**#include <fstream>**

**#include <iostream>**

**#include <string>**

**#include <cassert>**

**#include "ns3/core-module.h"**

**#include "ns3/global-route-manager.h"**

**#include "ns3/network-module.h"**

**#include "ns3/internet-module.h"**

**#include "ns3/bridge-module.h"**

**#include "ns3/csma-module.h"**

**#include "ns3/applications-module.h"**

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

using namespace ns3;

**NS\_LOG\_COMPONENT\_DEFINE ("FifthExample");**

**class MyApp : public Application**

**{**

**public:**

**MyApp ();**

**virtual ~MyApp();**

**void Setup (Ptr<Socket> socket, Address address, uint32\_t packetSize, uint32\_t nPackets,**

**DataRate dataRate);**

**private:**

**virtual void StartApplication (void);**

**virtual void StopApplication (void);**

**void ScheduleTx (void);**

**void SendPacket (void);**

**Ptr<Socket>m\_socket;**

**Address m\_peer;**

**uint32\_t m\_packetSize;**

**uint32\_t m\_nPackets;**

**DataRate m\_dataRate;**

**EventId m\_sendEvent;**

**bool m\_running;**

**uint32\_t m\_packetsSent;**

**};**

**MyApp::MyApp () : m\_socket (0),**

**m\_peer (),**

**m\_packetSize (0),**

**m\_nPackets (0),**

**m\_dataRate (0),**

**m\_sendEvent (),**

**m\_running (false),**

**m\_packetsSent (0)**

**{**

**}**

**MyApp::~MyApp()**

**{**

**m\_socket = 0;**

**}**

**void MyApp::Setup (Ptr<Socket> socket, Address address, uint32\_t packetSize, uint32\_t nPackets, DataRate dataRate)**

**{**

**m\_socket = socket;**

**m\_peer = address;**

**m\_packetSize = packetSize;**

**m\_nPackets = nPackets;**

**m\_dataRate = dataRate;**

**}**

**void MyApp::StartApplication (void)**

**{**

**m\_running = true;**

**m\_packetsSent = 0;**

**m\_socket->Bind ();**

**m\_socket->Connect (m\_peer);**

**SendPacket ();**

**}**

**void MyApp::StopApplication (void)**

**{**

**m\_running = false;**

**if (m\_sendEvent.IsRunning ())**

**{**

**Simulator::Cancel (m\_sendEvent);**

**}**

**if (m\_socket)**

**{**

**m\_socket->Close ();**

**}**

**}**

**void MyApp:: SendPacket(void)**

**{**

**Ptr<Packet> packet = Create<Packet> (m\_packetSize);**

**m\_socket->Send (packet);**

**if (++m\_packetsSent<m\_nPackets)**

**{**

**ScheduleTx ();**

**}**

**}**

**void MyApp::ScheduleTx (void)**

**{**

**if (m\_running)**

**{**

**Time tNext (Seconds (m\_packetSize \* 8 / static\_cast<double> (m\_dataRate.GetBitRate())));**

**m\_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);**

**}**

**}**

**static void CwndChange (uint32\_t oldCwnd, uint32\_t newCwnd)**

**{**

**NS\_LOG\_UNCOND (Simulator::Now ().GetSeconds () << "\t" <<newCwnd);**

**}**

**static void RxDrop (Ptr<const Packet> p)**

**{**

**NS\_LOG\_UNCOND ("RxDrop at "<< Simulator::Now ().GetSeconds ());**

**}**

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

{

CommandLine cmd;

cmd.Parse (argc, argv);

/\* Configuration. \*/

/\* Build nodes. \*/

NodeContainer term\_0;

term\_0.Create (1);

NodeContainer term\_1;

term\_1.Create (1);

NodeContainer term\_2;

term\_2.Create (1);

NodeContainer term\_3;

term\_3.Create (1);

NodeContainer term\_4;

term\_4.Create (1);

NodeContainer term\_5;

term\_5.Create (1);

NodeContainer bridge\_0;

bridge\_0.Create (1);

/\* Build link. \*/

CsmaHelper csma\_bridge\_0;

csma\_bridge\_0.SetChannelAttribute ("DataRate", DataRateValue (100000000));

csma\_bridge\_0.SetChannelAttribute ("Delay", **TimeValue (NanoSeconds (6560)));**

