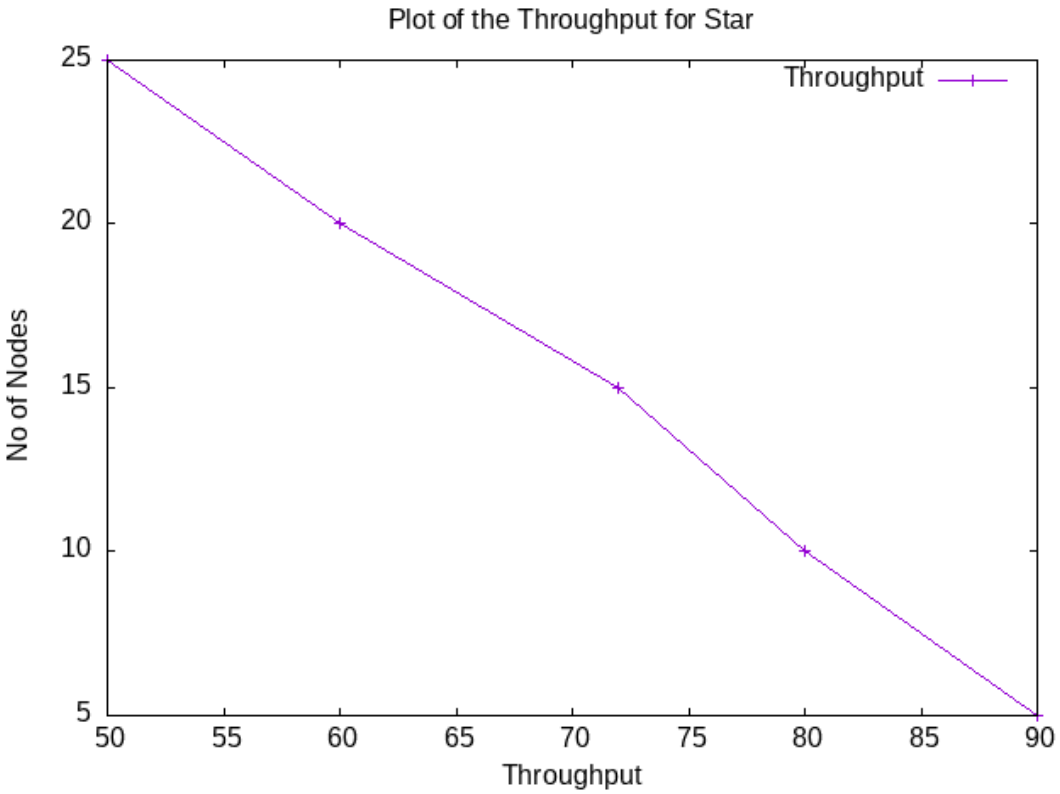
Star Program

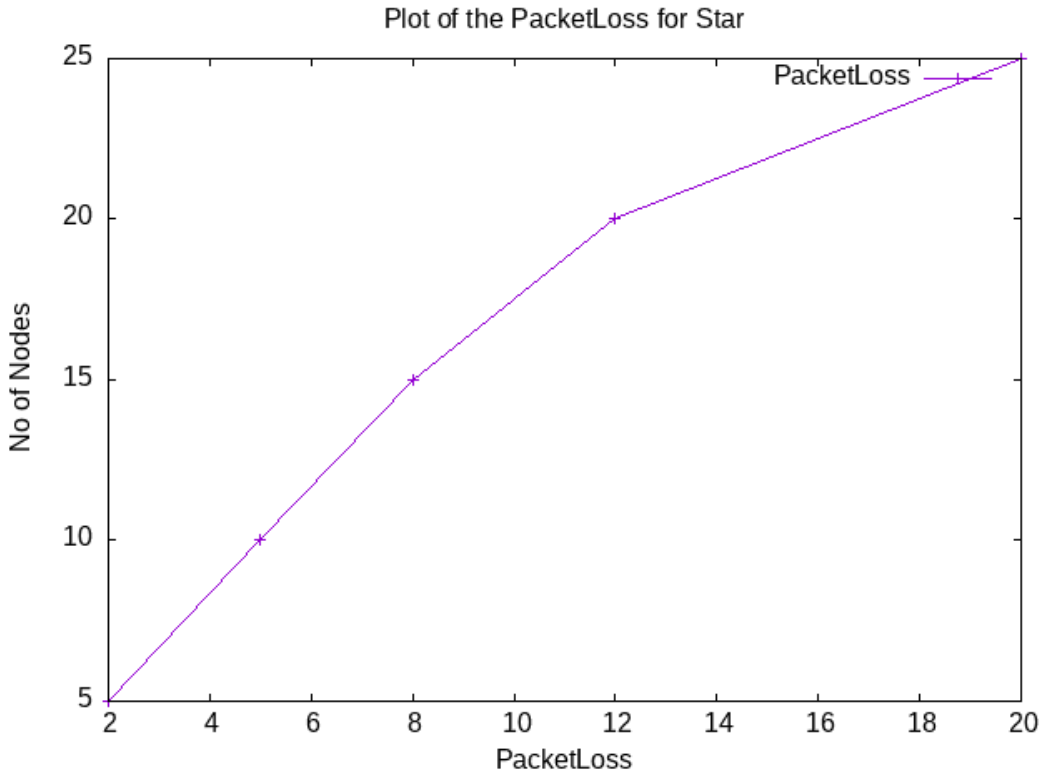
```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/point-to-point-module.h"

using namespace ns3;
NS_LOG_COMPONENT_DEFINE("StarTopology");

void PrintStats(Ptr<FlowMonitor> monitor, Ptr<Ipv4FlowClassifier>
classifier) {
FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();
double totalThroughput = 0.0;
double totalPacketLoss = 0.0;
uint32_t flowCount = 0;

for (auto i : stats) {
FlowId flowId = i.first;
FlowMonitor::FlowStats s = i.second;

Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(flowId);
std::cout << "Flow from " << t.sourceAddress << " to " <<
t.destinationAddress << std::endl;
std::cout << " Transmitted Bytes: " << s.txBytes << std::endl;
std::cout << " Received Bytes: " << s.rxBytes << std::endl;
std::cout << " Lost Packets: " << s.lostPackets << std::endl;

if (s.rxPackets > 0) {
double delay = (s.delaySum.GetSeconds() * 1000 / s.rxPackets);
std::cout << " Average Delay (ms): " << delay << std::endl;
double throughput = (s.rxBytes * 8.0 / s.timeLastRxPacket.GetSeconds() /
1000000.0);
std::cout << " Throughput (Mbps): " << throughput << std::endl;

totalThroughput += throughput;
flowCount++;
} else {
std::cout << " No packets received." << std::endl;
}

totalPacketLoss += s.lostPackets;
}

if (flowCount > 0) {
```

```
std::cout << "Average Throughput (Mbps): " << (totalThroughput / flowCount)
<< std::endl;
