simple topology of Three nodes

```
#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/flow-monitor-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ThreeNodeTopology");

int main(int argc, char *argv[]) {
    // Logging
    LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
    LogComponentEnable("UdpServer", LOG_LEVEL_INFO);

    // Create nodes
    NodeContainer nodes;
    nodes.Create(3);

    // Setup point-to-point links with variable latency
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("1Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("1ms"));

    NetDeviceContainer dev0_1 = p2p.Install(nodes.Get(0), nodes.Get(1));
    NetDeviceContainer dev1_2 = p2p.Install(nodes.Get(1), nodes.Get(2));

    // Install internet stack
    InternetStackHelper stack;
    stack.Install(nodes);

    // Enable IP forwarding on Node1
    Ptr<Ipv4> ipv4Node1 = nodes.Get(1)->GetObject<Ipv4>();
    ipv4Node1->SetAttribute("IpForward", BooleanValue(true));

    // Assign IP addresses
    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.1.1.0", "255.255.255.0");
    ipv4.Assign(dev0_1);
    ipv4.SetBase("10.1.2.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = ipv4.Assign(dev1_2);
```

```
// Add global routing
Ipv4GlobalRoutingHelper::PopulateRoutingTables();

// Server on Node2 (Port 5000)
UdpServerHelper server(5000);
ApplicationContainer serverApp = server.Install(nodes.Get(2));
serverApp.Start(Seconds(1.0));
serverApp.Stop(Seconds(10.0));

// Client on Node0 (to Server Port 5000)
UdpClientHelper client(interfaces.GetAddress(1), 5000);
client.SetAttribute("MaxPackets", UintegerValue(1000));
client.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
client.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientApp = client.Install(nodes.Get(0));
clientApp.Start(Seconds(2.0));
clientApp.Stop(Seconds(9.0));

// Add second server (Port 5001) and client
UdpServerHelper server2(5001);
ApplicationContainer serverApp2 = server2.Install(nodes.Get(2));
serverApp2.Start(Seconds(1.0));
serverApp2.Stop(Seconds(10.0));

UdpClientHelper client2(interfaces.GetAddress(1), 5001);
client2.SetAttribute("MaxPackets", UintegerValue(1000));
client2.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
client2.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientApp2 = client2.Install(nodes.Get(0));
clientApp2.Start(Seconds(2.0));
clientApp2.Stop(Seconds(9.0));

// Enable PCAP tracing in the "scratch" directory
p2p.EnablePcap("scratch/exp1-node0", dev0_1.Get(0), true);  // Node0 -> Node1
p2p.EnablePcap("scratch/exp1-node1", dev1_2.Get(0), true);  // Node1 -> Node2

// Flow Monitor for throughput measurement
FlowMonitorHelper flowMon;
Ptr<FlowMonitor> monitor = flowMon.InstallAll();

// Set explicit simulation stop time
Simulator::Stop(Seconds(10.0));  // 🔑 Fixes hanging issue
```

```
    Simulator::Run();
    // Calculate throughput here using FlowMonitor data
    Simulator::Destroy();
    return 0;
}
```

