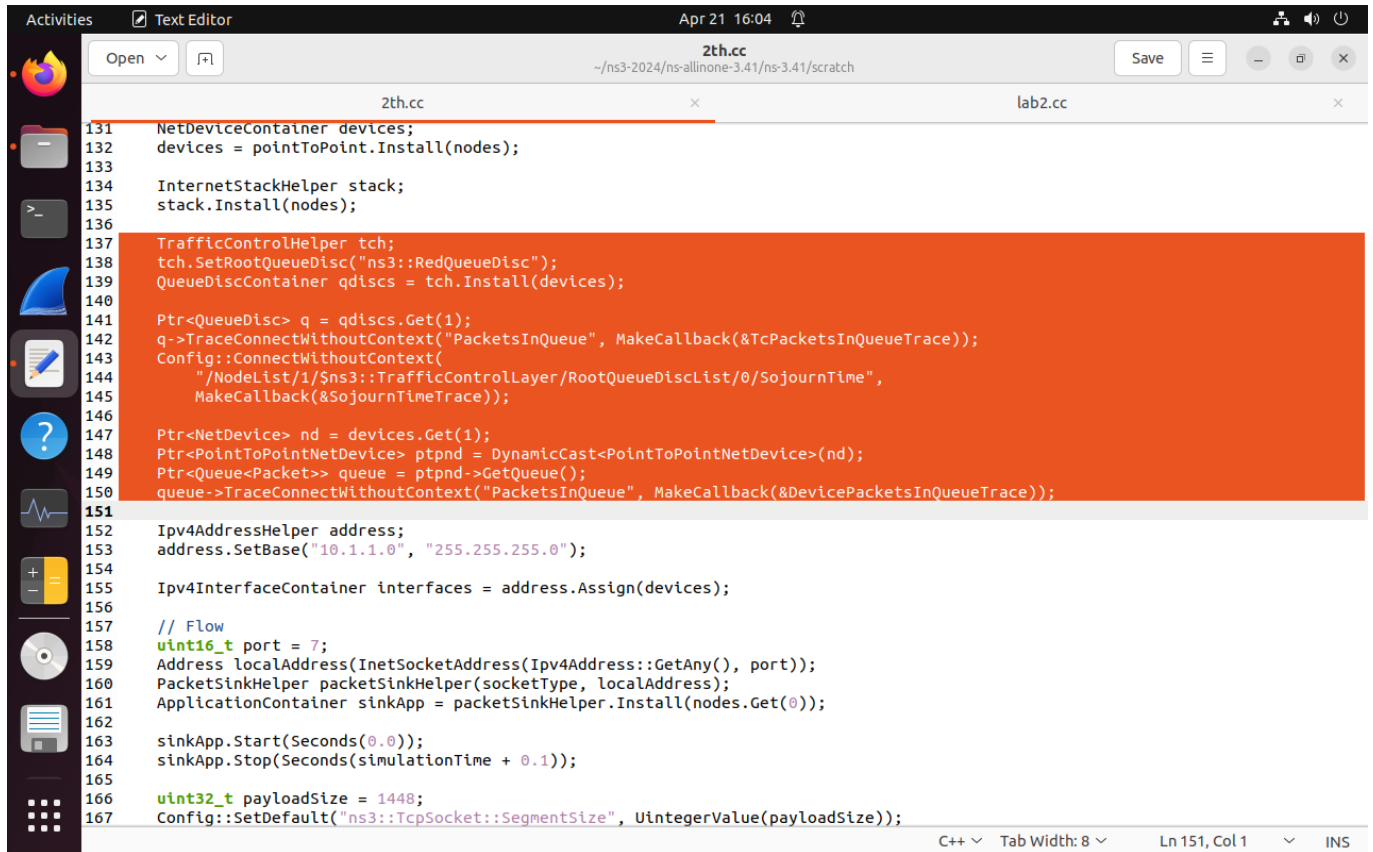