} else {
std::cout << "No flows received any packets." << std::endl;
}

double averagePacketLoss = totalPacketLoss / stats.size();
std::cout << "Average Packet Loss: " << averagePacketLoss << " packets" <<
std::endl;
}

int main(int argc, char* argv[]) {
uint32_t nNodes = 15;
CommandLine cmd(__FILE__);
cmd.AddValue("nNodes", "Number of peripheral nodes", nNodes);
cmd.Parse(argc, argv);

nNodes = std::max(nNodes, 2u);

NodeContainer centralNode;
centralNode.Create(1);
NodeContainer peripheralNodes;
peripheralNodes.Create(nNodes);
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", StringValue("5000Mbps"));
p2p.SetChannelAttribute("Delay", TimeValue(MicroSeconds(5)));
NetDeviceContainer centralDevice, peripheralDevices[nNodes];
Ipv4InterfaceContainer peripheralInterfaces[nNodes];

InternetStackHelper stack;
stack.Install(centralNode);
stack.Install(peripheralNodes);

Ipv4AddressHelper address;

for (uint32_t i = 0; i < nNodes; ++i) {
NodeContainer linkNodes = NodeContainer(centralNode.Get(0),
peripheralNodes.Get(i));
NetDeviceContainer linkDevices = p2p.Install(linkNodes);
std::ostringstream subnet;
subnet << "10.1." << i + 1 << ".0";
address.SetBase(subnet.str().c_str(), "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(linkDevices);

centralDevice.Add(linkDevices.Get(0));
peripheralDevices[i].Add(linkDevices.Get(1));
peripheralInterfaces[i] = interfaces;
}
for (uint32_t i = 1; i < nNodes; i += 2) {
