Name : Hit vaghela
Roll No. : 22BCE368

Name : Kalpesh vala
Roll No. : 22BCE375

Subject : Data communication

Topic : Routing algorithm implementations

1. Abstract

Computer networks require routing algorithms to determine how data packets are delivered from source to destination. The Distance Vector Algorithm and the Link State Algorithm are two well-known algorithms in this field. Based on routing information that is transmitted iteratively, the Distance Vector Algorithm—which is demonstrated by protocols like RIP and EIGRP—determines the shortest path to each destination node. On the other hand, the Link State Algorithm, which is symbolized by Dijkstra's Algorithm, creates a topology map of the entire network and uses centralized data to determine the shortest paths. This study presents a succinct analysis of these algorithms, emphasizing their fundamental ideas, traits, and uses in network routing.

1.1 Keywords

Distance vector algorithm , Link State Algorithm , NS-3 Network simulator , RIP Protocol , Dijkstra algorithm, OLSR Protocol , Routing table

2. Introduction

1) Distance Vector Algorithm:

- The Distance Vector Algorithm, also known as the Bellman-Ford Algorithm, is one of the earliest and simplest routing algorithms used in computer networks. It operates on the principle of iteratively exchanging routing information between neighboring routers to determine the shortest path to each destination node. Each router maintains a routing table containing information about the distance (cost) to reach each destination and the next-hop router along the path.
- One of the defining features of the Distance Vector Algorithm is its distributed nature, wherein each router makes routing decisions based solely on the information received from its neighbors. Routers exchange routing updates periodically or in response to changes in the network topology. However, this decentralized approach can lead to slow convergence and routing loops, especially in large networks.
- Despite its simplicity, the Distance Vector Algorithm has found practical applications in various network protocols, including RIP (Routing Information Protocol) and EIGRP (Enhanced Interior Gateway Routing Protocol). Its ease of implementation and low computational overhead make it suitable for small to medium-sized networks with relatively stable topologies.

2) Link State Algorithm

- In contrast to the Distance Vector Algorithm, the Link State Algorithm, also known as Dijkstra's Algorithm, adopts a centralized approach to routing. It operates by building a complete topological map of the network, known as the Link State Database (LSDB), through the exchange of Link State Advertisements (LSAs) among all routers in the network. Each router then computes the shortest path to every destination node using Dijkstra's shortest path algorithm based on the information in the LSDB.
- The Link State Algorithm offers several advantages over the Distance Vector Algorithm, including faster convergence, loop-free routing, and scalability. By maintaining a global view of the network topology, routers can make more informed routing decisions and adapt quickly to changes in the network. However, the Link State Algorithm requires more memory and computational resources to maintain and process the LSDB, making it less suitable for resource-constrained environments.
- The Link State Algorithm serves as the foundation for many modern routing protocols, such as OSPF (Open Shortest Path First) and IS-IS (Intermediate System to Intermediate System), used in large-scale enterprise and service provider networks. Its robustness and efficiency make it well-suited for dynamic and complex network environments

3. Terminology

Routing : Routing is the process of determining the path that data packets must take through a network from source to destination. It involves choosing the best route based on several factors such as network topology, link costs, traffic conditions and road policies. The purpose of routing is to ensure efficient and reliable data transmission by minimizing delays, congestion and packet loss.

Routing protocols : Routing protocols are simply algorithms that routers use to communicate with each other, sharing information about the best route to a destination.

Routing Table: A data structure stored in a router that maps destination addresses to outgoing interfaces or followed routers.

Distance Vector: A data structure stored in a router that maps destination addresses to outgoing interfaces or followed routers.

Link State: A routing algorithm in which routers exchange link state information with all other routers on the network.

Routing Information Protocol (RIP): A distance vector routing protocol used to exchange routing information between routers in small and medium-sized networks.

Open Shortest Path First (OSPF): A link-state routing protocol commonly used in large-scale networks to calculate the shortest path between routers based on link costs.

4. Implementation

Algorithm used : Distance vector , link - state routing

Platform or Tools : NS-3 (Network simulator) , Wireshark , Tracemetrics

Coding part :

1) **Distance vector algorithm :**

```
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/flow-monitor-module.h" // Include the FlowMonitorHelper
module
#include "ns3/internet-apps-module.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-routing-table-entry.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/netanim-module.h" // Include the XmlHelper module

#include <fstream>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("RipSimpleRouting");

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

int
main(int argc, char** argv)
{
   // double xPosition = 0.0; // Set the x-coordinate for the nodes
   // double yPosition = 0.0; // Set the y-coordinate for the nodes
   bool verbose = false;
   bool printRoutingTables = false;
   bool showPings = false;
   std::string SplitHorizon("PoisonReverse");

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

   if (verbose)
   {
       LogComponentEnableAll(LogLevel(LOG_PREFIX_TIME | LOG_PREFIX_NODE));
       LogComponentEnable("RipSimpleRouting", LOG_LEVEL_INFO);
       LogComponentEnable("Rip", LOG_LEVEL_ALL);
       LogComponentEnable("Ipv4Interface", LOG_LEVEL_ALL);
       LogComponentEnable("Icmpv4L4Protocol", LOG_LEVEL_ALL);
       LogComponentEnable("Ipv4L3Protocol", LOG_LEVEL_ALL);
       LogComponentEnable("ArpCache", LOG_LEVEL_ALL);
       LogComponentEnable("Ping", LOG_LEVEL_ALL);
   }

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

```
    else
    {
        Config::SetDefault("ns3::Rip::SplitHorizon",
EnumValue(RipNg::POISON_REVERSE));
    }

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

    NS_LOG_INFO("Create channels.");
    CsmaHelper csma;
    csma.SetChannelAttribute("DataRate", DataRateValue(5000000));
    csma.SetChannelAttribute("Delay", TimeValue(MilliSeconds(2)));
    NetDeviceContainer ndc1 = csma.Install(net1);
    NetDeviceContainer ndc2 = csma.Install(net2);
    NetDeviceContainer ndc3 = csma.Install(net3);
    NetDeviceContainer ndc4 = csma.Install(net4);
    NetDeviceContainer ndc5 = csma.Install(net5);
    NetDeviceContainer ndc6 = csma.Install(net6);
    NetDeviceContainer ndc7 = csma.Install(net7);
```

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

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

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

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

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

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

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

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

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

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

   ipv4.SetBase(Ipv4Address("10.0.3.0"), Ipv4Mask("255.255.255.0"));
   Ipv4InterfaceContainer iic4 = ipv4.Assign(ndc4);

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

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

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

   Ptr<Ipv4StaticRouting> staticRouting;
   staticRouting = Ipv4RoutingHelper::GetRouting<Ipv4StaticRouting>(
       src->GetObject<Ipv4>()->GetRoutingProtocol());
   staticRouting->SetDefaultRoute("10.0.0.2", 1); // source will send the
data to the next hop with
                                                   // the IP address
"10.0.0.2" via its interface 1.
   staticRouting = Ipv4RoutingHelper::GetRouting<Ipv4StaticRouting>(
       dst->GetObject<Ipv4>()->GetRoutingProtocol());
   staticRouting->SetDefaultRoute("10.0.6.1", 1);

   if (!printRoutingTables)
   {
       Ptr<OutputStreamWrapper> routingStream =
Create<OutputStreamWrapper>(&std::cout);

       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(30.0), a,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(30.0), b,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(30.0), c,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(30.0), d,
routingStream);
```

```
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(60.0), a,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(60.0), b,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(60.0), c,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(60.0), d,
routingStream);

       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(90.0), a,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(90.0), b,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(90.0), c,
routingStream);
       Ipv4RoutingHelper::PrintRoutingTableAt(Seconds(90.0), d,
routingStream);
   }

   NS_LOG_INFO("Create Applications.");
   uint32_t packetSize = 1024;
   Time interPacketInterval = Seconds(1.0);
   PingHelper ping(Ipv4Address("10.0.6.2"));

   ping.SetAttribute("Interval", TimeValue(interPacketInterval));
   ping.SetAttribute("Size", UintegerValue(packetSize));
   if (showPings)
   {
       ping.SetAttribute("VerboseMode",
EnumValue(Ping::VerboseMode::VERBOSE));
   }
   ApplicationContainer apps = ping.Install(src);
   AnimationInterface anim("rip-simple_animation.xml");
   anim.SetConstantPosition(src, 1.0, 12.0);
   anim.SetConstantPosition(dst, 1.0, 35.0);
   anim.SetConstantPosition(a, 23.0, 12.0);
   anim.SetConstantPosition(b, 23.0, 35.0);
   anim.SetConstantPosition(c, 55.0, 12.0);
   anim.SetConstantPosition(d, 55.0, 35.0);
   apps.Start(Seconds(1.0));
   apps.Stop(Seconds(110.0));

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

   FlowMonitorHelper flowmon;                              // Create a
FlowMonitorHelper object
   Ptr<FlowMonitor> monitor = flowmon.InstallAll(); // Install flow
monitor on all nodes

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



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

   monitor->SerializeToXmlFile("rip-simple-routing-flow.xml",
                               true,
                               true); // Serialize flow monitor data to
XML file
   Simulator::Destroy();
   NS_LOG_INFO("Done.");
   // AnimationInterface anim("new_animation.xml");

   // Set constant positions for n


   // The rest of your code...
   return 0;
}
```