two bus topology(LAN) of four nodes each

```
#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/flow-monitor-module.h" // Required for FlowMonitor
#include "ns3/netanim-module.h"      // Required for TraceMatrices

using namespace ns3;

int main() {
    // 1. Enable basic logging
    LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
    LogComponentEnable("UdpServer", LOG_LEVEL_INFO);

    // 2. Create topology
    NodeContainer lan1, lan2;
    lan1.Create(4); // LAN1: n0-n3
    lan2.Create(4); // LAN2: n4-n7

    // 3. Connect LANs via P2P (n3-n4)
    NodeContainer p2pNodes;
    p2pNodes.Add(lan1.Get(3));
    p2pNodes.Add(lan2.Get(0));

    // 4. Configure links
    CsmaHelper csma;
    csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
    csma.SetChannelAttribute("Delay", TimeValue(NanoSeconds(6560)));

    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));

    // 5. Install devices
```

```
NetDeviceContainer lan1Devs = csma.Install(lan1);
NetDeviceContainer lan2Devs = csma.Install(lan2);
NetDeviceContainer p2pDevs = p2p.Install(p2pNodes);

// 6. Install internet stacks
InternetStackHelper stack;
stack.Install(lan1);
stack.Install(lan2);

// 7. Assign IPs
Ipv4AddressHelper ip;
ip.SetBase("10.1.1.0", "255.255.255.0");
ip.Assign(lan1Devs);

ip.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer lan2Ifs = ip.Assign(lan2Devs);

ip.SetBase("10.1.3.0", "255.255.255.0");
ip.Assign(p2pDevs);

// 8. Enable routing
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
lan1.Get(3)->GetObject<Ipv4>()->SetAttribute("IpForward", BooleanValue(true));
lan2.Get(0)->GetObject<Ipv4>()->SetAttribute("IpForward", BooleanValue(true));

// 9. Setup server on LAN2's n2 (10.1.2.3)
UdpServerHelper server(9);
ApplicationContainer serverApp = server.Install(lan2.Get(2));
serverApp.Start(Seconds(1.0));
serverApp.Stop(Seconds(10.0));

// 10. Setup client on LAN1's n1
UdpClientHelper client(lan2Ifs.GetAddress(2), 9);
client.SetAttribute("MaxPackets", UintegerValue(100));
client.SetAttribute("Interval", TimeValue(MilliSeconds(100)));
client.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientApp = client.Install(lan1.Get(1));
clientApp.Start(Seconds(2.0));
clientApp.Stop(Seconds(9.0));

// 11. Enable PCAP tracing
csma.EnablePcap("lab6-lan1", lan1Devs.Get(1), true);
csma.EnablePcap("lab6-lan2", lan2Devs.Get(2), true);
p2p.EnablePcap("lab6-p2p", p2pDevs.Get(0), true);
```

```
    // 12. Setup FlowMonitor (for .xml output)
    FlowMonitorHelper flowMon;
    Ptr<FlowMonitor> monitor = flowMon.InstallAll();

    // 13. Setup TraceMatrices (for .tr output)
    AsciiTraceHelper ascii;
    csma.EnableAsciiAll(ascii.CreateFileStream("lab6.tr"));

    // 14. Run simulation
    Simulator::Stop(Seconds(11.0));
    Simulator::Run();

    // 15. Save FlowMonitor results
    monitor->SerializeToXmlFile("lab6-flowmon.xml", true, true);

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

Enhanced Multi-Node Topology with Error Model and Multiple UDP Flows

```
#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/flow-monitor-module.h"
#include "ns3/error-model.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("FourNodeTopologyWithErrorModel");

int main(int argc, char *argv[]) {
    // Enable logging
    LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
    LogComponentEnable("UdpServer", LOG_LEVEL_INFO);

    // Create 4 nodes: Node0, Node1, Node2, Node3
    NodeContainer nodes;
    nodes.Create(4);
```

```
// Setup point-to-point links between nodes:
// Link0: Node0 <-> Node1
// Link1: Node1 <-> Node2 (error model applied)
// Link2: Node2 <-> Node3
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("1ms"));

NetDeviceContainer dev0_1 = p2p.Install(nodes.Get(0), nodes.Get(1));
NetDeviceContainer dev1_2 = p2p.Install(nodes.Get(1), nodes.Get(2));
NetDeviceContainer dev2_3 = p2p.Install(nodes.Get(2), nodes.Get(3));


        Ptr<RateErrorModel> em = CreateObject<RateErrorModel>();
        em->SetAttribute("ErrorRate", DoubleValue(0.01));
        // Explicitly set the error unit to per bit
        em->SetAttribute("ErrorUnit", StringValue("ERROR_UNIT_PACKET"));
        dev1_2.Get(1)->SetAttribute("ReceiveErrorModel", PointerValue(em));


// Install internet stack on all nodes
InternetStackHelper stack;
stack.Install(nodes);