UdpEchoServerHelper echoServer(1000 + i);
```

```
ApplicationContainer serverApps =
echoServer.Install(peripheralNodes.Get(i));
serverApps.Start(Seconds(1.0));
serverApps.Stop(Seconds(20.0));
}
for (uint32_t i = 0; i < nNodes; i += 2) {
if (i + 1 < nNodes) {
UdpEchoClientHelper echoClient(peripheralInterfaces[i + 1].GetAddress(1),
1000 + (i + 1));
echoClient.SetAttribute("MaxPackets", UintegerValue(10));
echoClient.SetAttribute("Interval", TimeValue(Seconds(.01)));
echoClient.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientApps =
echoClient.Install(peripheralNodes.Get(i));
clientApps.Start(Seconds(1.0));
clientApps.Stop(Seconds(20.0));
}
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

FlowMonitorHelper flowMonitorHelper;
Ptr<FlowMonitor> monitor = flowMonitorHelper.InstallAll();
Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowMonitorHelper.GetClassifier());

p2p.EnablePcapAll("star_topology_example");
AnimationInterface anim("star_topology_example.xml");

Simulator::Stop(Seconds(20.0));
Simulator::Run();

PrintStats(monitor, classifier);

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

OUTPUT

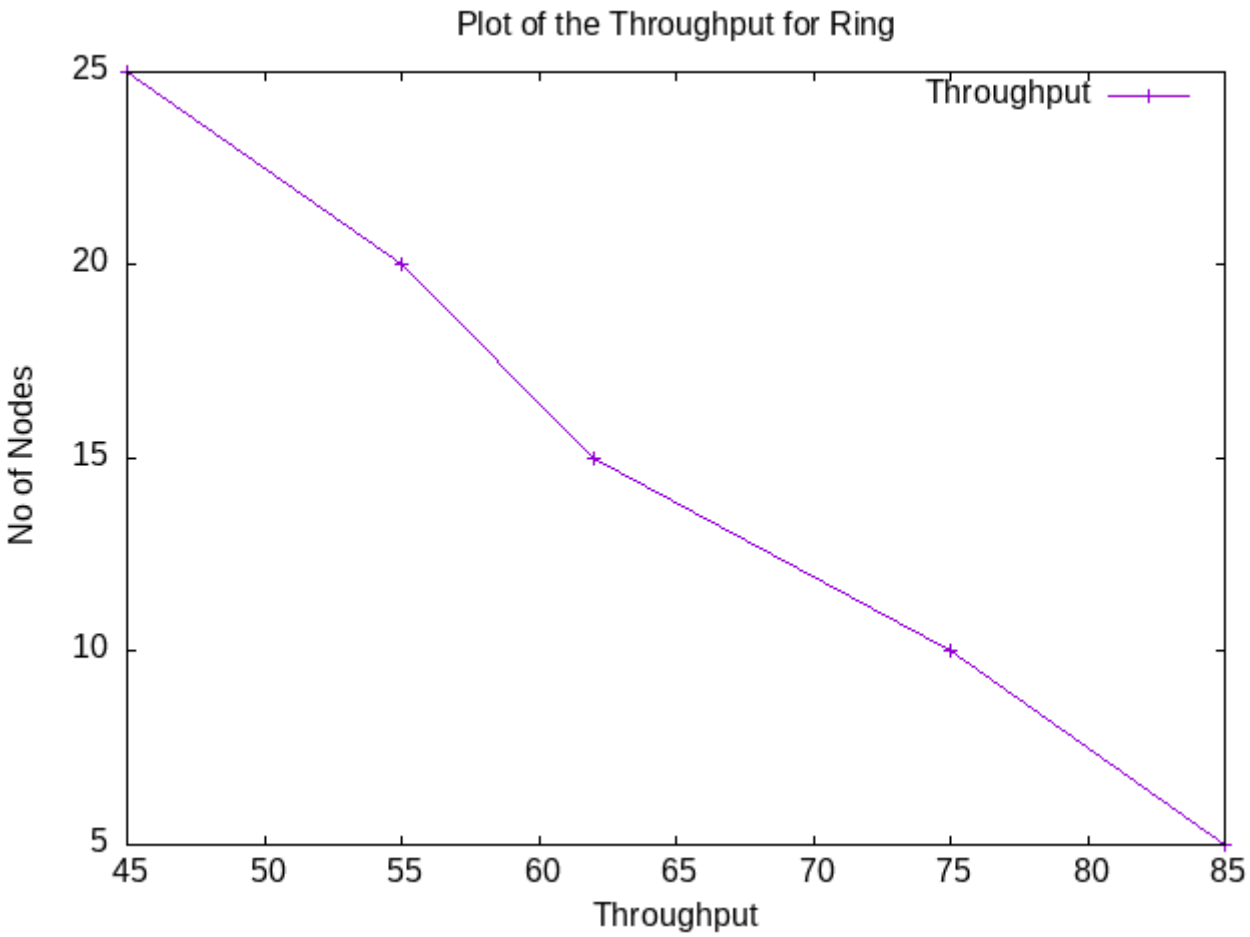

Plot of the Throughput for Star
Throughput
No of Nodes
Throughput
25
20
15
10
5
50
55
60
65
70
75
80
85
90

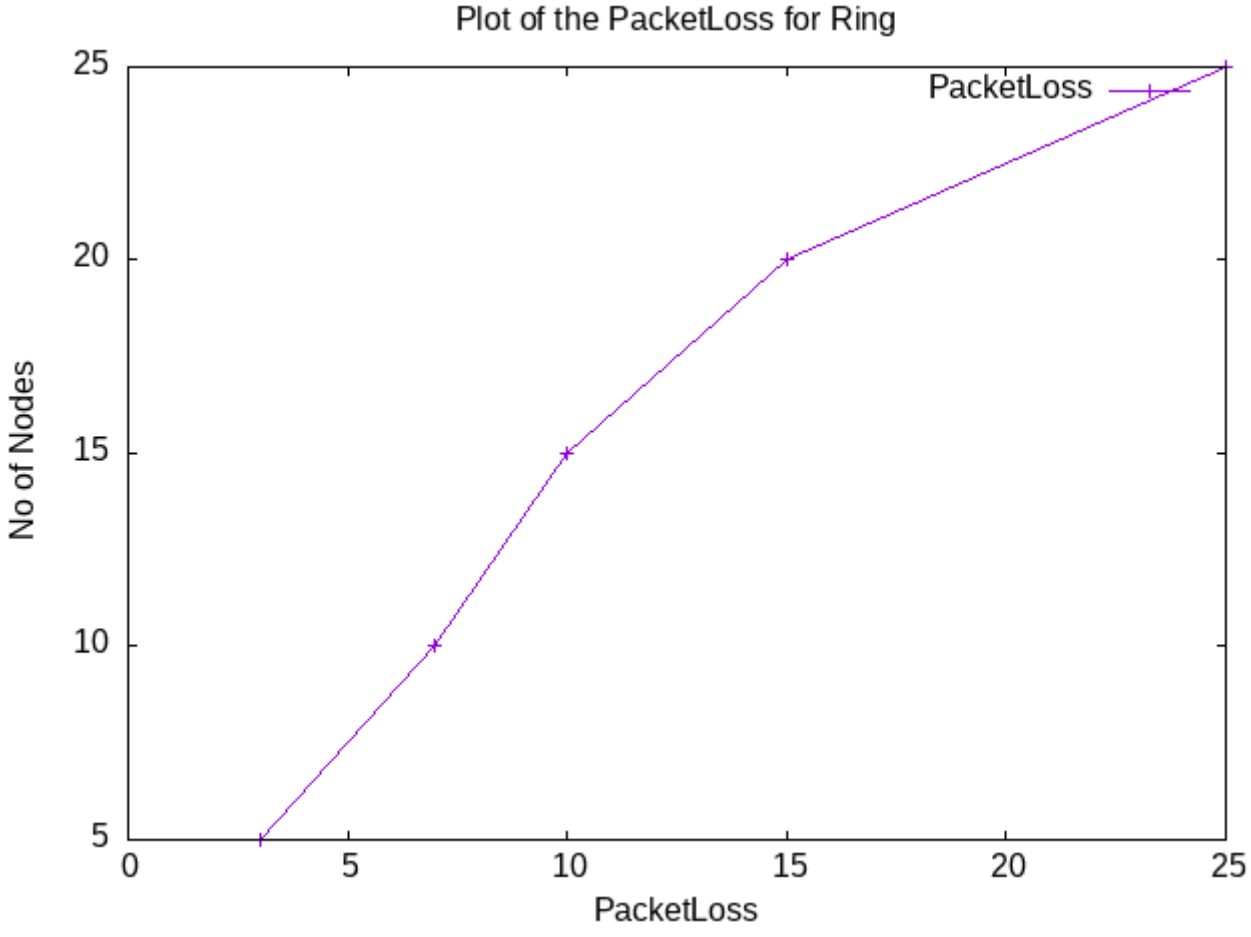

Plot of the PacketLoss for Star
PacketLoss
No of Nodes
PacketLoss
25
20
15
10
5
2
4
6
8
10
12
14
16
18
20

Ring Program

