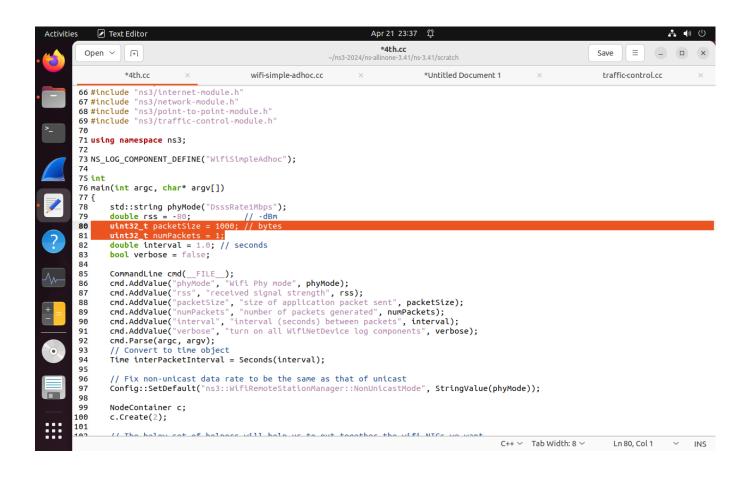
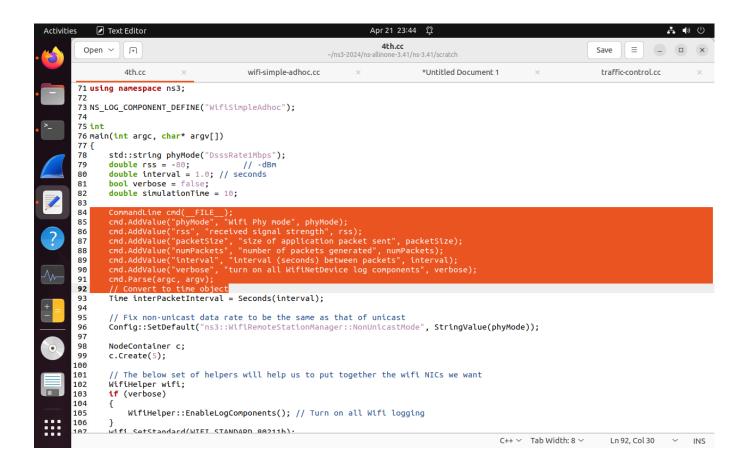
DELETE CODE

Copy these Header files from traffic-contril.cc into wifi-simple-adhoc.cc

```
Text Editor
                                                                                        Apr 21 23:33
                                                                                     traffic-control.cc
  Open V 1
                                                                                                                                                                     Save
                                                  wifi-simple-adhoc.cc
                                                                                                                                                            traffic-control.cc
              4th.cc
                                                                                                       *Untitled Document 1
        DUT WITHOUT ANY WARRANTY: WITHOUT EVEN THE IMPLIED WARRANTY OT
    * MERCHANTABILITY Or FITNESS FOR A PARTICULAR PURPOSE. See the
     * GNU General Public License for more details.
     \mbox{*} You should have received a copy of the GNU General Public License \mbox{*} along with this program; if not, write to the Free Software
13
     * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
15
16
     * Author: Pasquale Imputato p.imputato@gmail.com
     * Author: Stefano Avallone <<u>stefano.avallone@unina.it</u>>
18
20
20
21 #include "ns3/applications-module.h"
22 #include "ns3/core-module.h"
23 #include "ns3/flow-monitor-module.h"
24 #include "ns3/internet-module.h"
25 #include "ns3/network-module.h"
26 #include "ns3/point-to-point-module.h"
27 #include "ns3/traffic-control-module.h"
29 // This simple example shows how to use TrafficControlHelper to install a
30 // QueueDisc on a device.
31 //
32 // The default QueueDisc is a pfifo_fast with a capacity of 1000 packets (as in 33 // Linux). However, in this example, we install a RedQueueDisc with a capacity 34 // of 10000 packets.
35 //
36 // Network topology
38 //
              10.1.1.0
39 // n0 ---
40 //
41 //
           point-to-point
42 // The output will consist of all the traced changes in the length of the RED 43 // internal queue and in the length of the netdevice queue:
45 //
            DevicePacketsInQueue 0 to 1
                                                   C++ × Tab Width: 8 × Ln 21, Col 1 ×
```

```
69 #include "ns3/traffic-control-module.h"
 70
71 using namespace ns3;
 73 NS_LOG_COMPONENT_DEFINE("WifiSimpleAdhoc");
 78
 79
 80
 81
 82
 84
 86
 87
 88
 89
 90
91
      * \param socket The sending socket.
* \param pktSize The packet size.
* \param pktCount The packet count.
* \param pktInterval The interval between two packets.
 93
 95
 97
 98
       enerateTraffic(Ptr<Socket> socket, uint32_t pktSize, uint32_t pktCount, Time pktInterval)
 99
100
           if (pktCount > 0)
101
                socket->Send(Create<Packet>(pktSize));
Simulator::Schedule(pktInterval,
102
103
104
105
                                             pktSizé,
pktCount
106
107
108
109
                                             .
pktInterval):
110
111
```





```
Text Editor
                                                                                          Apr 21 23:38 🐧
                                                                                                                                                                                     → • ∪
                                                                             *4th.cc
~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch
                                                                                                                                                                          \equiv
  Open ~
                1+
               *4th.cc
                                                     wifi-simple-adhoc.cc
                                                                                                          *Untitled Document 1
                                                                                                                                                                 traffic-control.cc
144
           internet.Install(c);
145
          Ipv4AddressHelper ipv4;
NS_LOG_INFO("Assign IP Addresses.");
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i = ipv4.Assign(devices);
146
147
148
150
          TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
Ptr<Socket> recvSink = Socket::CreateSocket(c.Get(0), tid);
InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), 80);
151
152
153
154
155
          Ptr<Socket> source = Socket::CreateSocket(c.Get(1), tid);
InetSocketAddress remote = InetSocketAddress(Ipv4Address("255.255.255.255"), 80);
source->SetAllowBroadcast(true);
source->Connect(remote);
157
159
160
161
          // Tracing
wifiPhy.EnablePcap("wifi-simple-adhoc", devices);
162
164
          // Output what we are doing NS_LOG_UNCOND("Testing " << numPackets << " packets sent with receiver rss " << rss);
166
168
          169
170
171
172
173
175
177
          Simulator::Destroy();
178
179
          return 0;
180 }
                                                                     C++ × Tab Width: 8 × Ln 176, Col 22 ×
```