Code Overview:

1. **Header Includes: The code includes various NS-3 module headers required for network simulation, including core, CSMA, flow monitor, Internet, IPv4 routing, and NetAnim.**

2. **Namespace: The code uses the ns3 namespace to access NS-3 classes and functions.**

3. **Log Component Definition: The NS_LOG_COMPONENT_DEFINE macro defines the logging component for this program.**

4. **Function Definitions:**
   a. **TearDownLink: This function is defined to tear down a link between two nodes by setting their corresponding IPv4 interfaces down.**
   b. **main: The main function where the simulation is configured and executed.**

5. **Command Line Parsing: The code uses the CommandLine class to parse command-line arguments for controlling various simulation parameters such as verbosity, routing table printing, ping display, and split horizon strategy.**

6. **Router Configuration: Nodes representing routers (RouterA, RouterB, RouterC, RouterD) and source-destination nodes (SrcNode, DstNode) are created and organized into node containers. Channels between nodes are created using the CSMA helper.**

7. **IPv4 and Routing Configuration: RIP routing protocol is configured using the RipHelper. Interfaces are excluded, and interface metrics are set as per the split horizon strategy specified in the command-line arguments. The IPv4 stack and routing helper are installed on the routers.**

8. **Address Assignment: IPv4 addresses are assigned to the nodes using the Ipv4AddressHelper, and default routes are set for the source and destination nodes.**

9. **Application Setup: A Ping application is configured to send ICMP echo requests to the destination node's IPv4 address.**

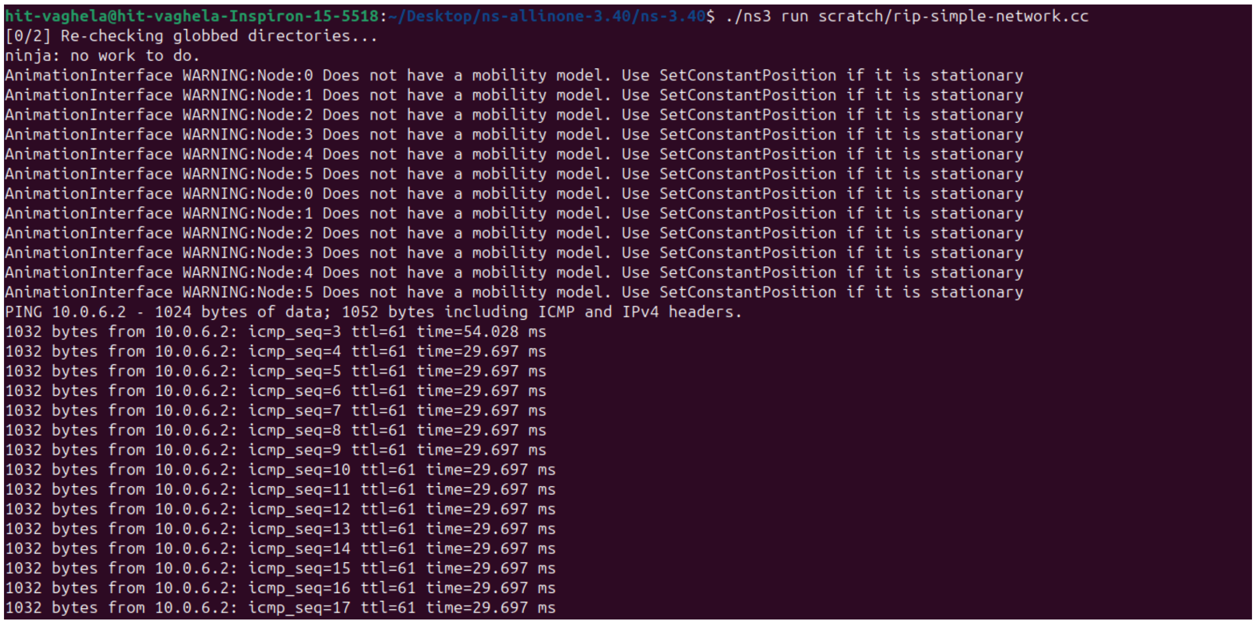
10. **Animation Interface Setup: The NetAnim animation interface is configured to visualize the network topology and node movements during the simulation.**

11. **Tracing and Monitoring: ASCII tracing and PCAP tracing are enabled to capture network events and packet traces. Flow monitoring is set up to monitor flows in the network.**
12. **Simulation Execution: The simulation is scheduled to run for a specified duration. During the simulation, a link between RouterB and RouterD is torn down after 40 seconds.**

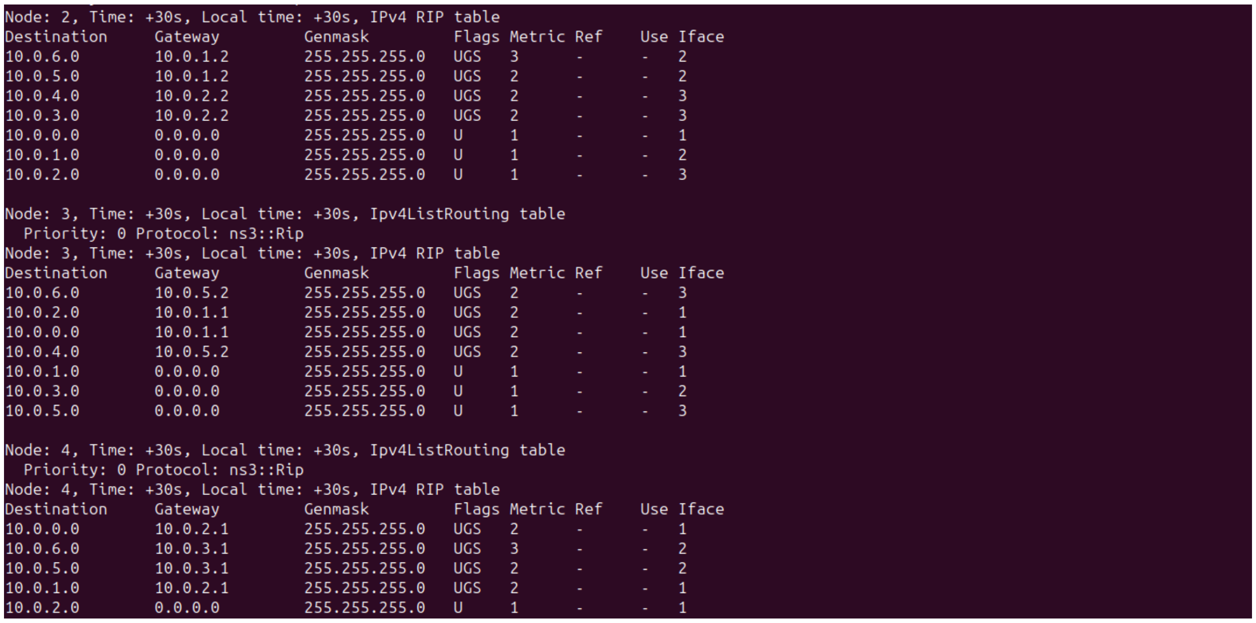
13. **Result Serialization: Flow monitor data is serialized to an XML file for further analysis.**