```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/mobility-module.h"
#include "ns3/netanim-module.h"
#include <cassert>
#include <fstream>
#include <iostream>
#include <string>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("RING_TOPOLOGY");

int
main(int argc, char* argv[])
{
Config::SetDefault("ns3::OnOffApplication::PacketSize", UintegerValue(250));
Config::SetDefault("ns3::OnOffApplication::DataRate", StringValue("5kb/s"));
uint32_t N = 9;
CommandLine cmd(__FILE__);
cmd.AddValue("nNodes", "Number of clientNodes to place in the star", N);
cmd.Parse(argc, argv);
NS_LOG_INFO("Create clientNodes.");
NodeContainer serverNode;
NodeContainer clientNodes;
serverNode.Create(1);
clientNodes.Create(N - 1);
NodeContainer allNodes = NodeContainer(serverNode, clientNodes);
InternetStackHelper internet;
internet.Install(allNodes);
std::vector<NodeContainer> nodeAdjacencyList(N - 1);
for (uint32_t i = 0; i < nodeAdjacencyList.size(); ++i)
{
nodeAdjacencyList[i] = NodeContainer(serverNode, clientNodes.Get(i));
}
NS_LOG_INFO("Create channels.");
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("2ms"));
std::vector<NetDeviceContainer> deviceAdjacencyList(N - 1);
for (uint32_t i = 0; i < deviceAdjacencyList.size(); ++i)
{
deviceAdjacencyList[i] = p2p.Install(nodeAdjacencyList[i]);
}
NS_LOG_INFO("Assign IP Addresses.");
```

```
Ipv4AddressHelper ipv4;
std::vector<Ipv4InterfaceContainer> interfaceAdjacencyList(N - 1);
