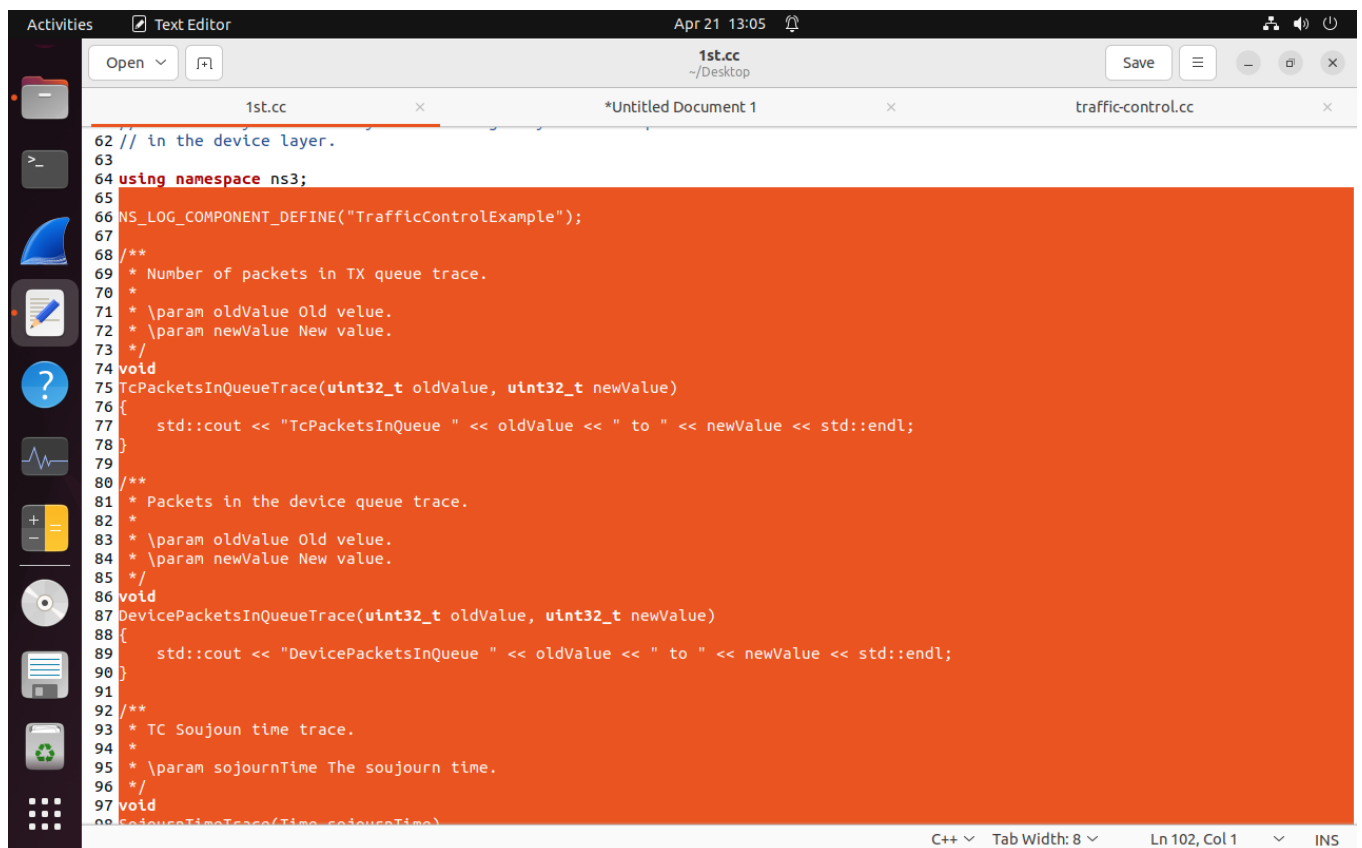
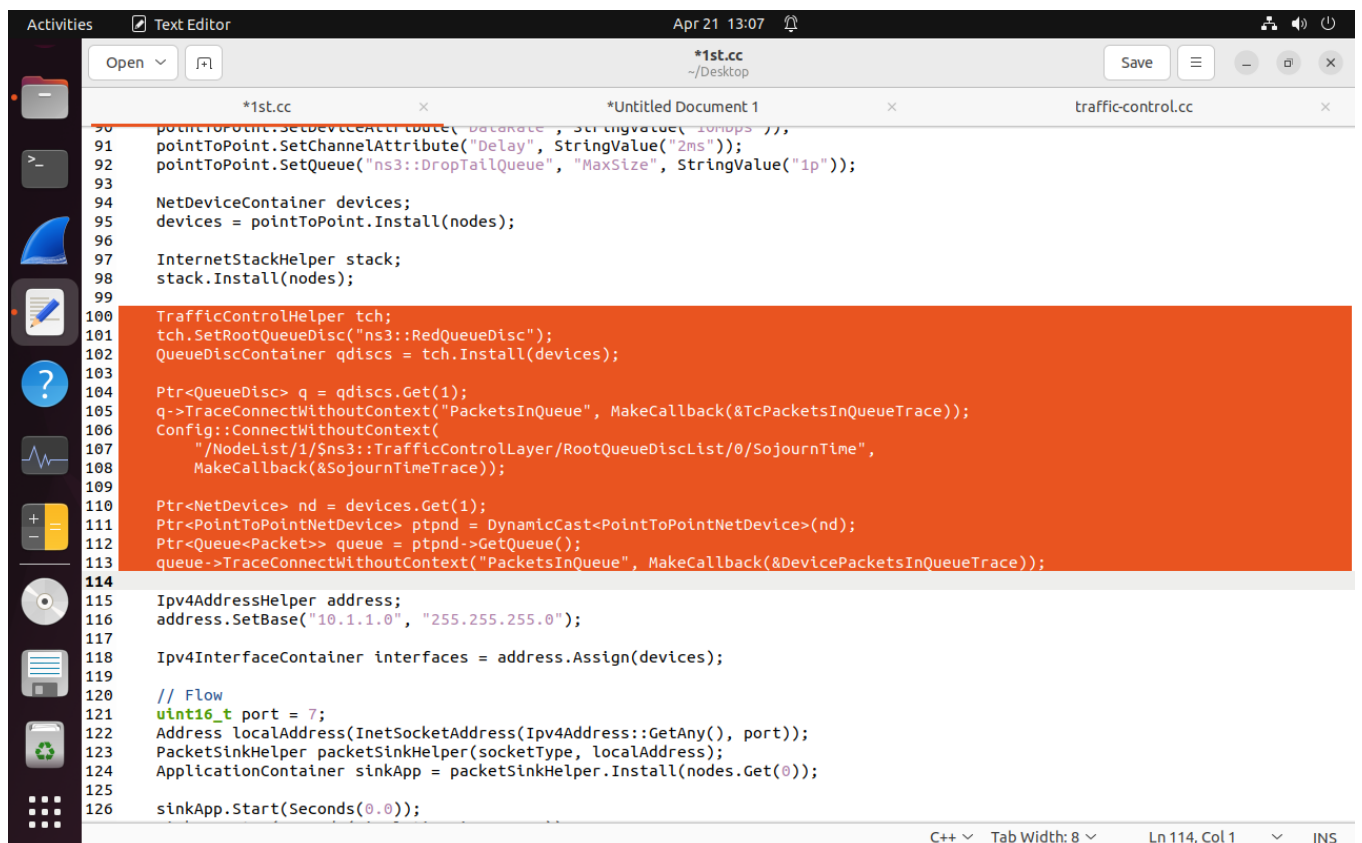