// Enable IP forwarding on intermediate nodes (Node1 and Node2)
for (uint32_t i = 1; i < nodes.GetN()-1; i++) {
    Ptr<Ipv4> ipv4 = nodes.Get(i)->GetObject<Ipv4>();
    ipv4->SetAttribute("IpForward", BooleanValue(true));
}

// Assign IP addresses for each link
Ipv4AddressHelper ipv4;
// Link0: Node0 <-> Node1, network 10.1.1.0/24
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer if0_1 = ipv4.Assign(dev0_1);
// Link1: Node1 <-> Node2, network 10.1.2.0/24
ipv4.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer if1_2 = ipv4.Assign(dev1_2);
// Link2: Node2 <-> Node3, network 10.1.3.0/24
ipv4.SetBase("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer if2_3 = ipv4.Assign(dev2_3);

// Populate global routing tables
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

```
// UDP Flow 1: from Node0 (client) to Node2 (server) on port 5000
UdpServerHelper udpServer1(5000);
ApplicationContainer serverApp1 = udpServer1.Install(nodes.Get(2));
serverApp1.Start(Seconds(1.0));
serverApp1.Stop(Seconds(10.0));

UdpClientHelper udpClient1(if1_2.GetAddress(1), 5000);
udpClient1.SetAttribute("MaxPackets", UintegerValue(1000));
udpClient1.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
udpClient1.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientApp1 = udpClient1.Install(nodes.Get(0));
clientApp1.Start(Seconds(2.0));
clientApp1.Stop(Seconds(9.0));

// UDP Flow 2: from Node0 (client) to Node3 (server) on port 5001
UdpServerHelper udpServer2(5001);
ApplicationContainer serverApp2 = udpServer2.Install(nodes.Get(3));
serverApp2.Start(Seconds(1.0));
serverApp2.Stop(Seconds(10.0));

UdpClientHelper udpClient2(if2_3.GetAddress(1), 5001);
udpClient2.SetAttribute("MaxPackets", UintegerValue(1000));
udpClient2.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
udpClient2.SetAttribute("PacketSize", UintegerValue(1024));
ApplicationContainer clientApp2 = udpClient2.Install(nodes.Get(0));
clientApp2.Start(Seconds(2.0));
clientApp2.Stop(Seconds(9.0));

// Enable PCAP tracing on each link
p2p.EnablePcap("scratch/q1-node0-1", dev0_1.Get(0), true);
p2p.EnablePcap("scratch/q1-node1-2", dev1_2.Get(0), true);
p2p.EnablePcap("scratch/q1-node2-3", dev2_3.Get(0), true);

// Flow Monitor for performance measurement
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

Simulator::Stop(Seconds(10.0));
Simulator::Run();

// Analyze FlowMonitor statistics
monitor->CheckForLostPackets();
```

```
    Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();

    for (auto iter = stats.begin(); iter != stats.end(); ++iter) {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(iter->first);
        std::cout << "Flow " << iter->first << " (" << t.sourceAddress << " -> " <<
t.destinationAddress << ")\n";
        std::cout << "  Tx Packets: " << iter->second.txPackets << "\n";
        std::cout << "  Rx Packets: " << iter->second.rxPackets << "\n";
        std::cout << "  Lost Packets: " << (iter->second.txPackets - iter->second.rxPackets) << "\n";
        std::cout << "  Packet Loss Ratio: "
                  << ((iter->second.txPackets - iter->second.rxPackets) * 100.0 /
iter->second.txPackets) << "%\n\n";
    }

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

Concurrent UDP and TCP Flow Analysis

```
#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/flow-monitor-module.h"
#include "ns3/ipv4-global-routing-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("ConcurrentFlows");

int main(int argc, char *argv[]) {
    LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
    LogComponentEnable("UdpServer", LOG_LEVEL_INFO);
            LogComponentEnable ("TcpL4Protocol", LOG_LEVEL_INFO);

    // Create nodes
    NodeContainer nodes;
    nodes.Create(3);

    // Setup links
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));

    NetDeviceContainer dev0_1 = p2p.Install(nodes.Get(0), nodes.Get(1));
    NetDeviceContainer dev1_2 = p2p.Install(nodes.Get(1), nodes.Get(2));

