#include "ns3/core-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/network-module.h"

#include "ns3/applications-module.h"

#include "ns3/wifi-module.h"

#include "ns3/mobility-module.h"

#include "ns3/csma-module.h"

#include "ns3/internet-module.h"

#include "ns3/flow-monitor-module.h"

// Default Network Topology

//

// Number of wifi or csma nodes can be increased up to 250

// |

// Rank 0 | Rank 1

// -------------------------|----------------------------

// Wifi 10.1.3.0

// AP

// \* \* \* \*

// | | | | 10.1.1.0

// n5 n6 n7 n0 -------------- n1 n2 n3 n4

// point-to-point | | | |

// ================

// LAN 10.1.2.0

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("ThirdScriptExample");

int

main (int argc, char \*argv[])

{

uint32\_t nCsma = 3;

uint32\_t nWifi = 3;

double simulationTime = 10; //seconds

std::string socketType = "ns3::UdpSocketFactory";

CommandLine cmd;

cmd.Parse (argc,argv);

// Check for valid number of csma or wifi nodes

// 250 should be enough, otherwise IP addresses

// soon become an issue

if (nWifi > 250 || nCsma > 250)

{

std::cout << "Too many wifi or csma nodes, no more than 250 each." << std::endl;

return 1;

}

NodeContainer p2pNodes;

p2pNodes.Create (2);

PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;

p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;

csmaNodes.Add (p2pNodes.Get (1));

csmaNodes.Create (nCsma);

CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));

csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;

csmaDevices = csma.Install (csmaNodes);

NodeContainer wifiStaNodes;

wifiStaNodes.Create (nWifi);

NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();

YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();

phy.SetChannel (channel.Create ());

WifiHelper wifi;

wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;

Ssid ssid = Ssid ("ns-3-ssid");

mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;

staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac","Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;

apDevices = wifi.Install (phy, mac, wifiApNode);

MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",

"MinX", DoubleValue (0.0),

"MinY", DoubleValue (0.0),

"DeltaX", DoubleValue (5.0),

"DeltaY", DoubleValue (10.0),

"GridWidth", UintegerValue (3),

"LayoutType", StringValue ("RowFirst"));

mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",

"Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));

mobility.Install (wifiStaNodes);

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

mobility.Install (wifiApNode);

InternetStackHelper stack;

stack.Install (csmaNodes);

stack.Install (wifiApNode);

stack.Install (wifiStaNodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer p2pInterfaces;

p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer csmaInterfaces;

csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");

address.Assign (staDevices);

address.Assign (apDevices);

/\* UdpEchoServerHelper echoServer (9);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);

echoClient.SetAttribute ("MaxPackets", UintegerValue (1));

echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps =

echoClient.Install (wifiStaNodes.Get (nWifi - 1));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

\*/

//Flow

uint16\_t port = 7;

Address localAddress (InetSocketAddress (Ipv4Address::GetAny (), port));

PacketSinkHelper packetSinkHelper (socketType, localAddress);

ApplicationContainer sinkApp = packetSinkHelper.Install (csmaNodes.Get (nCsma));

sinkApp.Start (Seconds (0.0));

sinkApp.Stop (Seconds (simulationTime + 0.1));

uint32\_t payloadSize = 1448;

Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue (payloadSize));

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

ApplicationContainer apps;

AddressValue remoteAddress (InetSocketAddress (csmaInterfaces.GetAddress (nCsma), port));

onoff.SetAttribute ("Remote", remoteAddress);

apps.Add (onoff.Install (wifiStaNodes.Get (nWifi - 1)));

apps.Start (Seconds (1.0));

apps.Stop (Seconds (simulationTime + 0.1));

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

FlowMonitorHelper flowmon;

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

Simulator::Run ();

// Print per flow statistics

monitor->CheckForLostPackets ();

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

for (std::map<FlowId, FlowMonitor::FlowStats>::const\_iterator iter = stats.begin (); iter != stats.end

(); ++iter)

{

Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);

NS\_LOG\_UNCOND("Flow ID: " << iter->first << " Src Addr " << t.sourceAddress << " Dst

Addr " << t.destinationAddress);

NS\_LOG\_UNCOND("Tx Packets = " << iter->second.txPackets);

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;

}

Simulator::Destroy ();

return 0;

}