DELETE CODE



The screenshot shows a C++ code editor with a dark theme. The file 1st.cc is open, and a large section of code is highlighted in orange, indicating it is selected for deletion. The code includes namespace declarations, component definitions, and trace functions. The editor's status bar at the bottom shows 'C++', 'Tab Width: 8', 'Ln 102, Col 1', and 'INS'.

```
62 // in the device layer.
63
64 using namespace ns3;
65
66 NS_LOG_COMPONENT_DEFINE("TrafficControlExample");
67
68 /**
69  * Number of packets in TX queue trace.
70  * \param oldValue Old value.
71  * \param newValue New value.
72  */
73 void
74 TcPacketsInQueueTrace(uint32_t oldValue, uint32_t newValue)
75 {
76     std::cout << "TcPacketsInQueue " << oldValue << " to " << newValue << std::endl;
77 }
78
79
80 /**
81  * Packets in the device queue trace.
82  * \param oldValue Old value.
83  * \param newValue New value.
84  */
85 void
86 DevicePacketsInQueueTrace(uint32_t oldValue, uint32_t newValue)
87 {
88     std::cout << "DevicePacketsInQueue " << oldValue << " to " << newValue << std::endl;
89 }
90
91
92 /**
93  * TC Sojourn time trace.
94  * \param sojournTime The sojourn time.
95  */
96 void
97 SojournTimeTrace(Time sojournTime)
98 {
99 }
```