14. **Simulation Termination: The simulation is stopped, and resources are cleaned up before exiting.**

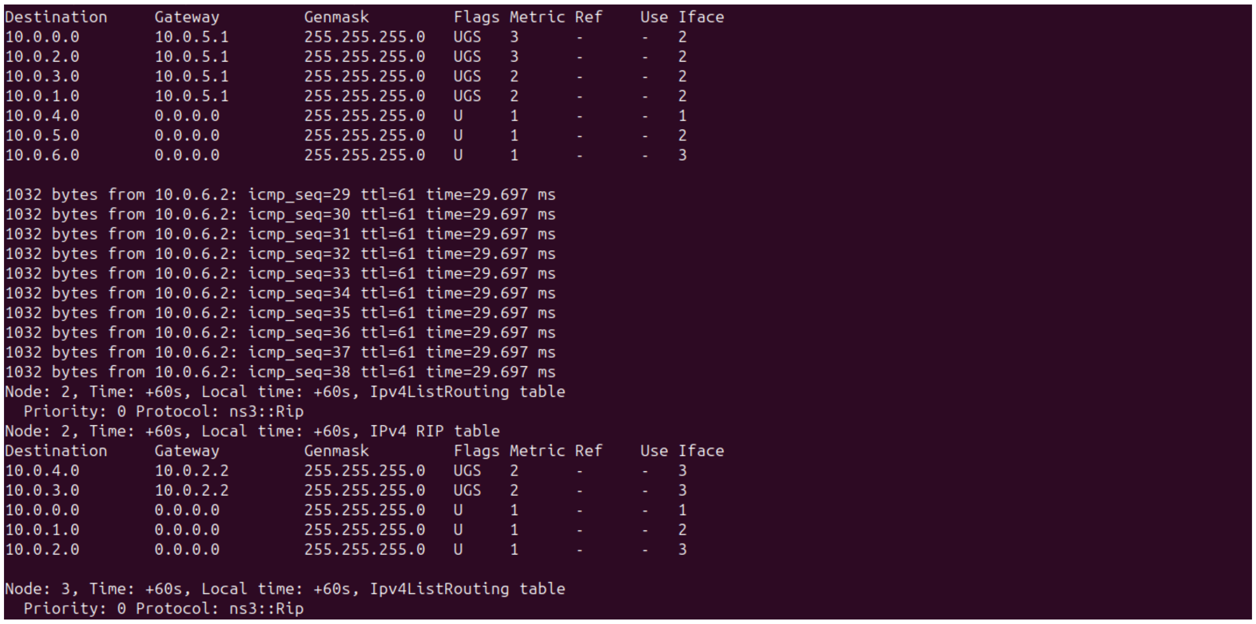
Output :



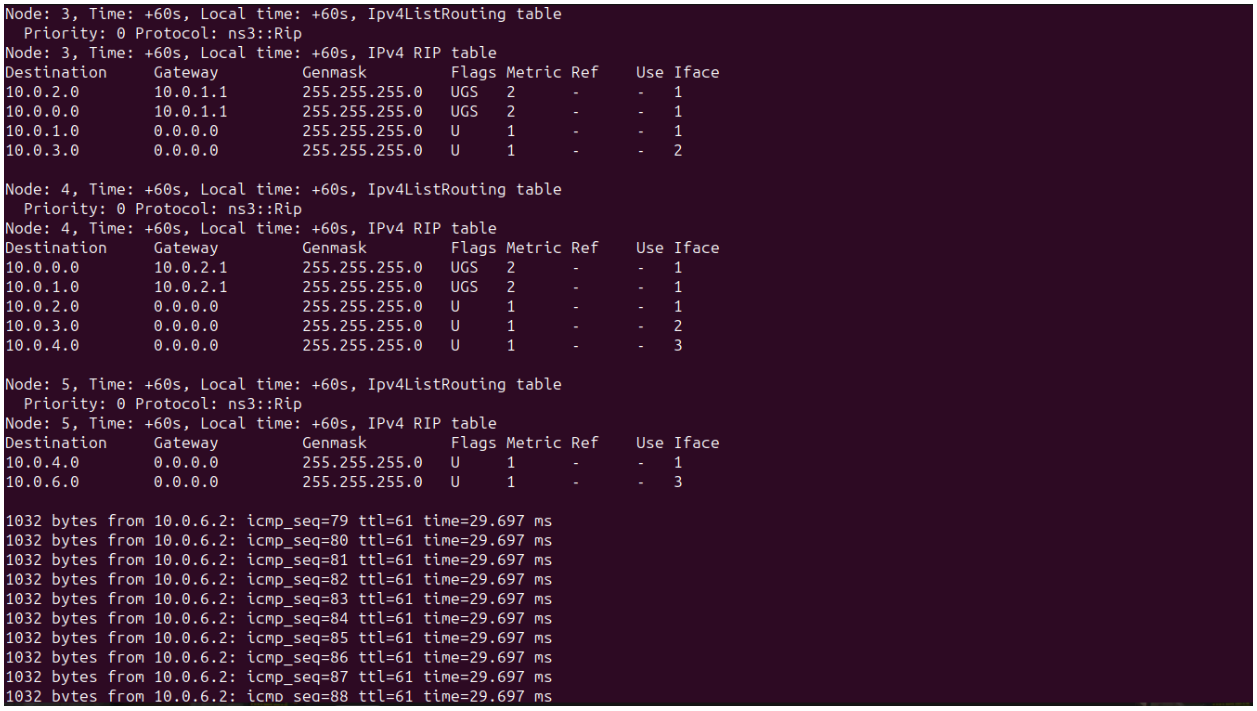
```
hit-vaghela@hit-vaghela-Inspiron-15-5518:~/Desktop/ns-allinone-3.40/ns-3.40$ ./ns3 run scratch/rip-simple-network.cc
[0/2] Re-checking globbed directories...
ninja: no work to do.
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:2 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:3 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:4 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:5 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:2 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:3 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:4 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:5 Does not have a mobility model. Use SetConstantPosition if it is stationary
PING 10.0.6.2 - 1024 bytes of data; 1052 bytes including ICMP and IPv4 headers.
1032 bytes from 10.0.6.2: icmp_seq=3 ttl=61 time=54.028 ms
1032 bytes from 10.0.6.2: icmp_seq=4 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=5 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=6 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=7 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=8 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=9 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=10 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=11 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=12 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=13 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=14 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=15 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=16 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=17 ttl=61 time=29.697 ms
```