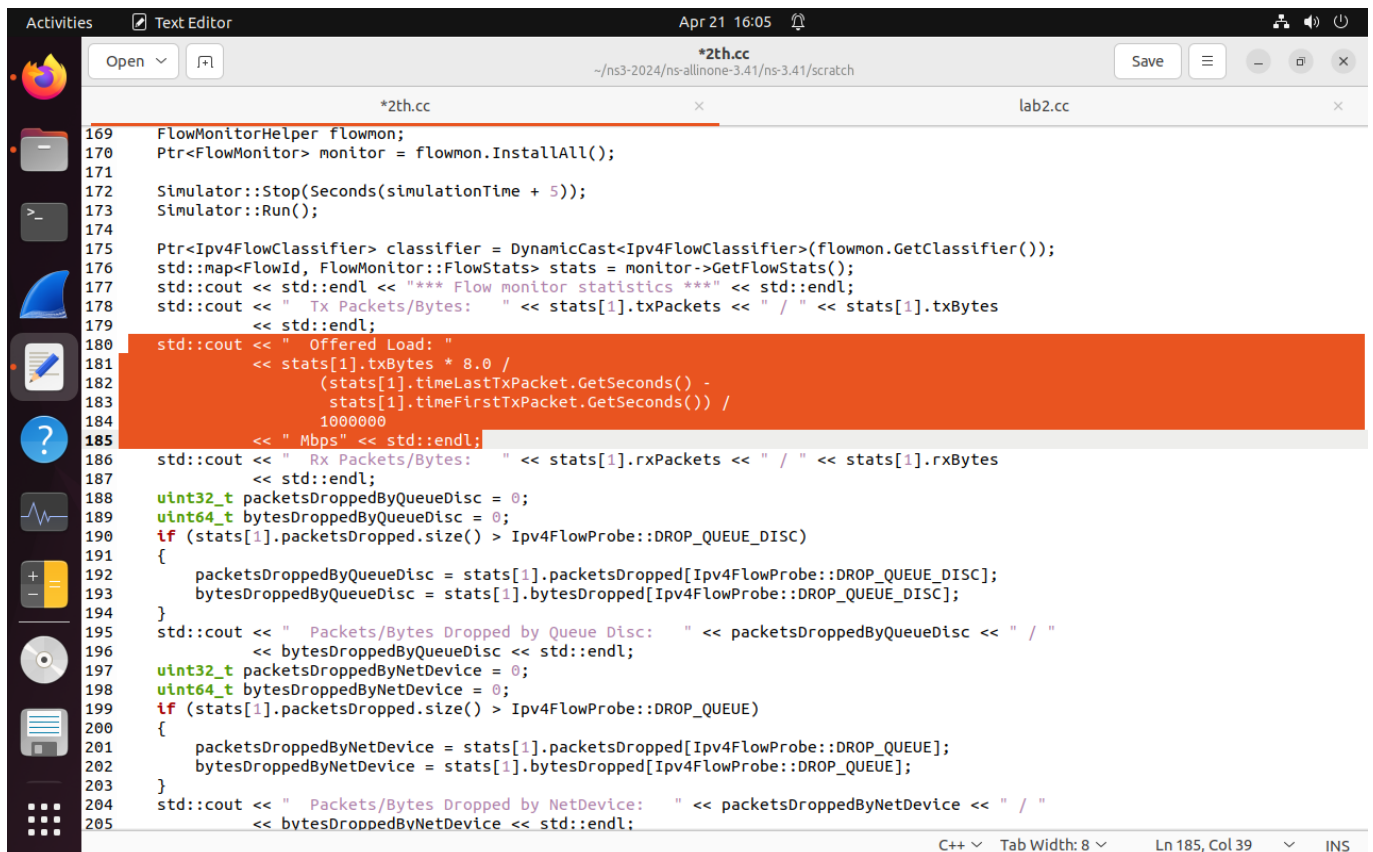