The screenshot shows the same C++ code editor with the file 1st.cc. A different section of code is highlighted in orange, indicating it is selected for deletion. This section includes the setup of network devices, the installation of traffic control helpers, and the configuration of queue disciplines and traces. The editor's status bar at the bottom shows 'C++', 'Tab Width: 8', 'Ln 114, Col 1', and 'INS'.

```
90 pointToPoint.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
91 pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
92 pointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("1p"));
93
94 NetDeviceContainer devices;
95 devices = pointToPoint.Install(nodes);
96
97 InternetStackHelper stack;
98 stack.Install(nodes);
99
100 TrafficControlHelper tch;
101 tch.SetRootQueueDisc("ns3::RedQueueDisc");
102 QueueDiscContainer qdiscs = tch.Install(devices);
103
104 Ptr<QueueDisc> q = qdiscs.Get(1);
105 q->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&TcPacketsInQueueTrace));
106 Config::ConnectWithoutContext(
107     "/NodeList/1/$ns3::TrafficControlLayer/RootQueueDiscList/0/SojournTime",
108     MakeCallback(&SojournTimeTrace));
109
110 Ptr<NetDevice> nd = devices.Get(1);
111 Ptr<PointToPointNetDevice> ptpnd = DynamicCast<PointToPointNetDevice>(nd);
112 Ptr<Queue<Packet>> queue = ptpnd->GetQueue();
113 queue->TraceConnectWithoutContext("PacketsInQueue", MakeCallback(&DevicePacketsInQueueTrace));
114
115 Ipv4AddressHelper address;
116 address.SetBase("10.1.1.0", "255.255.255.0");
117
118 Ipv4InterfaceContainer interfaces = address.Assign(devices);
119
120 // Flow
121 uint16_t port = 7;
122 Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
123 PacketSinkHelper packetSinkHelper(socketType, localAddress);
124 ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(0));
125
126 sinkApp.Start(Seconds(0.0));
```

Activities Text Editor Apr 21 13:08

Open *1st.cc ~/Desktop Save

```
145 std::cout << " Offered Load: "
146 << stats[1].txBytes * 8.0 /
147 (stats[1].timeLastTxPacket.GetSeconds() -
148 stats[1].timeFirstTxPacket.GetSeconds()) /
149 1000000
150 << " Mbps" << std::endl;
151 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
152 << std::endl;
153 uint32_t packetsDroppedByQueueDisc = 0;
154 uint64_t bytesDroppedByQueueDisc = 0;
155 if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE_DISC)
156 {
157     packetsDroppedByQueueDisc = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
158     bytesDroppedByQueueDisc = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE_DISC];
159 }
160 std::cout << " Packets/Bytes Dropped by Queue Disc: " << packetsDroppedByQueueDisc << " / "
161 << bytesDroppedByQueueDisc << std::endl;
162 uint32_t packetsDroppedByNetDevice = 0;
163 uint64_t bytesDroppedByNetDevice = 0;
164 if (stats[1].packetsDropped.size() > Ipv4FlowProbe::DROP_QUEUE)
165 {
166     packetsDroppedByNetDevice = stats[1].packetsDropped[Ipv4FlowProbe::DROP_QUEUE];
167     bytesDroppedByNetDevice = stats[1].bytesDropped[Ipv4FlowProbe::DROP_QUEUE];
168 }
169 std::cout << " Packets/Bytes Dropped by NetDevice: " << packetsDroppedByNetDevice << " / "
170 << bytesDroppedByNetDevice << std::endl;
171 std::cout << " Throughput: "
172 << stats[1].rxBytes * 8.0 /
173 (stats[1].timeLastRxPacket.GetSeconds() -
174 stats[1].timeFirstRxPacket.GetSeconds()) /
175 1000000
176 << " Mbps" << std::endl;
177 std::cout << " Mean delay: " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
178 << std::endl;
179 std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
180 << std::endl;
181 auto dscpVec = classifier->GetDscpCounts(1);
```