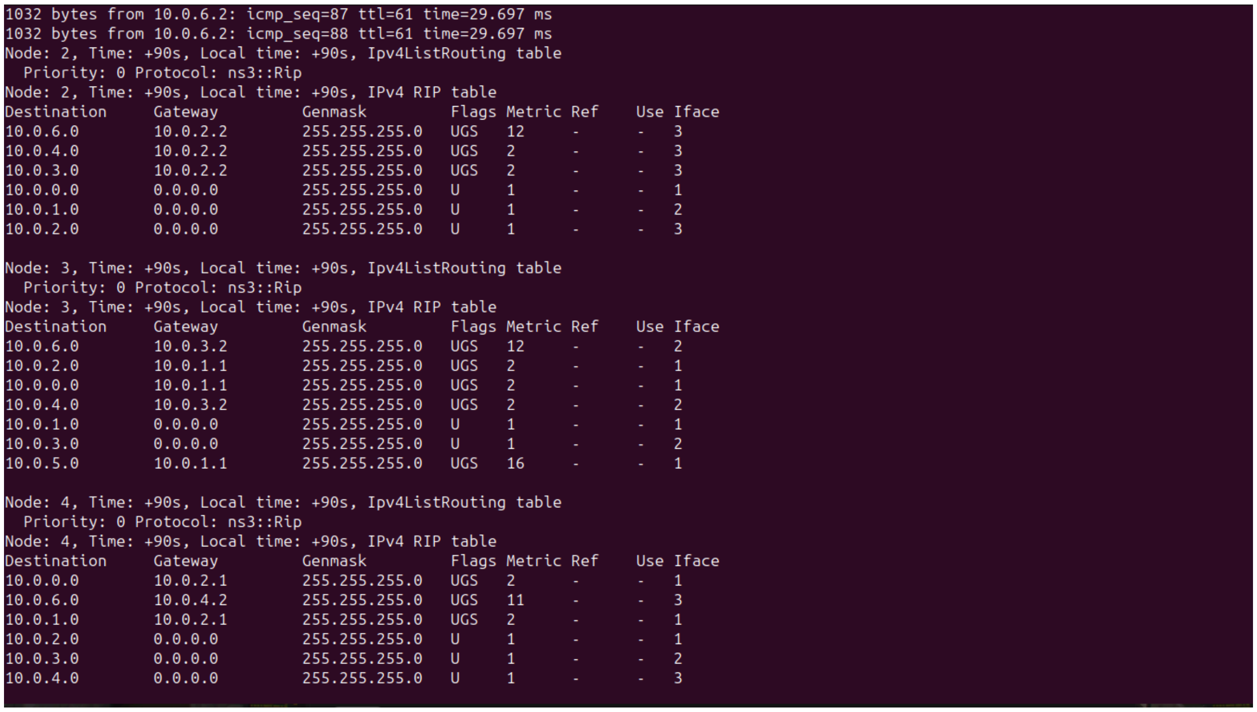
```
Node: 2, Time: +30s, Local time: +30s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.6.0        10.0.1.2        255.255.255.0   UGS   3      -      -   2
10.0.5.0        10.0.1.2        255.255.255.0   UGS   2      -      -   2
10.0.4.0        10.0.2.2        255.255.255.0   UGS   2      -      -   3
10.0.3.0        10.0.2.2        255.255.255.0   UGS   2      -      -   3
10.0.0.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.1.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.2.0        0.0.0.0         255.255.255.0   U     1      -      -   3

Node: 3, Time: +30s, Local time: +30s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 3, Time: +30s, Local time: +30s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.6.0        10.0.5.2        255.255.255.0   UGS   2      -      -   3
10.0.2.0        10.0.1.1        255.255.255.0   UGS   2      -      -   1
10.0.0.0        10.0.1.1        255.255.255.0   UGS   2      -      -   1
10.0.4.0        10.0.5.2        255.255.255.0   UGS   2      -      -   3
10.0.1.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.3.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.5.0        0.0.0.0         255.255.255.0   U     1      -      -   3

Node: 4, Time: +30s, Local time: +30s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 4, Time: +30s, Local time: +30s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.2.1        255.255.255.0   UGS   2      -      -   1
10.0.6.0        10.0.3.1        255.255.255.0   UGS   3      -      -   2
10.0.5.0        10.0.3.1        255.255.255.0   UGS   2      -      -   2
10.0.1.0        10.0.2.1        255.255.255.0   UGS   2      -      -   1
10.0.2.0        0.0.0.0         255.255.255.0   U     1      -      -   1
```