    InternetStackHelper stack;
    stack.Install(nodes);

    // Assign IPs
    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.1.1.0", "255.255.255.0");
    ipv4.Assign(dev0_1);
    ipv4.SetBase("10.1.2.0", "255.255.255.0");
    Ipv4InterfaceContainer iface1_2 = ipv4.Assign(dev1_2);

    nodes.Get(1)->GetObject<Ipv4>()->SetAttribute("IpForward", BooleanValue(true));
    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    // UDP Server (Node2, port 5000)
    UdpServerHelper udpServer(5000);
    ApplicationContainer udpServerApp = udpServer.Install(nodes.Get(1));
    udpServerApp.Start(Seconds(1.0));
    udpServerApp.Stop(Seconds(10.0));

    // UDP Client (Node0)
    UdpClientHelper udpClient(iface1_2.GetAddress(1), 5000);
    udpClient.SetAttribute("MaxPackets", UintegerValue(1000));
    udpClient.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
    udpClient.SetAttribute("PacketSize", UintegerValue(1024));
    ApplicationContainer udpClientApp = udpClient.Install(nodes.Get(0));
    udpClientApp.Start(Seconds(2.0));
    udpClientApp.Stop(Seconds(9.0));

    // TCP Server (Node2, port 5001)
    PacketSinkHelper tcpSink("ns3::TcpSocketFactory",
InetSocketAddress(Ipv4Address::GetAny(), 5000));
    ApplicationContainer tcpServerApp = tcpSink.Install(nodes.Get(2));
    tcpServerApp.Start(Seconds(1.0));
    tcpServerApp.Stop(Seconds(10.0));

    // TCP Client (Node1)
    BulkSendHelper tcpClient("ns3::TcpSocketFactory",
InetSocketAddress(iface1_2.GetAddress(1), 5001));
    tcpClient.SetAttribute("MaxBytes", UintegerValue(0));
    ApplicationContainer tcpClientApp = tcpClient.Install(nodes.Get(0));
    tcpClientApp.Start(Seconds(2.0));
    tcpClientApp.Stop(Seconds(9.0));

    // Enable PCAP
    p2p.EnablePcapAll("q2");

    // Flow Monitor
    FlowMonitorHelper flowMon;
    Ptr<FlowMonitor> monitor = flowMon.InstallAll();

    Simulator::Stop(Seconds(10.0));
    Simulator::Run();

    // Output metrics
    monitor->CheckForLostPackets();
    FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();
    for (auto &flow : stats) {
        std::cout << "Flow " << flow.first << " (TCP/UDP): Throughput="
                  << flow.second.rxBytes * 8.0 / 9.0 / 1e6 << " Mbps, Loss="
                  << flow.second.lostPackets << "\n";
    }