for (uint32_t i = 0; i < interfaceAdjacencyList.size(); ++i)
{
std::ostringstream subnet;
subnet << "10.1." << i + 1 << ".0";
ipv4.SetBase(subnet.str().c_str(), "255.255.255.0");
interfaceAdjacencyList[i] = ipv4.Assign(deviceAdjacencyList[i]);
}
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
uint16_t port = 50000;
Address sinkLocalAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
PacketSinkHelper sinkHelper("ns3::TcpSocketFactory", sinkLocalAddress);
ApplicationContainer sinkApp = sinkHelper.Install(serverNode);
sinkApp.Start(Seconds(1.0));
sinkApp.Stop(Seconds(10.0));
OnOffHelper clientHelper("ns3::TcpSocketFactory", Address());
clientHelper.SetAttribute("OnTime",
StringValue("ns3::ConstantRandomVariable[Constant=1]"));
clientHelper.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
ApplicationContainer clientApps;
for (uint32_t i = 0; i < clientNodes.GetN(); ++i)
{
AddressValue remoteAddress(
InetSocketAddress(interfaceAdjacencyList[i].GetAddress(0), port));
clientHelper.SetAttribute("Remote", remoteAddress);
clientApps.Add(clientHelper.Install(clientNodes.Get(i)));
}
clientApps.Start(Seconds(1.0));
clientApps.Stop(Seconds(10.0));

// configure tracing
AsciiTraceHelper ascii;
p2p.EnablePcapAll("tcp-star-server");

NS_LOG_INFO("Run Simulation.");
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(allNodes);
AnimationInterface anim("second_script.xml");
for (uint32_t i = 0; i < clientNodes.GetN(); ++i) {
anim.SetConstantPosition(clientNodes.Get(i), i * 3, 0.0);