C++ Tab Width: 8 Ln 170, Col 55 INS

Activities Text Editor Apr 21 13:08

Open *1st.cc ~/Desktop Save

```
146 << stats[1].txBytes * 8.0 /
147 (stats[1].timeLastTxPacket.GetSeconds() -
148 stats[1].timeFirstTxPacket.GetSeconds()) /
149 1000000
150 << " Mbps" << std::endl;
151 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
152 << std::endl;
153
154 std::cout << " Throughput: "
155 << stats[1].rxBytes * 8.0 /
156 (stats[1].timeLastRxPacket.GetSeconds() -
157 stats[1].timeFirstRxPacket.GetSeconds()) /
158 1000000
159 << " Mbps" << std::endl;
160 std::cout << " Mean delay: " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
161 << std::endl;
162 std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
163 << std::endl;
164 auto dscpVec = classifier->GetDscpCounts(1);
165 for (auto p : dscpVec)
166 {
167     std::cout << " DSCP value: 0x" << std::hex << static_cast<uint32_t>(p.first) << std::dec
168 << " count: " << p.second << std::endl;
169 }
170
171 Simulator::Destroy();
172
173 std::cout << std::endl << "*** Application statistics ***" << std::endl;
174 double thr = 0;
175 uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
176 thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
177 std::cout << " Rx Bytes: " << totalPacketsThr << std::endl;
178 std::cout << " Average Goodput: " << thr << " Mbit/s" << std::endl;
179 std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
180 std::cout << q->GetStats() << std::endl;
181 return 0;
182 }
```

C++ Tab Width: 8 Ln 169, Col 6 INS

Activities Text Editor Apr 21 13:09

Open *1st.cc ~/Desktop Save

*1st.cc *Untitled Document 1 traffic-control.cc

```
140 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(Flowmon.GetClassifier());
141 std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
142 std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
143 std::cout << " Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
144 << std::endl;
145 std::cout << " Offered Load: "
146 << stats[1].txBytes * 8.0 /
147 (stats[1].timeLastTxPacket.GetSeconds() -
148 stats[1].timeFirstTxPacket.GetSeconds()) /
149 1000000
150 << " Mbps" << std::endl;
151 std::cout << " Rx Packets/Bytes: " << stats[1].rxPackets << " / " << stats[1].rxBytes
152 << std::endl;
153
154 std::cout << " Throughput: "
155 << stats[1].rxBytes * 8.0 /
156 (stats[1].timeLastRxPacket.GetSeconds() -
157 stats[1].timeFirstRxPacket.GetSeconds()) /
158 1000000
159 << " Mbps" << std::endl;
160 std::cout << " Mean delay: " << stats[1].delaySum.GetSeconds() / stats[1].rxPackets
161 << std::endl;
162 std::cout << " Mean jitter: " << stats[1].jitterSum.GetSeconds() / (stats[1].rxPackets - 1)
163 << std::endl;
164
165 Simulator::Destroy();
166
167 std::cout << std::endl << "*** Application statistics ***" << std::endl;
168 double thr = 0;
169 uint64_t totalPacketsThr = DynamicCast<PacketSink>(sinkApp.Get(0))->GetTotalRx();
170 thr = totalPacketsThr * 8 / (simulationTime * 1000000.0); // Mbit/s
171 std::cout << " Rx Bytes: " << totalPacketsThr << std::endl;
172 std::cout << " Average Goodput: " << thr << " Mbit/s" << std::endl;
173 std::cout << std::endl << "*** TC Layer statistics ***" << std::endl;
174 std::cout << q->GetStats() << std::endl;
175 return 0;
176 }
```

C++ Tab Width: 8 Ln 174, Col 45 INS

CHANGES IN CODE

```
std::string transportProt = "Udp"; // Changed transport protocol to UDP
```

```
nodes.Create(4); // Modified network topology to include 4 nodes
```

```
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(); //add these line after the following
```

```
//Flow  
ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(3)); // change it to .Get(3)
```

```
InetSocketAddress rmt(interfaces23.GetAddress(1), port); // Set UDP destination address
```

```
apps.Add(onoff.Install(nodes.Get(0))); // Configure UDP traffic
```