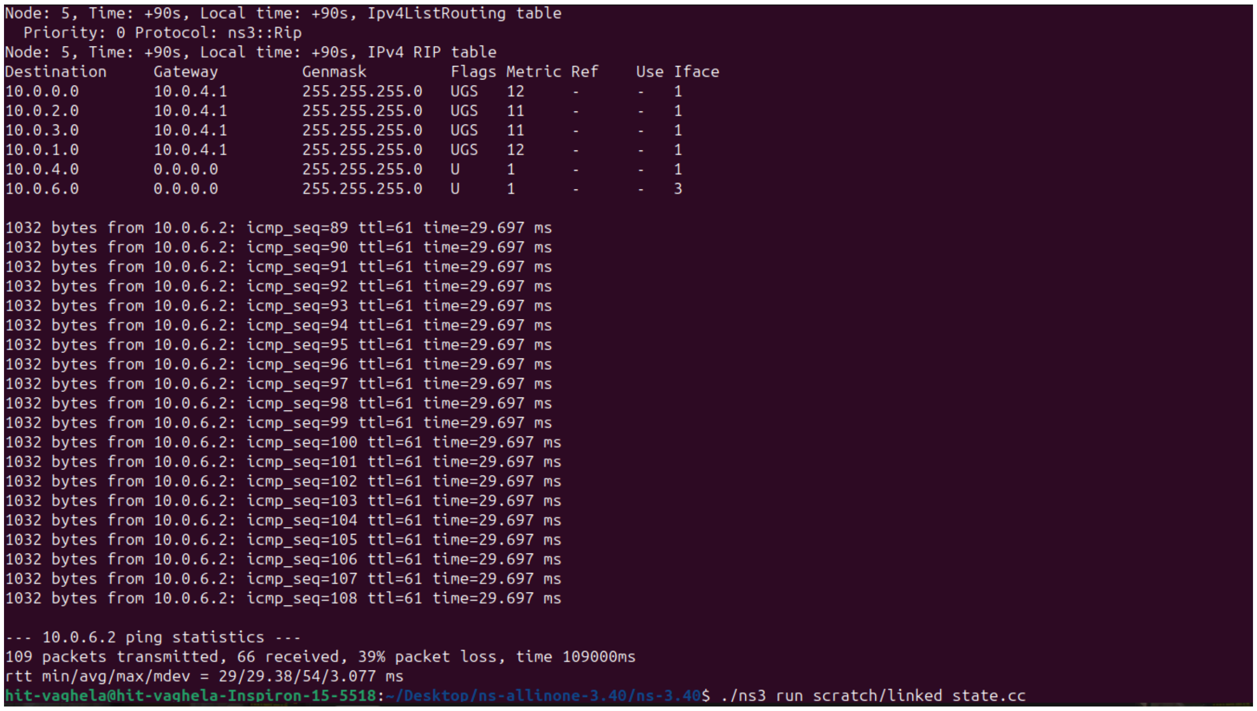
```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.5.1        255.255.255.0   UGS   3      -      -   2
10.0.2.0        10.0.5.1        255.255.255.0   UGS   3      -      -   2
10.0.3.0        10.0.5.1        255.255.255.0   UGS   2      -      -   2
10.0.1.0        10.0.5.1        255.255.255.0   UGS   2      -      -   2
10.0.4.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.5.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.6.0        0.0.0.0         255.255.255.0   U     1      -      -   3

1032 bytes from 10.0.6.2: icmp_seq=29 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=30 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=31 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=32 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=33 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=34 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=35 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=36 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=37 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=38 ttl=61 time=29.697 ms
Node: 2, Time: +60s, Local time: +60s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 2, Time: +60s, Local time: +60s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.4.0        10.0.2.2        255.255.255.0   UGS   2      -      -   3
10.0.3.0        10.0.2.2        255.255.255.0   UGS   2      -      -   3
10.0.0.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.1.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.2.0        0.0.0.0         255.255.255.0   U     1      -      -   3

Node: 3, Time: +60s, Local time: +60s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
```



```
Node: 3, Time: +60s, Local time: +60s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 3, Time: +60s, Local time: +60s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.2.0        10.0.1.1        255.255.255.0   UGS   2      -      -   1
10.0.0.0        10.0.1.1        255.255.255.0   UGS   2      -      -   1
10.0.1.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.3.0        0.0.0.0         255.255.255.0   U     1      -      -   2

Node: 4, Time: +60s, Local time: +60s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 4, Time: +60s, Local time: +60s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.2.1        255.255.255.0   UGS   2      -      -   1
10.0.1.0        10.0.2.1        255.255.255.0   UGS   2      -      -   1
10.0.2.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.3.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.4.0        0.0.0.0         255.255.255.0   U     1      -      -   3

Node: 5, Time: +60s, Local time: +60s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 5, Time: +60s, Local time: +60s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.4.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.6.0        0.0.0.0         255.255.255.0   U     1      -      -   3

1032 bytes from 10.0.6.2: icmp_seq=79 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=80 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=81 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=82 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=83 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=84 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=85 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=86 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=87 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=88 ttl=61 time=29.697 ms
```



```
1032 bytes from 10.0.6.2: icmp_seq=87 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=88 ttl=61 time=29.697 ms
Node: 2, Time: +90s, Local time: +90s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 2, Time: +90s, Local time: +90s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.6.0        10.0.2.2        255.255.255.0   UGS   12     -      -   3
10.0.4.0        10.0.2.2        255.255.255.0   UGS   2      -      -   3
10.0.3.0        10.0.2.2        255.255.255.0   UGS   2      -      -   3
10.0.0.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.1.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.2.0        0.0.0.0         255.255.255.0   U     1      -      -   3

Node: 3, Time: +90s, Local time: +90s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 3, Time: +90s, Local time: +90s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.6.0        10.0.3.2        255.255.255.0   UGS   12     -      -   2
10.0.2.0        10.0.1.1        255.255.255.0   UGS   2      -      -   1
10.0.0.0        10.0.1.1        255.255.255.0   UGS   2      -      -   1
10.0.4.0        10.0.3.2        255.255.255.0   UGS   2      -      -   2
10.0.1.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.3.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.5.0        10.0.1.1        255.255.255.0   UGS   16     -      -   1

Node: 4, Time: +90s, Local time: +90s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 4, Time: +90s, Local time: +90s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.2.1        255.255.255.0   UGS   2      -      -   1
10.0.6.0        10.0.4.2        255.255.255.0   UGS   11     -      -   3
10.0.1.0        10.0.2.1        255.255.255.0   UGS   2      -      -   1
10.0.2.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.3.0        0.0.0.0         255.255.255.0   U     1      -      -   2
10.0.4.0        0.0.0.0         255.255.255.0   U     1      -      -   3
```



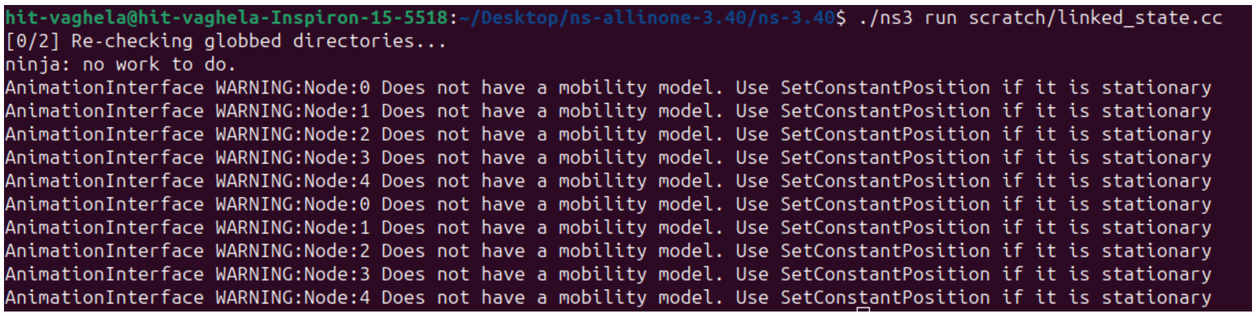
```
Node: 5, Time: +90s, Local time: +90s, Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Rip
Node: 5, Time: +90s, Local time: +90s, IPv4 RIP table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0        10.0.4.1        255.255.255.0   UGS   12     -      -   1
10.0.2.0        10.0.4.1        255.255.255.0   UGS   11     -      -   1
10.0.3.0        10.0.4.1        255.255.255.0   UGS   11     -      -   1
10.0.1.0        10.0.4.1        255.255.255.0   UGS   12     -      -   1
10.0.4.0        0.0.0.0         255.255.255.0   U     1      -      -   1
10.0.6.0        0.0.0.0         255.255.255.0   U     1      -      -   3

1032 bytes from 10.0.6.2: icmp_seq=89 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=90 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=91 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=92 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=93 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=94 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=95 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=96 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=97 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=98 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=99 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=100 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=101 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=102 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=103 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=104 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=105 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=106 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=107 ttl=61 time=29.697 ms
1032 bytes from 10.0.6.2: icmp_seq=108 ttl=61 time=29.697 ms

--- 10.0.6.2 ping statistics ---
109 packets transmitted, 66 received, 39% packet loss, time 109000ms
rtt min/avg/max/mdev = 29/29.38/54/3.077 ms
hit-vaghela@hit-vaghela-Inspiron-15-5518:~/Desktop/ns-allinone-3.40/ns-3.40$ ./ns3 run scratch/linked_state.cc
```