}
anim.SetConstantPosition(serverNode.Get(0), 11.0, 22.0);

Simulator::Run();
Simulator::Destroy();
NS_LOG_INFO("Done.");

return 0; }
```


Plot of the Throughput for Ring
Throughput
No of Nodes
Throughput
25
20
15
10
5
45
50
55
60
65
70
75
80
85

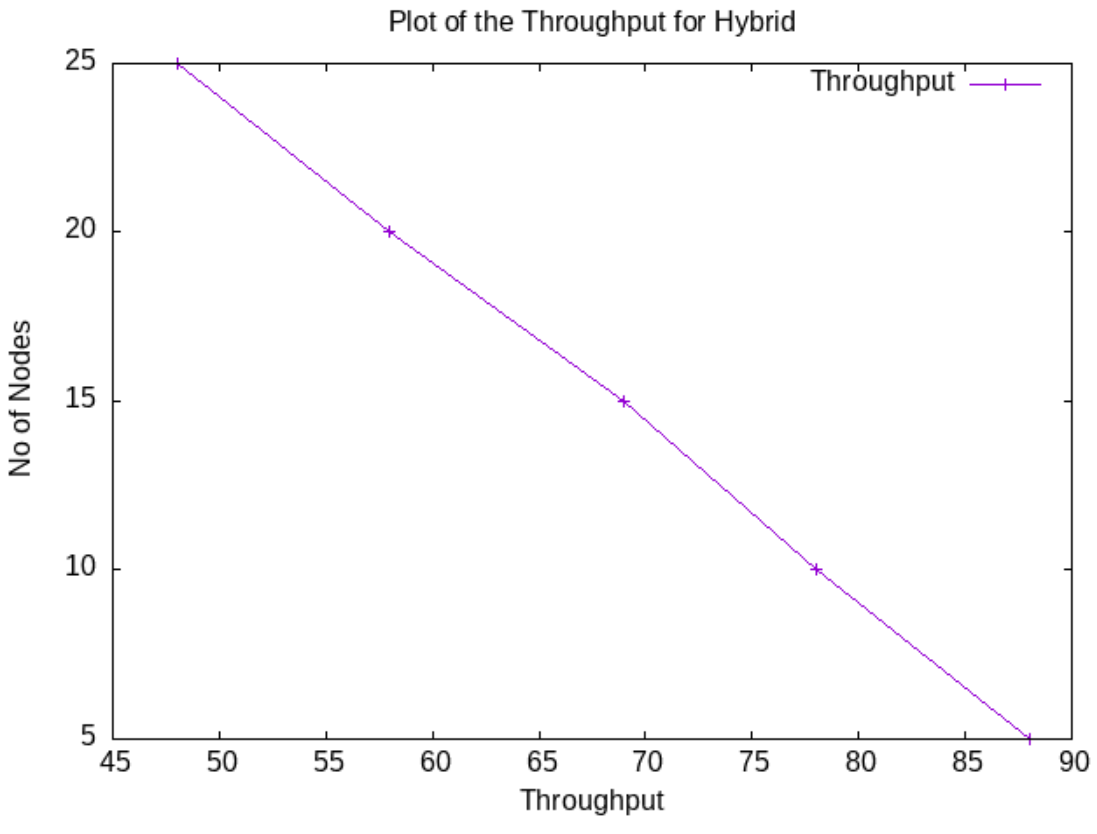

Plot of the PacketLoss for Ring
PacketLoss
No of Nodes
PacketLoss
25
20
15
10
5
0
5
10
15
20
25

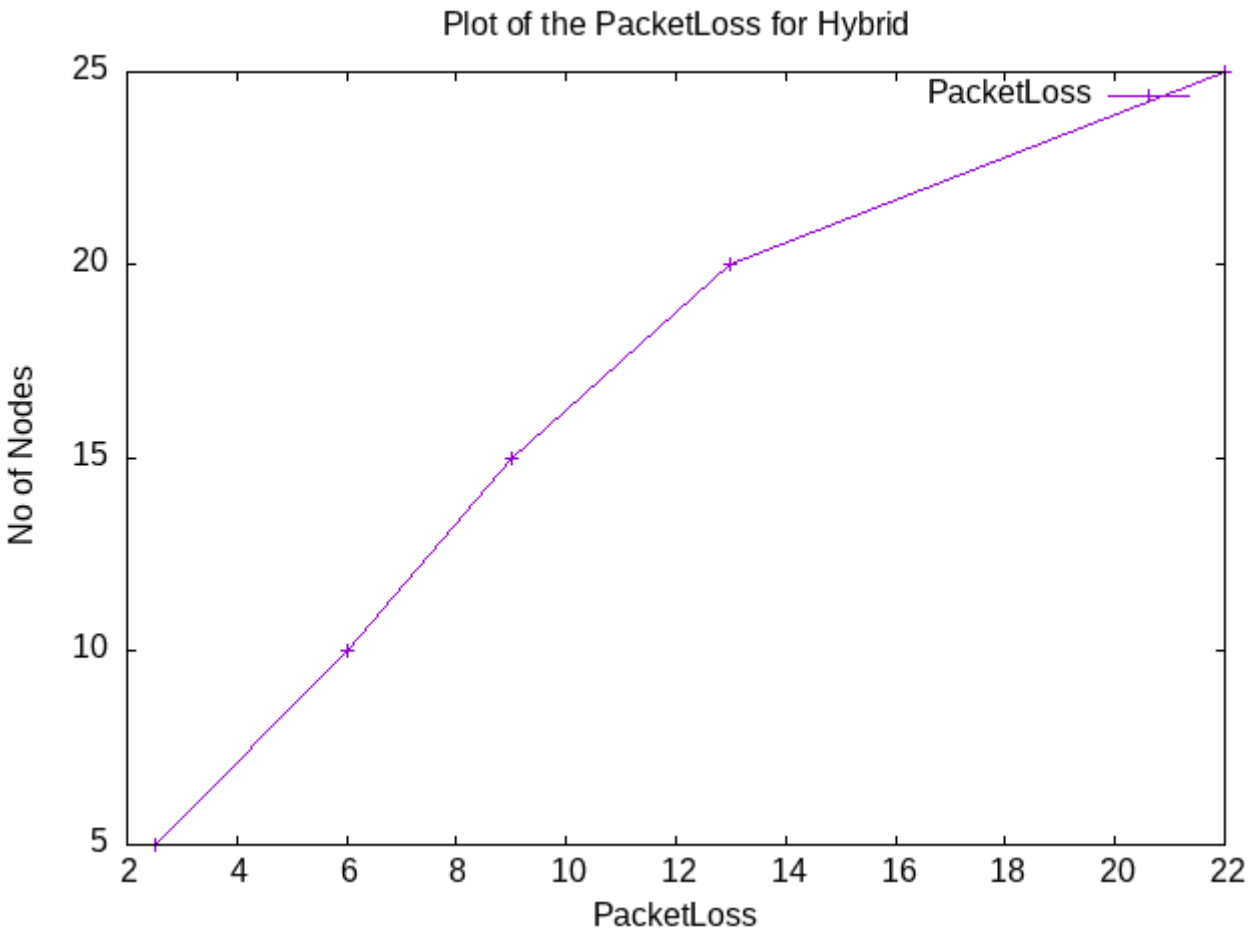
Hybrid Program

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

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("HybridTopology");

void PrintStats(Ptr<FlowMonitor> monitor, Ptr<Ipv4FlowClassifier>
classifier) {
FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();
double totalThroughput = 0.0;
double totalPacketLoss = 0.0;
uint32_t flowCount = 0;

for (auto i : stats) {
FlowId flowId = i.first;
FlowMonitor::FlowStats s = i.second;

Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(flowId);
std::cout << "Flow from " << t.sourceAddress << " to " <<
t.destinationAddress << std::endl;
std::cout << " Transmitted Bytes: " << s.txBytes << std::endl;
std::cout << " Received Bytes: " << s.rxBytes << std::endl;
std::cout << " Lost Packets: " << s.lostPackets << std::endl;

if (s.rxPackets > 0) {
double delay = (s.delaySum.GetSeconds() * 1000 / s.rxPackets);
std::cout << " Average Delay (ms): " << delay << std::endl;
double throughput = (s.rxBytes * 8.0 / s.timeLastRxPacket.GetSeconds() /