/\* Build link net device container. \*/

NodeContainer all\_bridge\_0;

all\_bridge\_0.Add (term\_0);

all\_bridge\_0.Add (term\_1);

all\_bridge\_0.Add (term\_2);

all\_bridge\_0.Add (term\_3);

all\_bridge\_0.Add (term\_4);

all\_bridge\_0.Add (term\_5);

NetDeviceContainer terminalDevices\_bridge\_0;

NetDeviceContainer BridgeDevices\_bridge\_0;

for (int i = 0; i < 6; i++)

{

NetDeviceContainer link = csma\_bridge\_0.Install(NodeContainer(all\_bridge\_0.Get(i), bridge\_0));

terminalDevices\_bridge\_0.Add (link.Get(0));

BridgeDevices\_bridge\_0.Add (link.Get(1));

}

BridgeHelper bridge\_bridge\_0;

bridge\_bridge\_0.Install (bridge\_0.Get(0), BridgeDevices\_bridge\_0);

NetDeviceContainer ndc\_bridge\_0 = terminalDevices\_bridge\_0;

/\* Install the IP stack. \*/

InternetStackHelper internetStackH;

internetStackH.Install (term\_0);

internetStackH.Install (term\_1);

internetStackH.Install (term\_2);

internetStackH.Install (term\_3);

internetStackH.Install (term\_4);

internetStackH.Install (term\_5);

**Ptr<RateErrorModel>em = CreateObject<RateErrorModel> ();**

**em->SetAttribute ("ErrorRate", DoubleValue (0.00001));**

**ndc\_bridge\_0.Get (5)->SetAttribute ("ReceiveErrorModel", PointerValue (em));**

/\* IP assign. \*/

Ipv4AddressHelper ipv4;

ipv4.SetBase ("10.0.0.0", "255.255.255.0");

Ipv4InterfaceContainer iface\_ndc\_bridge\_0 = ipv4.Assign (ndc\_bridge\_0);

/\* Generate Route. \*/

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

/\* Generate Application. \*/

uint16\_t port\_tcp\_0 = 121;

//Address sinkLocalAddress\_tcp\_0 (InetSocketAddress (Ipv4Address::GetAny (), port\_tcp\_0));

**Address sinkLocalAddress\_tcp\_0 (InetSocketAddress (iface\_ndc\_bridge\_0.GetAddress (5), port\_tcp\_0));**

//PacketSinkHelper sinkHelper\_tcp\_0 ("ns3::TcpSocketFactory", sinkLocalAddress\_tcp\_0);

**PacketSinkHelper sinkHelper\_tcp\_0 ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), port\_tcp\_0));**

ApplicationContainer sinkApp\_tcp\_0 = sinkHelper\_tcp\_0.Install (term\_5);

sinkApp\_tcp\_0.Start (Seconds (0.0));

sinkApp\_tcp\_0.Stop (Seconds (5.0));

OnOffHelper clientHelper\_tcp\_0 ("ns3::TcpSocketFactory", Address ());

clientHelper\_tcp\_0.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1]"));

clientHelper\_tcp\_0.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0]"));

ApplicationContainer clientApps\_tcp\_0;

AddressValue remoteAddress\_tcp\_0 (InetSocketAddress (iface\_ndc\_bridge\_0.GetAddress (5), port\_tcp\_0));

clientHelper\_tcp\_0.SetAttribute ("Remote", remoteAddress\_tcp\_0);

clientApps\_tcp\_0.Add (clientHelper\_tcp\_0.Install (term\_1));

clientApps\_tcp\_0.Start (Seconds (0.0));

clientApps\_tcp\_0.Stop (Seconds (5.0));

**Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (term\_1.Get (0),**

**TcpSocketFactory::GetTypeId ());**

**ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback**

**(&CwndChange));**

**Ptr<MyApp> app = CreateObject<MyApp> ();**

**app->Setup (ns3TcpSocket, sinkLocalAddress\_tcp\_0, 1040, 1000, DataRate ("1Mbps"));**

**term\_1.Get (0)->AddApplication (app);**

**app->SetStartTime (Seconds (1.));**

**app->SetStopTime (Seconds (20.));**

**ndc\_bridge\_0.Get (5)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback**

**(&RxDrop));**

/\* Simulation. \*/

/\* Pcap output. \*/

/\* Stop the simulation after x seconds. \*/

uint32\_t stopTime = 6;

Simulator::Stop (Seconds (stopTime));

/\* Start and clean simulation. \*/

Simulator::Run ();

Simulator::Destroy ();

}