Link-state routing :

```cpp
//Link state routing for Wired Networks, OLSR
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>

#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/olsr-helper.h"
#include "ns3/ipv4-static-routing-helper.h"
#include "ns3/ipv4-list-routing-helper.h"
#include "ns3/netanim-module.h"

using namespace ns3;
```

```
NS_LOG_COMPONENT_DEFINE ("SimplePointToPointOlsrExample");

int
main (int argc, char *argv[])
{
 Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue
(210));
 Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue
("448kb/s"));

 CommandLine cmd;
 cmd.Parse (argc, argv);

 NS_LOG_INFO ("Create nodes.");
 NodeContainer c;
 c.Create (5);
 NodeContainer n01 = NodeContainer (c.Get (0), c.Get (1));
 NodeContainer n14 = NodeContainer (c.Get (1), c.Get (4));
 NodeContainer n43 = NodeContainer (c.Get (4), c.Get (3));
 NodeContainer n30 = NodeContainer (c.Get (3), c.Get (0));
 NodeContainer n02 = NodeContainer (c.Get (0), c.Get (2));
 NodeContainer n23 = NodeContainer (c.Get (2), c.Get (3));

 // Enable OLSR
 NS_LOG_INFO ("Enabling OLSR Routing.");
 OlsrHelper olsr;

 Ipv4StaticRoutingHelper staticRouting;

 Ipv4ListRoutingHelper list;
 list.Add (staticRouting, 0);
 list.Add (olsr, 10);

 InternetStackHelper internet;
 internet.SetRoutingHelper (list); // has effect on the next Install ()
 internet.Install (c);

 // We create the channels first without any IP addressing information
 NS_LOG_INFO ("Create channels.");
 PointToPointHelper p2p;
 p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer nd01 = p2p.Install (n01);
 p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("10ms"));
NetDeviceContainer nd14 = p2p.Install (n14);
 p2p.SetDeviceAttribute ("DataRate", StringValue ("50Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("50ms"));
NetDeviceContainer nd43 = p2p.Install (n43);
 p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("5ms"));
NetDeviceContainer nd30 = p2p.Install (n30);
 p2p.SetDeviceAttribute ("DataRate", StringValue ("1Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));
NetDeviceContainer nd02 = p2p.Install (n02);
 p2p.SetDeviceAttribute ("DataRate", StringValue ("2Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer nd23 = p2p.Install (n23);

// Later, we add IP addresses.
NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i01 = ipv4.Assign (nd01);

ipv4.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer i14 = ipv4.Assign (nd14);

ipv4.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer i43 = ipv4.Assign (nd43);

ipv4.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer i30 = ipv4.Assign (nd30);

ipv4.SetBase ("10.1.5.0", "255.255.255.0");
Ipv4InterfaceContainer i02 = ipv4.Assign (nd02);

ipv4.SetBase ("10.1.6.0", "255.255.255.0");
Ipv4InterfaceContainer i23 = ipv4.Assign (nd23);
 // Create the OnOff application to send UDP datagrams of size
 // 210 bytes at a rate of 448 Kb/s
  NS_LOG_INFO ("Create Applications.");
 uint16_t port = 8000;   // Discard port (RFC 863)

 OnOffHelper onoff1 ("ns3::UdpSocketFactory",
                     InetSocketAddress (i02.GetAddress (1), port));
 onoff1.SetConstantRate (DataRate ("448kb/s"));

 ApplicationContainer onOffApp1 = onoff1.Install (c.Get (1));
 onOffApp1.Start (Seconds (10.0));
 onOffApp1.Stop (Seconds (20.0));

 // Create packet sinks to receive these packets
 PacketSinkHelper sink ("ns3::UdpSocketFactory",
                         InetSocketAddress (Ipv4Address::GetAny (), port));
 NodeContainer sinks = NodeContainer (c.Get (2), c.Get (1));
 ApplicationContainer sinkApps = sink.Install (sinks);
 sinkApps.Start (Seconds (0.0));
 sinkApps.Stop (Seconds (21.0));

 AsciiTraceHelper ascii;
 p2p.EnableAsciiAll (ascii.CreateFileStream ("lsr.tr"));
 p2p.EnablePcapAll ("lsr");

AnimationInterface anim("lsr.xml");
anim.SetConstantPosition(c.Get(0),0.0,50.0);
anim.SetConstantPosition(c.Get(1),50.0,100.0);
anim.SetConstantPosition(c.Get(2),50.0,50.0);
anim.SetConstantPosition(c.Get(3),50.0,0.0);
anim.SetConstantPosition(c.Get(4),100.0,50.0);

 Simulator::Stop (Seconds (30));
 NS_LOG_INFO ("Run Simulation.");
 Simulator::Run ();
 Simulator::Destroy ();
 NS_LOG_INFO ("Done.");

 return 0;
}
```

Output :



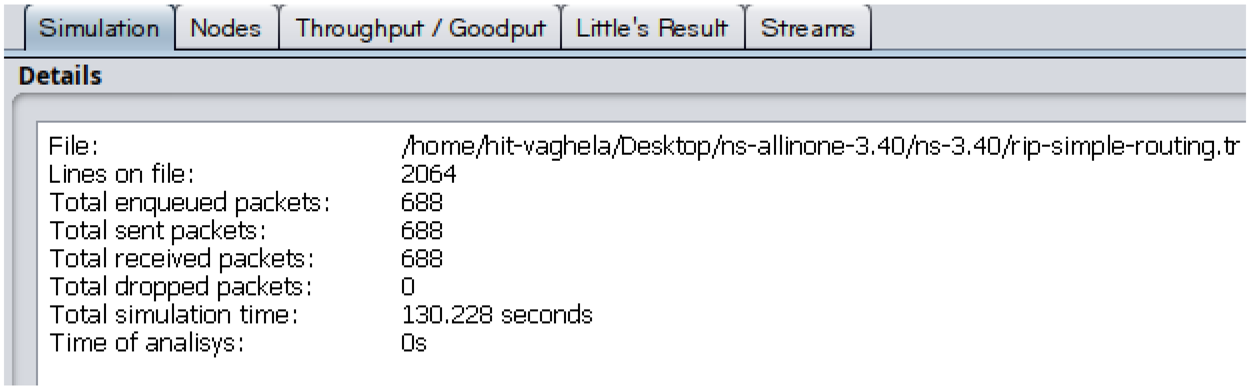
```
hit-vaghela@hit-vaghela-Inspiron-15-5518:~/Desktop/ns-allinone-3.40/ns-3.40$ ./ns3 run scratch/linked_state.cc
[0/2] Re-checking globbed directories...
ninja: no work to do.
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:2 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:3 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:4 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:2 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:3 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:4 Does not have a mobility model. Use SetConstantPosition if it is stationary
```

Code Overview :

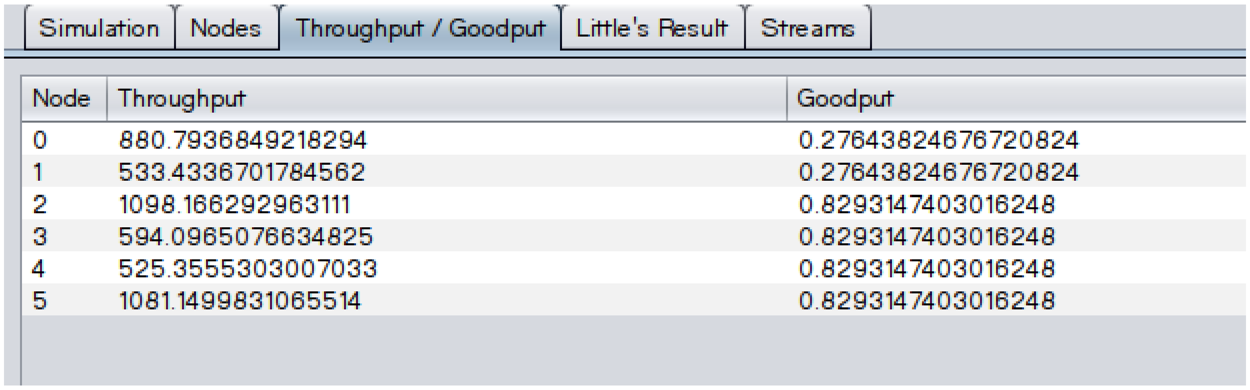
1. **Header Includes: The code includes necessary header files from NS-3 and standard C++ libraries.**
2. **Namespace: The ns3 namespace is used throughout the code to access NS-3 classes and functions.**
3. **Logging Component Definition: The NS_LOG_COMPONENT_DEFINE macro defines the logging component for this program.**
4. **Main Function: The main function is the entry point of the program where the simulation is configured and executed.**
5. **Configuration: Default settings for packet size and data rate are set using the Config::SetDefault method. Command-line arguments are parsed using CommandLine.**
6. **Node and Channel Creation: Nodes are created and organized into node containers. Point-to-point channels are established between pairs of nodes using the PointToPointHelper.**
7. **OLSR Configuration: The OLSR routing protocol is enabled and configured using the OlsrHelper. IPv4 static routing is also set up.**
8. **IP Address Assignment: IP addresses are assigned to the nodes/interfaces using the Ipv4AddressHelper.**
9. **Application Setup: OnOff applications are created to generate UDP traffic between specific nodes at a constant rate.**
10. **Packet Sink Creation: Packet sinks are established to receive packets generated by the OnOff applications.**
11. **Tracing and Monitoring: ASCII tracing and PCAP tracing are enabled to capture network events and packet traces. NetAnim animation interface is set up to visualize the network topology and node movements.**
12. **Simulation Execution: The simulation is scheduled to run for a specified duration using Simulator::Run.**
13. **Result Output: After the simulation completes, the program outputs a message indicating its completion.**

