DELETE CODE

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

Activities Text Editor Apr 21 23:33

Open traffic-control.cc ~/Desktop Save

4th.cc | wifi-simple-adhoc.cc | *Untitled Document 1 | traffic-control.cc

```
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * Author: Pasquale Imputato <p.imputato@gmail.com>
 * Author: Stefano Avallone <stefano.avallone@unina.it>
 */

#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"

// This simple example shows how to use TrafficControlHelper to install a
// QueueDisc on a device.
//
// The default QueueDisc is a pfifo_fast with a capacity of 1000 packets (as in
// Linux). However, in this example, we install a RedQueueDisc with a capacity
// of 10000 packets.
//
// Network topology
//
//       10.1.1.0
// n0 -------------- n1
//    point-to-point
//
// The output will consist of all the traced changes in the length of the RED
// internal queue and in the length of the netdevice queue:
//
//    DevicePacketsInQueue 0 to 1
```

C++ Tab Width: 8 Ln 21, Col 1 INS

```
#include "ns3/traffic-control-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("WifiSimpleAdhoc");

/**
 * Function called when a packet is received.
 *
 * \param socket The receiving socket.
 */
void
ReceivePacket(Ptr<Socket> socket)
{
    while (socket->Recv())
    {
        NS_LOG_UNCOND("Received one packet!");
    }
}

/**
 * Generate traffic.
 *
 * \param socket The sending socket.
 * \param pktSize The packet size.
 * \param pktCount The packet count.
 * \param pktInterval The interval between two packets.
 */
static void
GenerateTraffic(Ptr<Socket> socket, uint32_t pktSize, uint32_t pktCount, Time pktInterval)
{
    if (pktCount > 0)
    {
        socket->Send(Create<Packet>(pktSize));
        Simulator::Schedule(pktInterval,
                            &GenerateTraffic,
                            socket,
                            pktSize,
                            pktCount - 1,
                            pktInterval);
    }
    else
    {
        socket->Close();
    }
```

Activities Text Editor Apr 21 23:37

Open *4th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

*4th.cc wifi-simple-adhoc.cc *Untitled Document 1 traffic-control.cc

```cpp
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("WifiSimpleAdhoc");

int
main(int argc, char* argv[])
{
    std::string phyMode("DsssRate1Mbps");
    double rss = -80;           // -dBm
    uint32_t packetSize = 1000; // bytes
    uint32_t numPackets = 1;
    double interval = 1.0; // seconds
    bool verbose = false;

    CommandLine cmd(__FILE__);
    cmd.AddValue("phyMode", "Wifi Phy mode", phyMode);
    cmd.AddValue("rss", "received signal strength", rss);
    cmd.AddValue("packetSize", "size of application packet sent", packetSize);
    cmd.AddValue("numPackets", "number of packets generated", numPackets);
    cmd.AddValue("interval", "interval (seconds) between packets", interval);
    cmd.AddValue("verbose", "turn on all WifiNetDevice log components", verbose);
    cmd.Parse(argc, argv);
    // Convert to time object
    Time interPacketInterval = Seconds(interval);

    // Fix non-unicast data rate to be the same as that of unicast
    Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue(phyMode));

    NodeContainer c;
    c.Create(2);

    // The below set of helpers will help us to put together the wifi NICs we want
```

