#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

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

#include "ns3/applications-module.h"

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

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

using namespace ns3;

int main ()

{

double simulationTime = 10; //seconds

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

NodeContainer nodes;

nodes.Create (3);

PointToPointHelper p2p;

p2p.SetDeviceAttribute ("DataRate", StringValue ("10Mbps"));

p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));

p2p.SetQueue ("ns3::DropTailQueue", "MaxSize", StringValue ("1p"));

NetDeviceContainer dev01;

dev01= p2p.Install (nodes.Get(0),nodes.Get(1));

NetDeviceContainer dev12;

dev12= p2p.Install (nodes.Get(1),nodes.Get(2));

InternetStackHelper stack;

stack.Install (nodes);

Ipv4AddressHelper address;

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

Ipv4InterfaceContainer interfaces01 = address.Assign (dev01);

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

Ipv4InterfaceContainer interfaces12 = address.Assign (dev12);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

//Flow

uint16\_t port = 7;

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

PacketSinkHelper psh (socketType, localAddress);

ApplicationContainer sinkApp = psh.Install (nodes.Get (2));

sinkApp.Start (Seconds (0.0));

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

OnOffHelper onoff (socketType, Ipv4Address::GetAny ());

onoff.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

onoff.SetAttribute ("DataRate", StringValue ("50Mbps")); //bit/s

ApplicationContainer apps;

InetSocketAddress rmt (interfaces12.GetAddress (1), port);

AddressValue remoteAddress (rmt);

onoff.SetAttribute ("Remote", remoteAddress);

apps.Add (onoff.Install (nodes.Get (0)));

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;

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;

}

Simulator::Destroy ();

return 0;

}