Result :

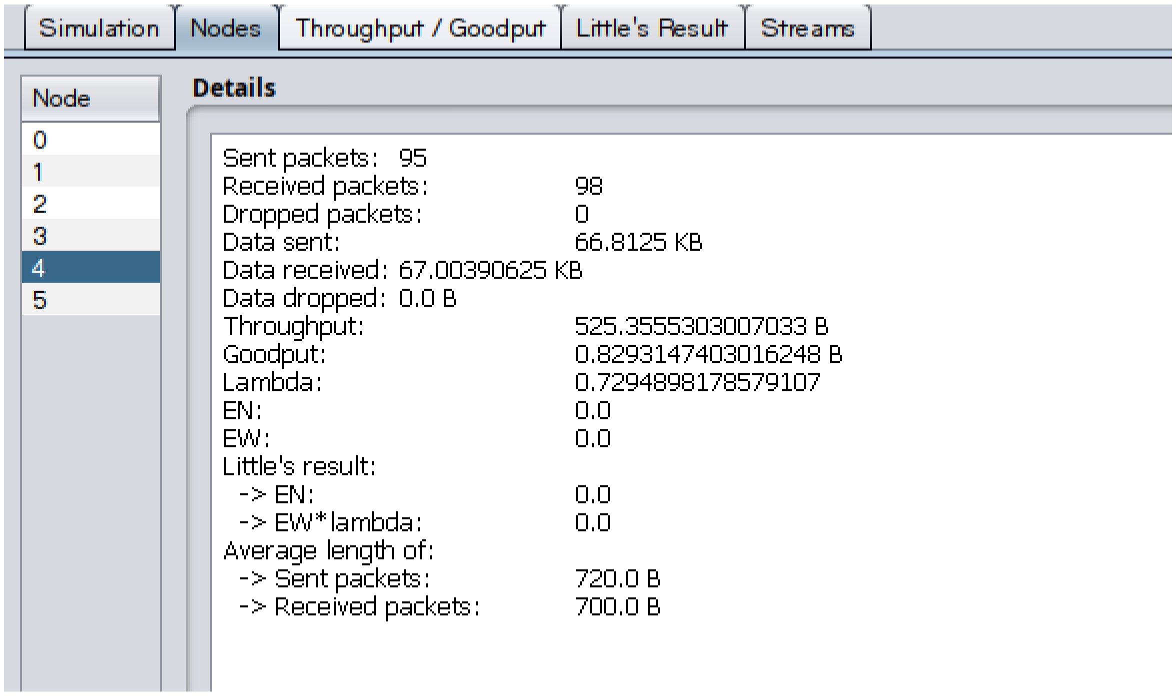
1) Distance vector algorithm



```
File:                       /home/hit-vaghela/Desktop/ns-allinone-3.40/ns-3.40/rip-simple-routing.tr
Lines on file:              2064
Total enqueued packets:     688
Total sent packets:         688
Total received packets:     688
Total dropped packets:      0
Total simulation time:      130.228 seconds
Time of analisys:           0s
```

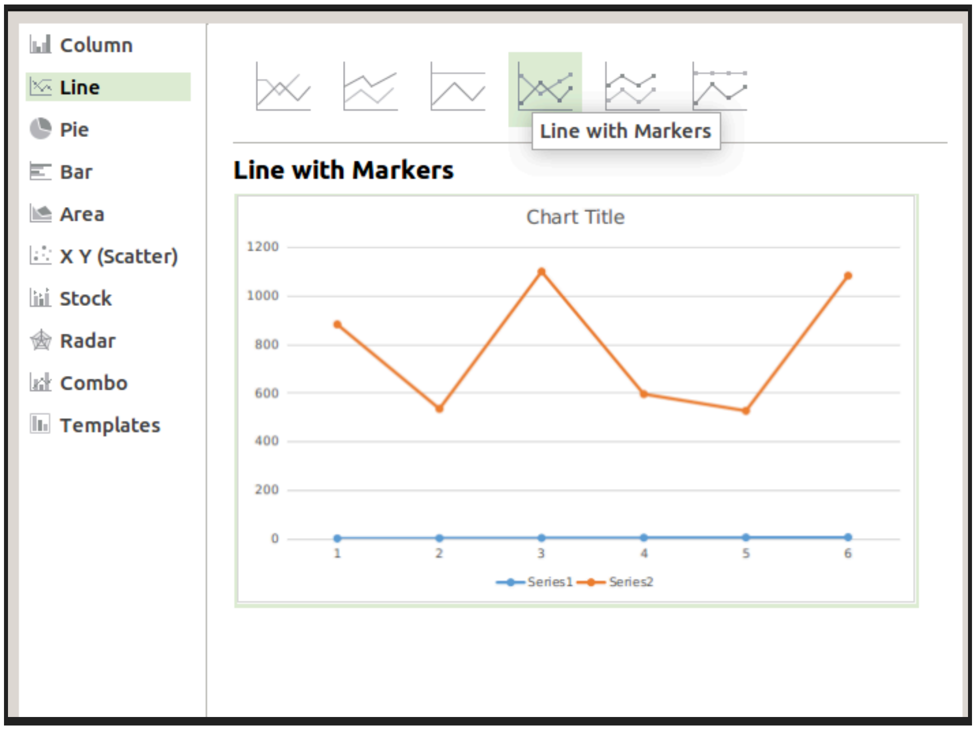


Node	Throughput	Goodput
0	880.7936849218294	0.27643824676720824
1	533.4336701784562	0.27643824676720824
2	1098.166292963111	0.8293147403016248
3	594.0965076634825	0.8293147403016248
4	525.3555303007033	0.8293147403016248
5	1081.1499831065514	0.8293147403016248

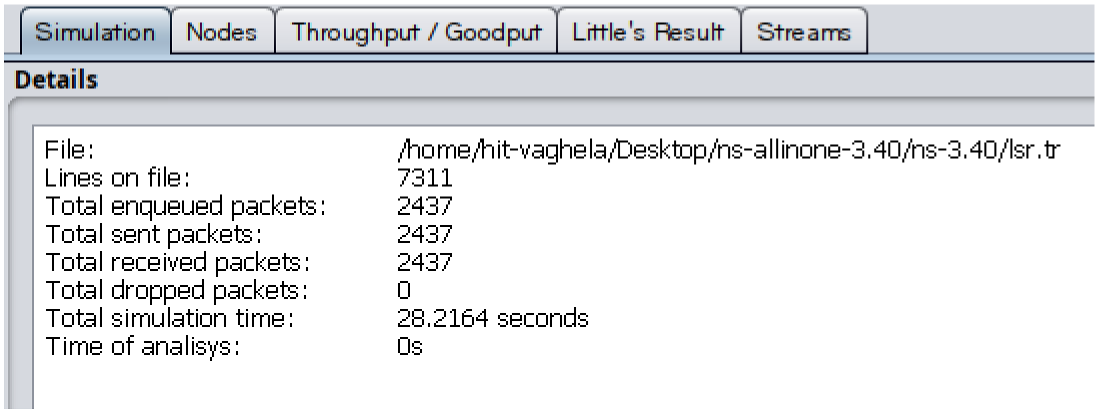


```
Sent packets:   95
Received packets:           98
Dropped packets:            0
Data sent:                  66.8125 KB
Data received:  67.00390625 KB
Data dropped:   0.0 B
Throughput:                 525.3555303007033 B
Goodput:                    0.8293147403016248 B
Lambda:                     0.7294898178579107
EN:                         0.0
EW:                         0.0
Little's result:
  -> EN:                    0.0
  -> EW*lambda:             0.0
Average length of:
  -> Sent packets:          720.0 B
  -> Received packets:      700.0 B
```