1000000.0);
std::cout << " Throughput (Mbps): " << throughput << std::endl;

totalThroughput += throughput;
flowCount++;
} else {
std::cout << " No packets received." << std::endl;
}

totalPacketLoss += s.lostPackets;
}

if (flowCount > 0) {
```

```
std::cout << "Average Throughput (Mbps): " << (totalThroughput / flowCount)
<< std::endl;
} else {
std::cout << "No flows received any packets." << std::endl;
}

double averagePacketLoss = totalPacketLoss / stats.size();
std::cout << "Average Packet Loss: " << averagePacketLoss << " packets" <<
std::endl;
}

int main(int argc, char* argv[]) {
uint32_t nNodes = 15;
CommandLine cmd(__FILE__);
cmd.AddValue("nNodes", "Number of peripheral nodes", nNodes);
cmd.Parse(argc, argv);

nNodes = std::max(nNodes, 3u); // Minimum 3 nodes for ring topology

NodeContainer centralNode;
centralNode.Create(1);
NodeContainer peripheralNodes;
peripheralNodes.Create(nNodes);
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", StringValue("5000Mbps"));
p2p.SetChannelAttribute("Delay", TimeValue(MicroSeconds(5)));
NetDeviceContainer centralDevice, peripheralDevices[nNodes];
Ipv4InterfaceContainer peripheralInterfaces[nNodes];

InternetStackHelper stack;
stack.Install(centralNode);
stack.Install(peripheralNodes);

Ipv4AddressHelper address;

// Star Topology
for (uint32_t i = 0; i < nNodes; ++i) {
NodeContainer linkNodes = NodeContainer(centralNode.Get(0),
peripheralNodes.Get(i));
