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

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

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

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

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

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

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

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

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

using namespace ns3;

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

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 (MilliSeconds (10)));

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

NodeContainer all\_bridge\_0;

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

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

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

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

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

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

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);

/\* 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\_udpEcho\_0 = 123;

UdpEchoServerHelper server\_udpEcho\_0 (port\_udpEcho\_0);

ApplicationContainer apps\_udpEcho\_0 = server\_udpEcho\_0.Install (term\_3.Get(0));

apps\_udpEcho\_0.Start (Seconds (0.0));

apps\_udpEcho\_0.Stop (Seconds (5.0));

Time interPacketInterval\_udpEcho\_0 = Seconds (1.0);

UdpEchoClientHelper client\_udpEcho\_0 (iface\_ndc\_bridge\_0.GetAddress(1), 123);

client\_udpEcho\_0.SetAttribute ("MaxPackets", UintegerValue (1));

client\_udpEcho\_0.SetAttribute ("Interval", TimeValue (interPacketInterval\_udpEcho\_0));

client\_udpEcho\_0.SetAttribute ("PacketSize", UintegerValue (1024));

apps\_udpEcho\_0 = client\_udpEcho\_0.Install (term\_0.Get (0));

apps\_udpEcho\_0.Start (Seconds (0.1));

apps\_udpEcho\_0.Stop (Seconds (5.0));

/\* Simulation. \*/

**FlowMonitorHelper flowmon;**

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

**NS\_LOG\_INFO("RUN SIMULATION");**

**Simulator::Stop (Seconds(9.0));**

**Simulator::Run ();**

**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 i = stats.begin (); i !=**

**stats.end (); ++i)**

**{**

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

**if ((t.sourceAddress=="10.0.0.5" && t.destinationAddress == "10.0.0.2"))**

**{**

**std::cout << "Flow "<< i->first << " (" << t.sourceAddress << " -> " <<**

**t.destinationAddress << ")\n";**

**std::cout <<" Tx Bytes: " << i->second.txBytes << "\n";**

**std::cout <<" Rx Bytes: " << i->second.rxBytes << "\n";**

**std::cout <<" Throughput: " << i->second.rxBytes \* 8.0 / (i->second.timeLastRxPacket.GetSeconds() -i->second.timeFirstTxPacket.GetSeconds())/1024/1024 << " Mbps\n";**

**}**

**}**

**Simulator::Destroy ();**

**return 0;**

**}**