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

Subnet Creation and Inter-Subnet Routing in NS3

```
#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/flow-monitor-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("InterSubnetRouting");
```

```
int main(int argc, char *argv[])
{
  // Enable logging
  LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
  LogComponentEnable("UdpServer", LOG_LEVEL_INFO);

  // Create nodes
  NodeContainer client, router, server;
  client.Create(1);
  router.Create(1);
  server.Create(1);

  // Create links
  PointToPointHelper p2p;
  p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
  p2p.SetChannelAttribute("Delay", StringValue("2ms"));

  NetDeviceContainer devClientRouter = p2p.Install(client.Get(0), router.Get(0));
  NetDeviceContainer devRouterServer = p2p.Install(router.Get(0), server.Get(0));

  // Install internet stack
  InternetStackHelper stack;
  stack.InstallAll(); // Install on all nodes at once

  // Enable IP forwarding on router
  router.Get(0)->GetObject<Ipv4>()->SetAttribute("IpForward", BooleanValue(true));

  // Assign IP addresses
  Ipv4AddressHelper ipv4;

  // Client-Router network (192.168.1.0/24)
  ipv4.SetBase("192.168.1.0", "255.255.255.0");
  Ipv4InterfaceContainer ifClientRouter = ipv4.Assign(devClientRouter);

  // Router-Server network (192.168.2.0/24)
  ipv4.SetBase("192.168.2.0", "255.255.255.0");
  Ipv4InterfaceContainer ifRouterServer = ipv4.Assign(devRouterServer);

  // Configure routing
  Ipv4StaticRoutingHelper routingHelper;

  // Client configuration
  Ptr<Ipv4StaticRouting> clientRouting =
routingHelper.GetStaticRouting(client.Get(0)->GetObject<Ipv4>());
```

```
  // Set default route to router
  clientRouting->AddNetworkRouteTo(Ipv4Address("0.0.0.0"), Ipv4Mask("0.0.0.0"),
ifClientRouter.GetAddress(1), 1);

  // Server configuration
  Ptr<Ipv4StaticRouting> serverRouting =
routingHelper.GetStaticRouting(server.Get(0)->GetObject<Ipv4>());
  // Set default route to router
  serverRouting->AddNetworkRouteTo(Ipv4Address("0.0.0.0"), Ipv4Mask("0.0.0.0"),
ifRouterServer.GetAddress(0), 1);

  // Applications setup
  UdpServerHelper serverHelper(6000);
  ApplicationContainer serverApp = serverHelper.Install(server.Get(0));
  serverApp.Start(Seconds(1.0));
  serverApp.Stop(Seconds(10.0));

  UdpClientHelper clientHelper(ifRouterServer.GetAddress(1), 6000);
  clientHelper.SetAttribute("MaxPackets", UintegerValue(500));
  clientHelper.SetAttribute("Interval", TimeValue(MilliSeconds(20)));
  clientHelper.SetAttribute("PacketSize", UintegerValue(1024));
  ApplicationContainer clientApp = clientHelper.Install(client.Get(0));
  clientApp.Start(Seconds(2.0));
  clientApp.Stop(Seconds(9.0));

  // Enable PCAP tracing
  p2p.EnablePcapAll("inter-subnet");

  // Flow monitor
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll();

  Simulator::Stop(Seconds(10.0));
  Simulator::Run();

  // Analyze results
  monitor->CheckForLostPackets();
  Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
  FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats();

  for (auto &iter : stats)
  {
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(iter.first);
    std::cout << "\nFlow " << iter.first << " (" << t.sourceAddress << " -> " << t.destinationAddress
<< ")\n";
    std::cout << "  Tx Packets: " << iter.second.txPackets << "\n";
    std::cout << "  Rx Packets: " << iter.second.rxPackets << "\n";
    std::cout << "  Lost Packets: " << iter.second.lostPackets << "\n";
    std::cout << "  Throughput: " << iter.second.rxBytes * 8.0 / 9.0 / 1000 << " Kbps\n";
  }

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

Simple Two-Node TCP Bulk Data Transfer Simulation

```
#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/flow-monitor-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("TcpBulkSendExample");

int main() {
    // Enable logging
    LogComponentEnable("BulkSendApplication", LOG_LEVEL_INFO);
    LogComponentEnable("PacketSink", LOG_LEVEL_INFO);

    // Set MSS (TCP SegmentSize)
    Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(1400));

    // Create 2 nodes
    NodeContainer nodes;
    nodes.Create(2);

    // Setup point-to-point link
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    p2p.SetChannelAttribute("Delay", StringValue("2ms"));

    NetDeviceContainer devices = p2p.Install(nodes);
    // Install internet stack
    InternetStackHelper stack;
    stack.Install(nodes);

    // Assign IP addresses
    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = address.Assign(devices);

    // PacketSink on Node 1 (Receiver)
    PacketSinkHelper sinkHelper("ns3::TcpSocketFactory",
                                InetSocketAddress(Ipv4Address::GetAny(), 5001));
    ApplicationContainer sinkApp = sinkHelper.Install(nodes.Get(1));
    sinkApp.Start(Seconds(1.0));
    sinkApp.Stop(Seconds(10.0));

    // BulkSend on Node 0 (Sender)
    BulkSendHelper source("ns3::TcpSocketFactory",
                          InetSocketAddress(interfaces.GetAddress(1), 5001));
    source.SetAttribute("MaxBytes", UintegerValue(0)); // Unlimited
    source.SetAttribute("SendSize", UintegerValue(1400)); // Match MSS if desired

