP to simulate Star topology.

**2.Objective: -**

* Understanding of Star topology
* Implementation of Star topology using NS3

**3.Theory: -**

Star Topology A star may be a topology for a Local Area Network (LAN) during which all nodes are individually connected to a central connection point, sort of a hub or a switch. A star takes more cable than e.g. a bus, but the benefit is that if a cable fails, just one node is going to be brought down. Each device within the network is connected to a central device called a hub. If one device wants to send data to another device, it’s to first send the info to the hub then the hub transmits that data to the designated device. The number of links required to connect nodes in the star topology is N where N is the number of nodes.



**Advantages of Star topology:**

* It is very reliable – if one cable or device fails then all the others will still work
* It is high-performing as no data collisions can occur
* Less expensive because each device only needs one I/O port and wishes to be connected with hub with one link.
* Easier to put in
* Robust in nature

**Disadvantages of Star topology:**

* Requires more cable than a linear bus.
* If the connecting network device (network switch) fails, nodes attached are disabled and can’t participate in network communication.
* More expensive than linear bus topology due to the value of the connecting devices (network switches)
* If hub goes down everything goes down, none of the devices can work without hub.
* Hub requires more resources and regular maintenance because it’s the central system of star.

**4.Program: -**

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#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"

// Network topology (default)

//

// n2 n3 n4 .

// \ | / .

// \|/ .

// n1--- n0---n5 .

// /|\ .

// / | \ .

// n8 n7 n6 .

//

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("StarAnimation");

int

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

{

//

// Set up some default values for the simulation.

//

Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (137));

// ??? try and stick 15kb/s into the data rate

Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue

("14kb/s"));

//

// Default number of nodes in the star. Overridable by command line argument.

//

uint32\_t nSpokes = 8;

std::string animFile = "star-animation.xml";

uint8\_t useIpv6 = 0;

Ipv6Address ipv6AddressBase = Ipv6Address("2001::");

Ipv6Prefix ipv6AddressPrefix = Ipv6Prefix(64);

CommandLine cmd;

cmd.AddValue ("nSpokes", "Number of spoke nodes to place in the star",

nSpokes);

cmd.AddValue ("animFile", "File Name for Animation Output", animFile);

cmd.AddValue ("useIpv6", "use Ipv6", useIpv6);

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.");

if (useIpv6 == 0)

{

star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));

}

else

{

star.AssignIpv6Addresses (ipv6AddressBase, ipv6AddressPrefix);

}

NS\_LOG\_INFO ("Create applications.");

//

// Create a packet sink on the star "hub" to receive packets.

//

uint16\_t port = 50000;

Address hubLocalAddress;

if (useIpv6 == 0)

{

hubLocalAddress = InetSocketAddress (Ipv4Address::GetAny (), port);

}

else

{

hubLocalAddress = Inet6SocketAddress (Ipv6Address::GetAny (), port);

}

PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory",

hubLocalAddress);

ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());

hubApp.Start (Seconds (1.0));

hubApp.Stop (Seconds (10.0));

//

// Create OnOff applications to send TCP to the hub, one on each spoke node.

//

OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());

onOffHelper.SetAttribute ("OnTime", StringValue

("ns3::ConstantRandomVariable[Constant=1]"));

onOffHelper.SetAttribute ("OffTime", StringValue

("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer spokeApps;

for (uint32\_t i = 0; i < star.SpokeCount (); ++i)

{

AddressValue remoteAddress;

if (useIpv6 == 0)

{

remoteAddress = AddressValue(InetSocketAddress

(star.GetHubIpv4Address (i), port));

}

else

{

remoteAddress = AddressValue(Inet6SocketAddress

(star.GetHubIpv6Address (i), port));

}

onOffHelper.SetAttribute ("Remote", remoteAddress);

spokeApps.Add (onOffHelper.Install (star.GetSpokeNode (i)));

}

spokeApps.Start (Seconds (1.0));

spokeApps.Stop (Seconds (10.0));

NS\_LOG\_INFO ("Enable static global routing.");

//

// Turn on global static routing so we can actually be routed across the star.