# DELETE CODE



The screenshot shows a C++ code editor with a file named `2th.cc` open. The code is being edited, and a large section of code is highlighted in orange, indicating it is selected for deletion. The code includes network-related setup, traffic control, and packet sink configuration. The editor's status bar at the bottom shows "C++", "Tab Width: 8", "Ln 151, Col 1", and "INS".

```
131 NetDeviceContainer devices;
132 devices = pointToPoint.Install(nodes);
133
134 InternetStackHelper stack;
135 stack.Install(nodes);
136
137 TrafficControlHelper tch;
138 tch.SetRootQueueDisc("ns3::RedQueueDisc");
139 QueueDiscContainer qdiscs = tch.Install(devices);
140
141 Ptr<QueueDisc> q = qdiscs.Get(1);
142 q->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&TcPacketsInQueueTrace));
143 Config::ConnectWithoutContext(
144     "/NodeList/1/$ns3::TrafficControlLayer/RootQueueDiscList/0/SojournTime",
145     MakeCallback(&SojournTimeTrace));
146
147 Ptr<NetDevice> nd = devices.Get(1);
148 Ptr<PointToPointNetDevice> ptpnd = DynamicCast<PointToPointNetDevice>(nd);
149 Ptr<Queue<Packet>> queue = ptpnd->GetQueue();
150 queue->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&DevicePacketsInQueueTrace));
151
152 Ipv4AddressHelper address;
153 address.SetBase("10.1.1.0", "255.255.255.0");
154
155 Ipv4InterfaceContainer interfaces = address.Assign(devices);
156
157 // Flow
158 uint16_t port = 7;
159 Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
160 PacketSinkHelper packetSinkHelper(socketType, localAddress);
161 ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(0));
162
163 sinkApp.Start(Seconds(0.0));
164 sinkApp.Stop(Seconds(simulationTime + 0.1));
165
166 uint32_t payloadSize = 1448;
167 Config::SetDefault("ns3::TcpSocket::SegmentSize", IntegerValue(payloadSize));
```



The screenshot shows a C++ code editor with a file named `*2th.cc` open. The code is being edited, and a large section of code is highlighted in orange, indicating it is selected for deletion. The code includes flow monitor setup, statistics calculation, and packet drop handling. The editor's status bar at the bottom shows "C++", "Tab Width: 8", "Ln 185, Col 39", and "INS".

```
169 FlowMonitorHelper flowmon;
170 Ptr<FlowMonitor> monitor = flowmon.InstallAll();
171
172 Simulator::Stop(Seconds(simulationTime + 5));
173 Simulator::Run();
174
175 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
176 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
177 std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
178 std::cout << " Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
179 << std::endl;
180 std::cout << " Offered Load: "
181 << stats[1].txBytes * 8.0 /
182 (stats[1].timeLastTxPacket.GetSeconds() -
183 stats[1].timeFirstTxPacket.GetSeconds()) /
184 1000000
185 << " Mbps" << std::endl;
186 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
187 << std::endl;
188 uint32_t packetsDroppedByQueueDisc = 0;
189 uint64_t bytesDroppedByQueueDisc = 0;
190 if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
191 {
192     packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
193     bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
194 }
195 std::cout << " Packets/Bytes Dropped by Queue Disc: " << packetsDroppedByQueueDisc << " / "
196 << bytesDroppedByQueueDisc << std::endl;
197 uint32_t packetsDroppedByNetDevice = 0;
198 uint64_t bytesDroppedByNetDevice = 0;
199 if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
200 {
201     packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
202     bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
203 }
204 std::cout << " Packets/Bytes Dropped by NetDevice: " << packetsDroppedByNetDevice << " / "
205 << bytesDroppedByNetDevice << std::endl;
```