C++ Tab Width: 8 Ln 80, Col 1 INS

Activities Text Editor Apr 21 23:44

Open 4th.cc ~/ns3-2024/ns-allinone-3.41/ns-3.41/scratch Save

4th.cc wifi-simple-adhoc.cc *Untitled Document 1 traffic-control.cc

```cpp
using namespace ns3;

NS_LOG_COMPONENT_DEFINE("WifiSimpleAdhoc");

int
main(int argc, char* argv[])
{
    std::string phyMode("DsssRate1Mbps");
    double rss = -80;           // -dBm
    double interval = 1.0; // seconds
    bool verbose = false;
    double simulationTime = 10;

    CommandLine cmd(__FILE__);
    cmd.AddValue("phyMode", "Wifi Phy mode", phyMode);
    cmd.AddValue("rss", "received signal strength", rss);
    cmd.AddValue("packetSize", "size of application packet sent", packetSize);
    cmd.AddValue("numPackets", "number of packets generated", numPackets);
    cmd.AddValue("interval", "interval (seconds) between packets", interval);
    cmd.AddValue("verbose", "turn on all WifiNetDevice log components", verbose);
    cmd.Parse(argc, argv);
    // Convert to time object
    Time interPacketInterval = Seconds(interval);

    // Fix non-unicast data rate to be the same as that of unicast
    Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue(phyMode));

    NodeContainer c;
    c.Create(5);

    // The below set of helpers will help us to put together the wifi NICs we want
    WifiHelper wifi;
    if (verbose)
    {
        WifiHelper::EnableLogComponents(); // Turn on all Wifi logging
    }
    wifi.SetStandard(WIFI_STANDARD_80211b);
```

C++ Tab Width: 8 Ln 92, Col 30 INS

Delete even Simulator::Run()

```
144     internet.Install(c);
145
146     Ipv4AddressHelper ipv4;
147     NS_LOG_INFO("Assign IP Addresses.");
148     ipv4.SetBase("10.1.1.0", "255.255.255.0");
149     Ipv4InterfaceContainer i = ipv4.Assign(devices);
150
151     TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
152     Ptr<Socket> recvSink = Socket::CreateSocket(c.Get(0), tid);
153     InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), 80);
154     recvSink->Bind(local);
155     recvSink->SetRecvCallback(MakeCallback(&ReceivePacket));
156
157     Ptr<Socket> source = Socket::CreateSocket(c.Get(1), tid);
158     InetSocketAddress remote = InetSocketAddress(Ipv4Address("255.255.255.255"), 80);
159     source->SetAllowBroadcast(true);
160     source->Connect(remote);
161
162     // Tracing
163     wifiPhy.EnablePcap("wifi-simple-adhoc", devices);
164
165     // Output what we are doing
166     NS_LOG_UNCOND("Testing " << numPackets << " packets sent with receiver rss " << rss);
167
168     Simulator::ScheduleWithContext(source->GetNode()->GetId(),
169                                    Seconds(1.0),
170                                    &GenerateTraffic,
171                                    source,
172                                    packetSize,
173                                    numPackets,
174                                    interPacketInterval);
175
176     Simulator::Run();
177     Simulator::Destroy();
178
179     return 0;
180 }
```

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

```
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", UintegerValue(payloadSize));
168
169     OnOffHelper onoff(socketType, Ipv4Address::GetAny());
170     onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
171     onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
172     onoff.SetAttribute("PacketSize", UintegerValue(payloadSize));
173     onoff.SetAttribute("DataRate", StringValue("50Mbps")); // bit/s
174     ApplicationContainer apps;
175
176     InetSocketAddress rmt(interfaces.GetAddress(0), port);
177     rmt.SetTos(0xb8);
178     AddressValue remoteAddress(rmt);
179     onoff.SetAttribute("Remote", remoteAddress);
180     apps.Add(onoff.Install(nodes.Get(1)));
181     apps.Start(Seconds(1.0));
182     apps.Stop(Seconds(simulationTime + 0.1));
183
184     FlowMonitorHelper flowmon;
185     Ptr<FlowMonitor> monitor = flowmon.InstallAll();
186
187     Simulator::Stop(Seconds(simulationTime + 5));
188     Simulator::Run();
189
190     Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>(flowmon.GetClassifier());
191     std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();
192     std::cout << std::endl << "*** Flow monitor statistics ***" << std::endl;
193     std::cout << "  Tx Packets/Bytes:   " << stats[1].txPackets << " / " << stats[1].txBytes
194               << std::endl;
195     std::cout << "  Offered Load: "
196               << stats[1].txBytes * 8.0 /
197                      (stats[1].timeLastTxPacket.GetSeconds() -
198                       stats[1].timeFirstTxPacket.GetSeconds()) /
199                      1000000
```

CHANGES IN CODE

```
double simulationTime = 10; // seconds                    //add these

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;
```