//

if (useIpv6 == 0)

{

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

}

// Set the bounding box for animation

star.BoundingBox (1, 1, 100, 100);

// Create the animation object and configure for specified output

AnimationInterface anim ("straT.xml");

NS\_LOG\_INFO ("Run Simulation.");

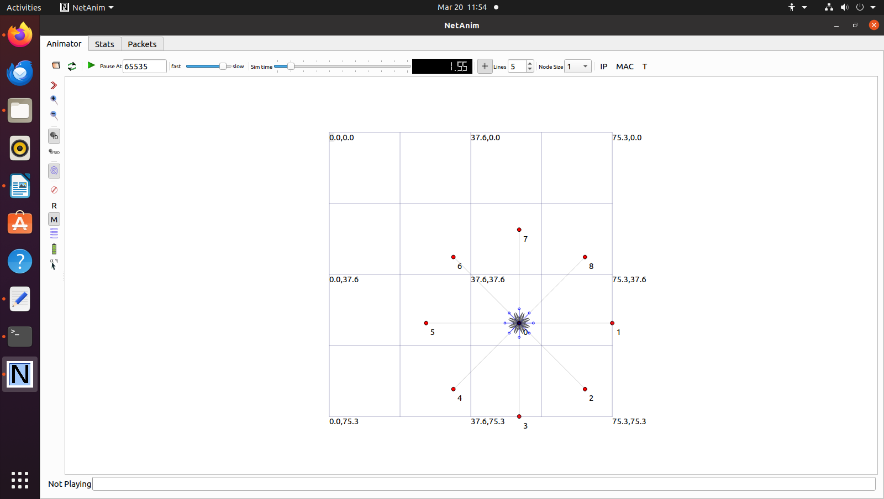
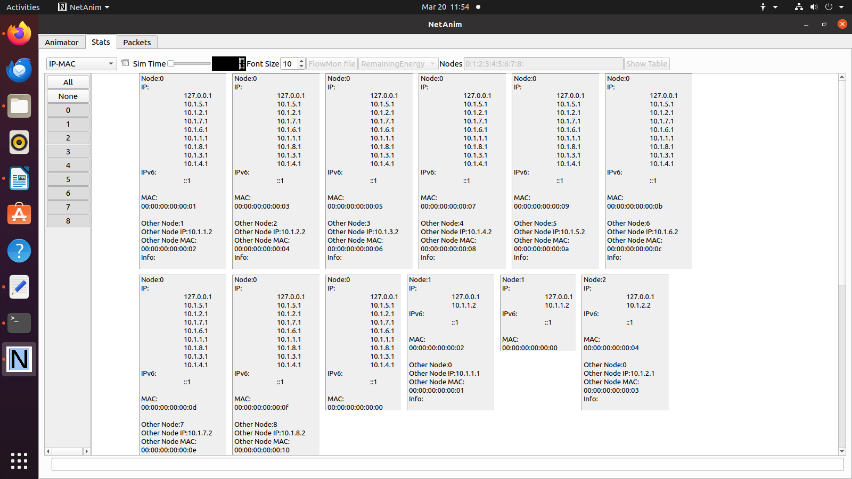
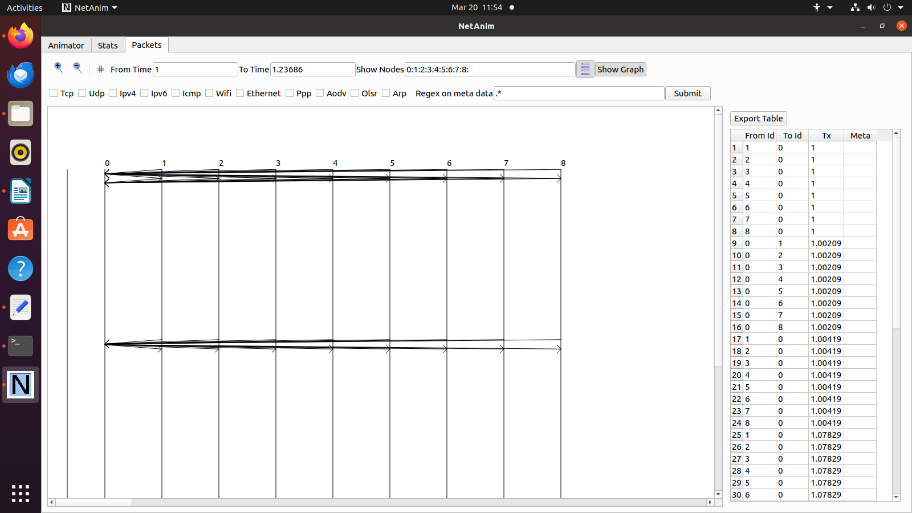
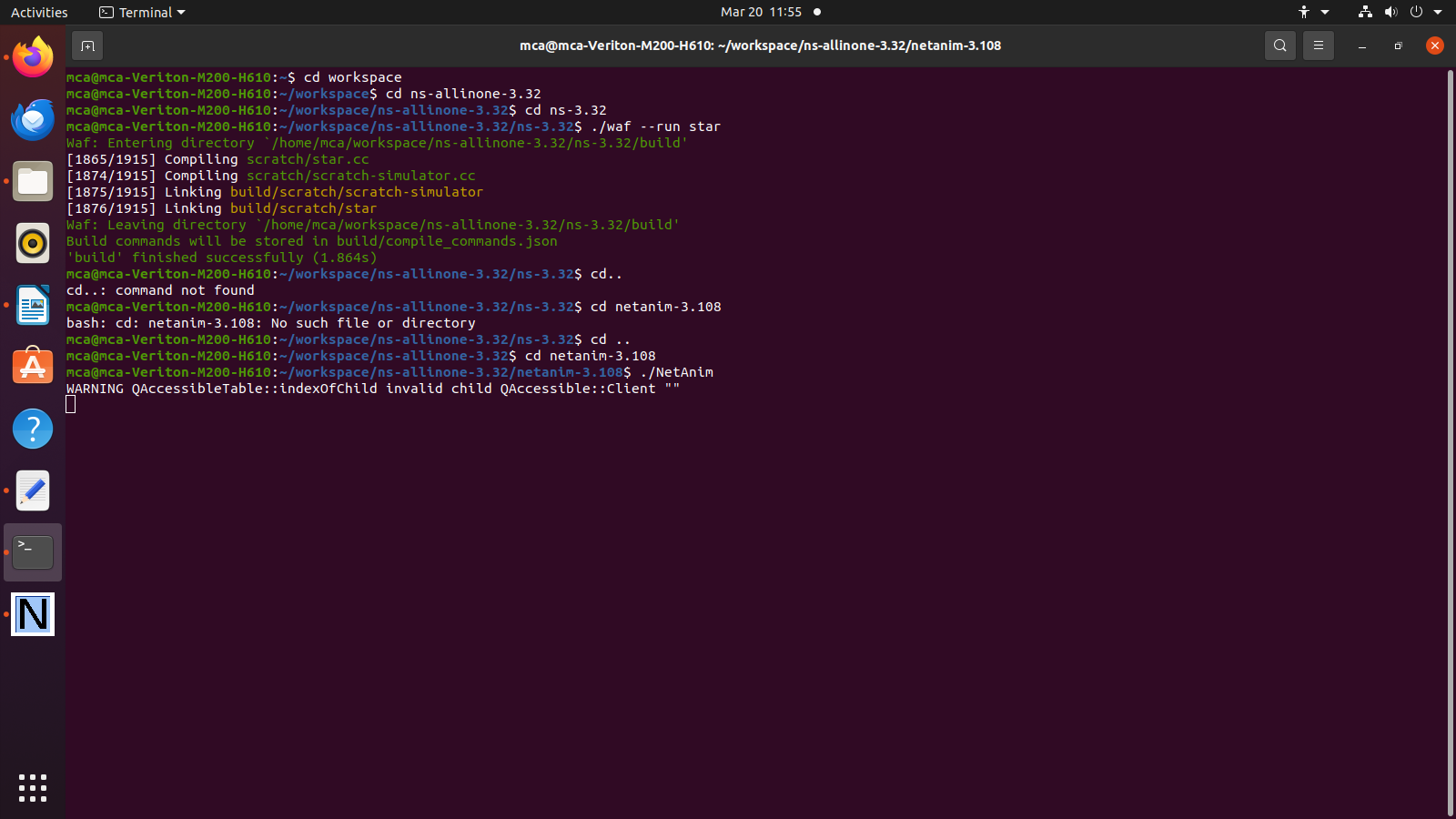