Activities Text Editor Apr 21 16:06

Open \*2th.cc Save

~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch

\*2th.cc lab2.cc

```
173 Simulator::Run();
174
175 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
176 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
177 std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
178 std::cout << " Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
179 << std::endl;
180
181 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
182 << std::endl;
183 uint32_t packetsDroppedByQueueDisc = 0;
184 uint64_t bytesDroppedByQueueDisc = 0;
185 if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
186 {
187     packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
188     bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
189 }
190 std::cout << " Packets/Bytes Dropped by Queue Disc: " << packetsDroppedByQueueDisc << " / "
191 << bytesDroppedByQueueDisc << std::endl;
192 uint32_t packetsDroppedByNetDevice = 0;
193 uint64_t bytesDroppedByNetDevice = 0;
194 if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
195 {
196     packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
197     bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
198 }
199 std::cout << " Packets/Bytes Dropped by NetDevice: " << packetsDroppedByNetDevice << " / "
200 << bytesDroppedByNetDevice << std::endl;
201 std::cout << " Throughput: "
202 << stats[1].rxBytes * 8.0 /
203 (stats[1].timeLastRxPacket.GetSeconds() -
204 stats[1].timeFirstRxPacket.GetSeconds()) /
205 1000000
206 << " Mbps" << std::endl;
207 std::cout << " Mean delay: " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
208 << std::endl;
209 std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
210 << std::endl;
211 return 0;
```

C++ Tab Width: 8 Ln 200, Col 55 INS

Activities Text Editor Apr 21 16:07

Open \*2th.cc Save

~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch

\*2th.cc lab2.cc

```
176 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
177 std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
178 std::cout << " Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
179 << std::endl;
180
181 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
182 << std::endl;
183
184 std::cout << " Throughput: "
185 << stats[1].rxBytes * 8.0 /
186 (stats[1].timeLastRxPacket.GetSeconds() -
187 stats[1].timeFirstRxPacket.GetSeconds()) /
188 1000000
189 << " Mbps" << std::endl;
190 std::cout << " Mean delay: " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
191 << std::endl;
192 std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
193 << std::endl;
194 auto dscpVec = classifier->GetDscpCounts(1);
195 for (auto p : dscpVec)
196 {
197     std::cout << " DSCP value: 0x" << std::hex << static_cast<uint32_t>(p.first) << std::dec
198 << " count: " << p.second << std::endl;
199 }
200
201 Simulator::Destroy();
202
203 std::cout << std::endl << "*** Application statistics ***" << std::endl;
204 double thr = 0;
205 uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
206 thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
207 std::cout << " Rx Bytes: " << totalPacketsThr << std::endl;
208 std::cout << " Average Goodput: " << thr << " Mbit/s" << std::endl;
209 std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
210 std::cout << q->GetStats() << std::endl;
211 return 0;
212 }
```

C++ Tab Width: 8 Ln 200, Col 1 INS

ActivitiesText EditorApr 21 16:07~ / ns3-2024/ns-allinone-3.41/ns-3.41/scratch

\*2th.cc

Save

lab2.cc

```
171 Simulator::Stop(Seconds(simulationTime + 5));
172 Simulator::Run();
173
174
175 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
176 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
177 std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
178 std::cout << " Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
179 << std::endl;
180
181 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
182 << std::endl;
183
184 std::cout << " Throughput: "
185 << stats[1].rxBytes * 8.0 /
186 (stats[1].timeLastRxPacket.GetSeconds() -
187 stats[1].timeFirstRxPacket.GetSeconds()) /
188 1000000
189 << " Mbps" << std::endl;
190 std::cout << " Mean delay: " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
191 << std::endl;
192 std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
193 << std::endl;
194 auto dscpVec = classifier->GetDscpCounts(1);
195
196 Simulator::Destroy();
197
198 std::cout << std::endl << "*** Application statistics ***" << std::endl;
199 double thr = 0;
200 uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
201 thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
202 std::cout << " Rx Bytes: " << totalPacketsThr << std::endl;
203 std::cout << " Average Goodput: " << thr << " Mbit/s" << std::endl;
204 std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
205 std::cout << q->GetStats() << std::endl;
206 return 0;
207 }
```