NetDeviceContainer linkDevices = p2p.Install(linkNodes);
std::ostringstream subnet;
subnet << "10.1." << i + 1 << ".0";
address.SetBase(subnet.str().c_str(), "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(linkDevices);

centralDevice.Add(linkDevices.Get(0));
peripheralDevices[i].Add(linkDevices.Get(1));
peripheralInterfaces[i] = interfaces;
}

// Star Topology
```

```
for (uint32_t i = 0; i < nNodes; ++i) {
NodeContainer linkNodes = NodeContainer(centralNode.Get(0),
peripheralNodes.Get(i));
NetDeviceContainer linkDevices = p2p.Install(linkNodes);
std::ostringstream subnet;
subnet << "10.1." << i + 1 << ".0";
address.SetBase(subnet.str().c_str(), "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(linkDevices);

centralDevice.Add(linkDevices.Get(0));
peripheralDevices[i].Add(linkDevices.Get(1));
peripheralInterfaces[i] = interfaces;
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables();

FlowMonitorHelper flowMonitorHelper;
Ptr<FlowMonitor> monitor = flowMonitorHelper.InstallAll();
Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowMonitorHelper.GetClassifier());

p2p.EnablePcapAll("star_topology_example");
AnimationInterface anim("star_topology_example.xml");

Simulator::Stop(Seconds(20.0));
Simulator::Run();

PrintStats(monitor, classifier);

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




Plot of the PacketLoss for Hybrid
PacketLoss
No of Nodes
PacketLoss
25
20
15
10
5
2
4
6
8
10
12
14
16
18
20
22