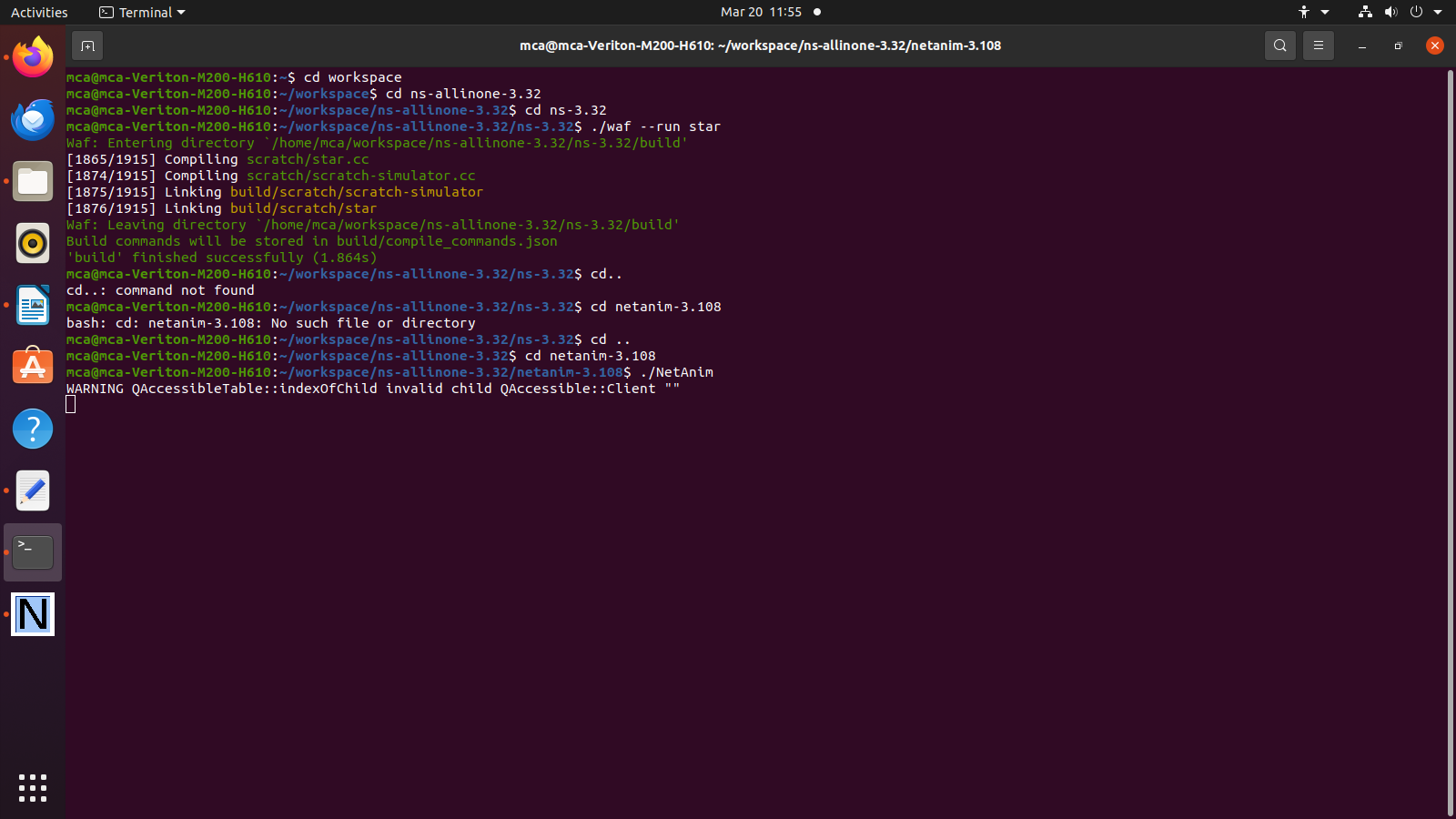
Simulator::Run ();

Simulator::Destroy ();

NS\_LOG\_INFO ("Done.");

return 0;

}

**5. Output: -**

**6. Conclusion: -**

Successfully implemented Star topology using NS3.