    ApplicationContainer sourceApp = source.Install(nodes.Get(0));
    sourceApp.Start(Seconds(1.0));
    sourceApp.Stop(Seconds(10.0));

    // Enable pcap tracing
    p2p.EnablePcapAll("scratch/q1");

    // FlowMonitor setup
    FlowMonitorHelper flowHelper;
    Ptr<FlowMonitor> monitor = flowHelper.InstallAll();

    // Run simulation
    Simulator::Stop(Seconds(10.0));
    Simulator::Run();

    // FlowMonitor stats
    monitor->CheckForLostPackets();
    Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowHelper.GetClassifier());
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
    for (auto const& flow : stats) {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(flow.first);
        std::cout << "Flow " << flow.first << " (" << t.sourceAddress << " -> " <<
t.destinationAddress << ")\n";
        std::cout << "  Tx Bytes: " << flow.second.txBytes << "\n";
        std::cout << "  Rx Bytes: " << flow.second.rxBytes << "\n";
        std::cout << "  Throughput: "
                  << (flow.second.rxBytes * 8.0 / (flow.second.timeLastRxPacket.GetSeconds() -
flow.second.timeFirstTxPacket.GetSeconds()) / 1e6)
                  << " Mbps\n";
        std::cout << "  Lost Packets: " << flow.second.lostPackets << "\n\n";
    }

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

Basic CSMA LAN Simulation

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("CsmaTopology");

int main() {
    LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
    LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
    LogComponentEnable("UdpServer", LOG_LEVEL_INFO);

    NodeContainer nodes;
    nodes.Create(4);

    CsmaHelper csma;
    csma.SetChannelAttribute("DataRate", StringValue("100Mbps"));
    csma.SetChannelAttribute("Delay", StringValue("1ms"));

    NetDeviceContainer lan1 = csma.Install(nodes);
    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper ipv4;
    ipv4.SetBase("10.1.1.0", "255.255.255.0");
    Ipv4InterfaceContainer interfaces = ipv4.Assign(lan1);

    UdpEchoServerHelper server(5000);
    ApplicationContainer serverApp = server.Install(nodes.Get(3));
    serverApp.Start(Seconds(1.0));
    serverApp.Stop(Seconds(10.0));

    UdpEchoClientHelper client(interfaces.GetAddress(3), 5000);
    client.SetAttribute("MaxPackets", UintegerValue(1000));
    client.SetAttribute("Interval", TimeValue(MilliSeconds(10)));
    client.SetAttribute("PacketSize", UintegerValue(1024));
    ApplicationContainer clientApp = client.Install(nodes.Get(0));
    clientApp.Start(Seconds(1.0));
    clientApp.Stop(Seconds(10.0));

    csma.EnablePcapAll("scratch/q2");

    FlowMonitorHelper flowmonHelper;
    Ptr<FlowMonitor> monitor = flowmonHelper.InstallAll();

    Simulator::Stop(Seconds(10));
    Simulator::Run();

    monitor->CheckForLostPackets();
    Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier>(flowmonHelper.GetClassifier());
    std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();

    for (const auto& flow : stats) {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow(flow.first);
        std::cout << "Flow ID: " << flow.first << " (" << t.sourceAddress << " -> " <<
t.destinationAddress << ")\n";
        std::cout << "  Tx Packets: " << flow.second.txPackets << "\n";
        std::cout << "  Rx Packets: " << flow.second.rxPackets << "\n";
        std::cout << "  Lost Packets: " << flow.second.lostPackets << "\n";
        std::cout << "  Delay Sum: " << flow.second.delaySum.GetSeconds() << " s\n";
        std::cout << "  Throughput: "
```

```
                  << (flow.second.rxBytes * 8.0 / (flow.second.timeLastRxPacket.GetSeconds() - flow.second.timeFirstTxPacket.GetSeconds())) / 1e6
                  << " Mbps\n\n";
    }

    Simulator::Destroy();
}
```