Throughput:

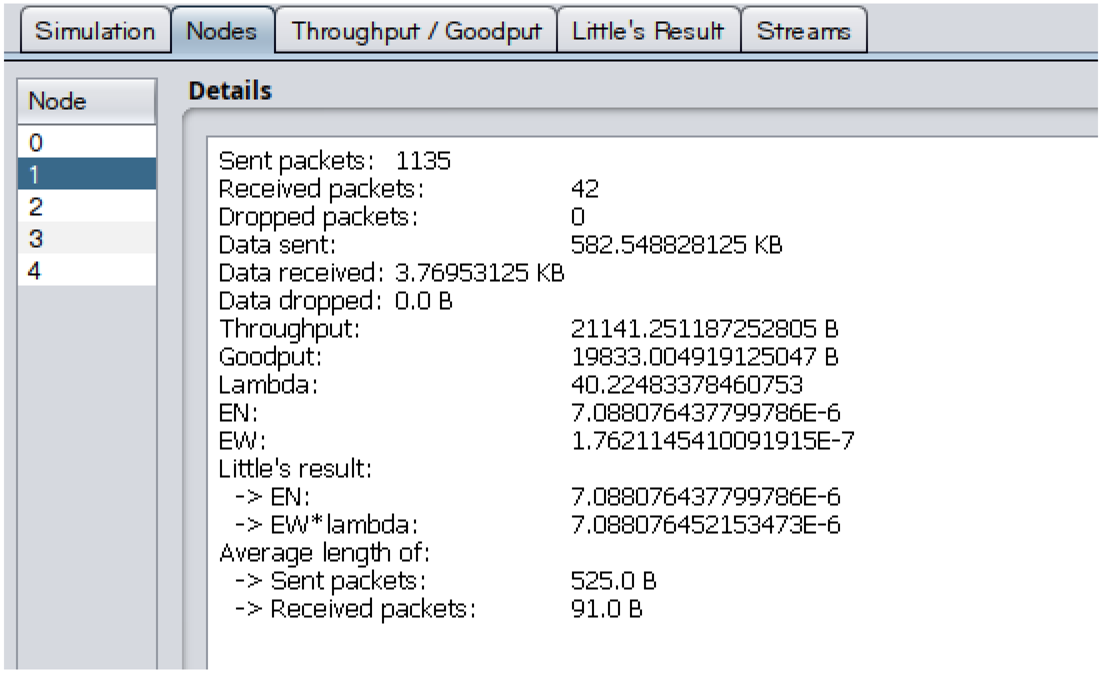


Link - state routing :

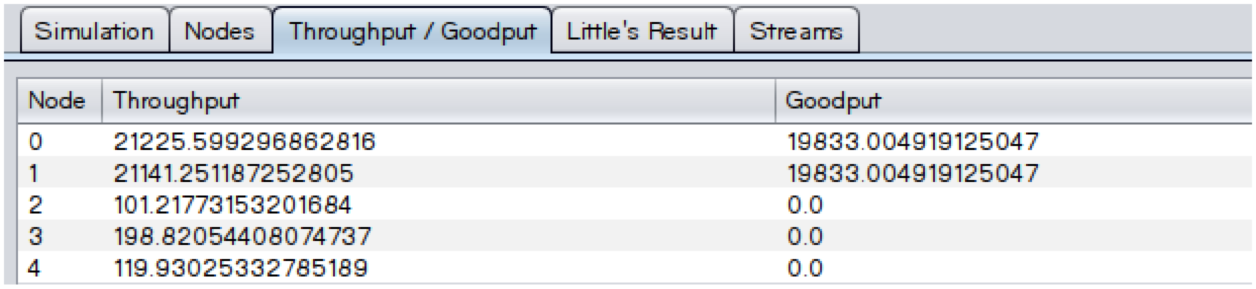


Details

```
File:                          /home/hit-vaghela/Desktop/ns-allinone-3.40/ns-3.40/lsr.tr
Lines on file:                 7311
Total enqueued packets:        2437
Total sent packets:            2437
Total received packets:        2437
Total dropped packets:         0
Total simulation time:         28.2164 seconds
Time of analisys:              0s
```

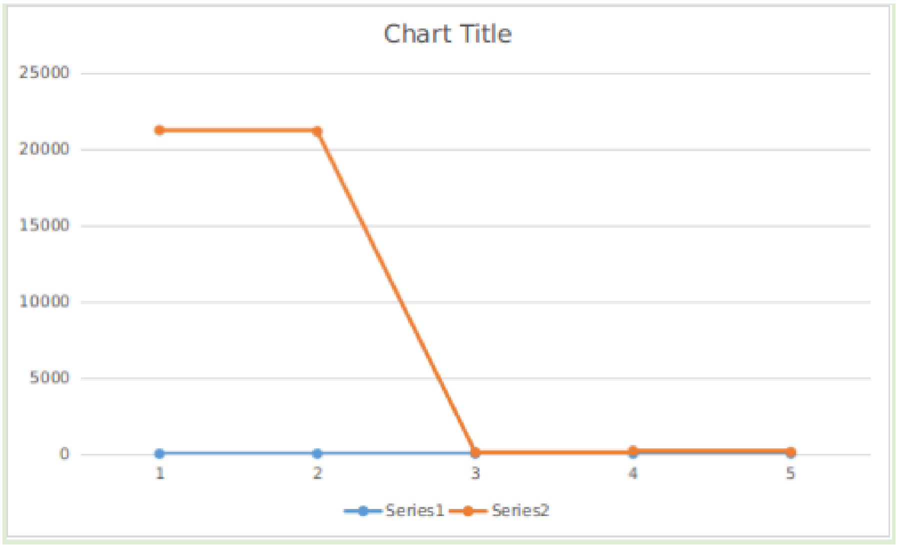


```
Sent packets:   1135
Received packets:           42
Dropped packets:            0
Data sent:                  582.548828125 KB
Data received:  3.76953125 KB
Data dropped:   0.0 B
Throughput:                 21141.251187252805 B
Goodput:                    19833.004919125047 B
Lambda:                     40.22483378460753
EN:                         7.088076437799786E-6
EW:                         1.7621145410091915E-7
Little's result:
  -> EN:                    7.088076437799786E-6
  -> EW*lambda:             7.088076452153473E-6
Average length of:
  -> Sent packets:          525.0 B
  -> Received packets:      91.0 B
```



Node	Throughput	Goodput
0	21225.599296862816	19833.004919125047
1	21141.251187252805	19833.004919125047
2	101.21773153201684	0.0
3	198.82054408074737	0.0
4	119.93025332785189	0.0

Throughput:



5 . References

1. Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Pearson Education.
2. Kurose, J. F., & Ross, K. W. (2017). Computer Networking: A Top-Down Approach (7th ed.). Pearson.
3. Perlman, R. (2001). Interconnections: Bridges, Routers, Switches, and Internetworking Protocols. Addison-Wesley.
4. Forouzan, B. A., & Fegan, S. C. (2012). Data Communications and Networking (5th ed.). McGraw-Hill Education.
5. Zhang, Z., & Zhang, Y. (2016). Computer Networking: Principles, Protocols and Practice. CreateSpace Independent Publishing Platform.