Copy these Header files from traffic-contril.cc into wifi-simple-adhoc.cc

```
Ipv4InterfaceContainer interfaces = address.Assign(devices);
155
156
157
                 // Flow
uint16_t port = 7;
Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
PacketSinkHelper packetSinkHelper(socketType, localAddress);
ApplicationContainer sinkApp = packetSinkHelper.Install(nodes.Get(0));
159
160
161
162
                 sinkApp.Start(Seconds(0.0));
sinkApp.Stop(Seconds(simulationTime + 0.1));
163
164
165
                 uint32_t payloadSize = 1448;
Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));
166
167
168
                 OnoffHelper onoff(socketType, Ipv4Address::GetAny());
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
169
170
171
172
173
174
175
176
                 InetsocketAddress fmt(interraces.GetAddress(ermt.SetTos(0xb8);
AddressValue remoteAddress(rmt);
onoff.SetAttribute("Remote", remoteAddress);
apps.Add(onoff.Install(nodes.Get(1)));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(simulationTime + 0.1));
177
179
180
181
182
183
                 FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();
184
185
186
187
188
189
                 Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier()); std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats(); std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl; std::cout << " | Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
190
191
192
193
                                               std::endl;
" Offered Load:
194
                  std::cout <<
195
                                         << stats[1].txBytes * 8.0 /
                                                          (stats[1].timeLastTxPacket.GetSeconds() -
    stats[1].timeFirstTxPacket.GetSeconds()) /
197
198
```

CHANGES IN CODE

```
//add these
double simulationTime = 10; // seconds
c.Create(5);
//positionAlloc->Add(Vector(0.0, 0.0, 0.0));
//positionAlloc->Add(Vector(5.0, 0.0, 0.0));
positionAlloc->Add(Vector(5.0, 5.0, 0.0));
positionAlloc->Add(Vector(5.0, 10.0, 0.0));
                                                     // add these lines
positionAlloc->Add(Vector(15.0, 0.0, 0.0));
//copy from traffic-control.cc and paste here
// Flow
  uint16_t port = 7;
  Address localAddress(InetSocketAddress(Ipv4Address::GetAny(), port));
  PacketSinkHelper packetSinkHelper("ns3::UdpSocketFactory", localAddress);
  ApplicationContainer sinkApp = packetSinkHelper.Install(c.Get(4));
  sinkApp.Start(Seconds(0.0));
  sinkApp.Stop(Seconds(simulationTime + 0.1));
  uint32 t payloadSize = 1448;
  Config::SetDefault("ns3::TcpSocket::SegmentSize", UintegerValue(payloadSize));
  OnOffHelper onoff("ns3::UdpSocketFactory", Ipv4Address::GetAny());
  onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
  onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
  onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
  onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
  ApplicationContainer apps;
  InetSocketAddress rmt(i.GetAddress(4), port);
  rmt.SetTos(0xb8);
  AddressValue remoteAddress(rmt);
  onoff.SetAttribute("Remote", remoteAddress);
  apps.Add(onoff.Install(C.Get(1)));
  apps.Start(Seconds(1.0));
  apps.Stop(Seconds(simulationTime + 0.1));
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll();
  Simulator::Stop(Seconds(simulationTime + 5));
  Simulator::Run();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
  std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
  std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
  std::cout << " Tx Packets/Bytes: " << stats[1].txPackets << " / " << stats[1].txBytes
       << std::endl:
```