C++ Tab Width: 8 Ln 205, Col 45 INS

# CHANGES IN CODE

```
std::string transportProt = "Udp";
```

```
nodes.Create(4);
```

```
NetDeviceContainer devices01;  
devices01 = pointToPoint.Install(nodes.Get(0),nodes.Get(1));
```

```
NetDeviceContainer devices12;  
devices12 = pointToPoint.Install(nodes.Get(1),nodes.Get(2));
```

```
NetDeviceContainer devices23;  
devices23 = pointToPoint.Install(nodes.Get(2),nodes.Get(3));
```

```
//Ipv4AddressHelper address;  
//address.SetBase("10.1.1.0", "255.255.255.0");  
Ipv4InterfaceContainer interfaces01 = address.Assign(devices01);  
address.SetBase("10.1.2.0", "255.255.255.0");  
Ipv4InterfaceContainer interfaces12 = address.Assign(devices12);  
address.SetBase("10.1.3.0", "255.255.255.0");  
Ipv4InterfaceContainer interfaces23 = address.Assign(devices23);  
Ipv4GlobalRoutingHelper::PopulateRoutingTables(); // type these global routing
```

// Flow ----> rename as UDP flow, copy these same code and paste after apps.stop() and make changes

```
//UDP Flow  
InetSocketAddress rmt(interfaces01.GetAddress(0), port); //change interfaces01  
apps.Add(onoff.Install(nodes.Get(3))); //change .Get(3)
```

```
// TCP Flow  
uint16_t porttcp = 9; //change to 9  
socketType = "ns3::TcpSocketFactory"; //add these line  
Address localAddresstcp(InetSocketAddress(Ipv4Address::GetAny(), porttcp ));  
PacketSinkHelper packetSinkHelpertcp (socketType, localAddresstcp);  
ApplicationContainer sinkApptcp = packetSinkHelpertcp.Install(nodes.Get(0));
```

```
sinkApptcp.Start(Seconds(0.5));  
sinkApptcp.Stop(Seconds(simulationTime + 0.1));
```

## Delete 2 line code from here

```
OnOffHelper onofftcp(socketType, Ipv4Address::GetAny());  
onofftcp.SetAttribute("OnTime",StringValue("ns3::ConstantRandomVariable[Constant=1]"));  
onofftcp.SetAttribute("OffTime",StringValue("ns3::ConstantRandomVariable[Constant=0]"));  
onofftcp.SetAttribute("PacketSize",UIntegerValue(payloadSize));  
onofftcp.SetAttribute("DataRate",StringValue("50Mbps")); // bit/s  
ApplicationContainer appstcp;
```

```

InetSocketAddress rmttcp(interfaces01.GetAddress(0), porttcp);           //change to interfaces01
rmttcp.SetTos(0xb8);
AddressValue remoteAddresstcp(rmttcp);
onofftcp.SetAttribute("Remote", remoteAddresstcp);
appstcp.Add(onofftcp.Install(nodes.Get(2)));           //change to .Get(2)
appstcp.Start(Seconds(1.5));                           // change to Seconds(1.5)
appstcp.Stop(Seconds(simulationTime + 0.1));

```

```

// write these code in the flow monitor statistics section
std::cout << " Dropped Packets/Bytes: " << stats[1].lostPackets
    << " / " << stats[1].rxBytes << std::endl;

```

```

//std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
//    << std::endl;
// auto dscpVec = classifier->GetDscpCounts(1);
//Add these for loop
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iter = stats.begin (); iter != stats.end (); ++iter)
{
    Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
    std::cout << "Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst Addr " <<
t.destinationAddress<< std::endl;
    std::cout << "Tx Packets  = " << iter->second.txPackets<< std::endl;
    std::cout << "Rx Packets  = " << iter->second.rxPackets<< std::endl;
    std::cout << "Lost Packets = " << iter->second.lostPackets<< std::endl;
    std::cout << "Throughput  = " << iter->second.rxBytes * 8.0 / (iter->second.timeLastRxPacket.GetSeconds()-
iter->second.timeFirstTxPacket.GetSeconds()) / 1000000 << " Kbps"<< std::endl;
}

```