/\* -\*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -\*- \*/

/\*

\* This program is free software; you can redistribute it and/or modify

\* it under the terms of the GNU General Public License version 2 as

\* published by the Free Software Foundation;

\*

\* This program is distributed in the hope that it will be useful,

\* 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

\*/

// Network topology

//

// n0 n1

// | |

// ----------

// | Switch |

// ----------

// | |

// n2 n3

//

//

// - CBR/UDP flows from n0 to n1 and from n3 to n0

// - DropTail queues

// - Tracing of queues and packet receptions to file "openflow-switch.tr"

// - If order of adding nodes and netdevices is kept:

// n0 = 00:00:00;00:00:01, n1 = 00:00:00:00:00:03, n3 = 00:00:00:00:00:07

// and port number corresponds to node number, so port 0 is connected to n0, for example.

#include <iostream>

#include <iostream>

#include <iomanip>

#include <string>

#include <iostream>

#include <sstream>

#include "ns3/gnuplot.h"

#include <fstream>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/csma-module.h"

#include "ns3/internet-module.h"

#include "ns3/applications-module.h"

#include "ns3/openflow-module.h"

#include "ns3/log.h"

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

#include "ns3/netanim-module.h"

#include "ns3/mobility-helper.h"

using namespace ns3;

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

bool verbose = false;

bool use\_drop = false;

ns3::Time timeout = ns3::Seconds (0);

bool SetVerbose (std::string value)

{

verbose = true;

return true;

}

bool SetDrop (std::string value)

{

use\_drop = true;

return true;

}

bool SetTimeout (std::string value)

{

try {

timeout = ns3::Seconds (atof (value.c\_str ()));

return true;

}

catch (...) { return false; }

return false;

}

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

{

#ifdef NS3\_OPENFLOW

//

// Allow the user to override any of the defaults and the above Bind() at

// run-time, via command-line arguments

//

CommandLine cmd;

cmd.AddValue ("v", "Verbose (turns on logging).", MakeCallback (&SetVerbose));

cmd.AddValue ("verbose", "Verbose (turns on logging).", MakeCallback (&SetVerbose));

cmd.AddValue ("d", "Use Drop Controller (Learning if not specified).", MakeCallback (&SetDrop));

cmd.AddValue ("drop", "Use Drop Controller (Learning if not specified).", MakeCallback(&SetDrop));

cmd.AddValue ("t", "Learning Controller Timeout (has no effect if drop controller is specified).", MakeCallback ( &SetTimeout));

cmd.AddValue ("timeout", "Learning Controller Timeout (has no effect if drop controller is specified).", MakeCallback ( &SetTimeout));

cmd.Parse (argc, argv);

if (verbose)

{

LogComponentEnable ("OpenFlowCsmaSwitchExample", LOG\_LEVEL\_INFO);

LogComponentEnable ("OpenFlowInterface", LOG\_LEVEL\_INFO);

LogComponentEnable ("OpenFlowSwitchNetDevice", LOG\_LEVEL\_INFO);

}

//

// Explicitly create the nodes required by the topology (shown above).

//

NS\_LOG\_INFO ("Create nodes.");

NodeContainer terminals;

terminals.Create (4);

NodeContainer csmaSwitch;

csmaSwitch.Create (1);

NS\_LOG\_INFO ("Build Topology");

CsmaHelper csma;

csma.SetChannelAttribute ("DataRate", DataRateValue (5000000));

csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));

// Create the csma links, from each terminal to the switch

NetDeviceContainer terminalDevices;

NetDeviceContainer switchDevices;

for (int i = 0; i < 4; i++)

{

NetDeviceContainer link = csma.Install (NodeContainer (terminals.Get (i), csmaSwitch));

terminalDevices.Add (link.Get (0));

switchDevices.Add (link.Get (1));

}

// Create the switch netdevice, which will do the packet switching

Ptr<Node> switchNode = csmaSwitch.Get (0);

OpenFlowSwitchHelper swtch;

if (use\_drop)

{

Ptr<ns3::ofi::DropController> controller = CreateObject<ns3::ofi::DropController> ();

swtch.Install (switchNode, switchDevices, controller);

}

else

{

Ptr<ns3::ofi::LearningController> controller = CreateObject<ns3::ofi::LearningController> ();

if (!timeout.IsZero ()) controller->SetAttribute ("ExpirationTime", TimeValue (timeout));

swtch.Install (switchNode, switchDevices, controller);

}

// Add internet stack to the terminals

InternetStackHelper internet;

internet.Install (terminals);

// We've got the "hardware" in place. Now we need to add IP addresses.

NS\_LOG\_INFO ("Assign IP Addresses.");

Ipv4AddressHelper ipv4;

ipv4.SetBase ("10.1.1.0", "255.255.255.0");

ipv4.Assign (terminalDevices);

// Create an OnOff application to send UDP datagrams from n0 to n1.

NS\_LOG\_INFO ("Create Applications.");

uint16\_t port = 9; // Discard port (RFC 863)

OnOffHelper onoff ("ns3::UdpSocketFactory",

Address (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)));

onoff.SetConstantRate (DataRate ("500kb/s"));

ApplicationContainer app = onoff.Install (terminals.Get (1));

// Start the application

app.Start (Seconds (1.0));

app.Stop (Seconds (10.0));

// Create an optional packet sink to receive these packets

PacketSinkHelper sink ("ns3::UdpSocketFactory",

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

app = sink.Install (terminals.Get (0));

app.Start (Seconds (0.0));

//

// Create a similar flow from n3 to n0, starting at time 1.0 seconds

//

onoff.SetAttribute ("Remote",

AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.2"), port)));

app = onoff.Install (terminals.Get (2));

app.Start (Seconds (1.0));

app.Stop (Seconds (10.0));

app = sink.Install (terminals.Get (0));

app.Start (Seconds (0.0));

//

// Create a similar flow from n3 to n0, starting at time 1.0 seconds

//

onoff.SetAttribute ("Remote",

AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.3"), port)));

app = onoff.Install (terminals.Get (3));

app.Start (Seconds (1.0));

app.Stop (Seconds (10.0));

app = sink.Install (terminals.Get (0));

app.Start (Seconds (0.0));

//Gnuplot parameters  
  
 std::string fileNameWithNoExtension = "FlowVSThroughput3to1";

std::string graphicsFileName = fileNameWithNoExtension + ".png";

std::string plotFileName = fileNameWithNoExtension + ".plt";

std::string plotTitle = "Flow vs Throughput";

std::string dataTitle = "Throughput";

// Instantiate the plot and set its title.

Gnuplot gnuplot (graphicsFileName);

gnuplot.SetTitle (plotTitle);

// Make the graphics file, which the plot file will be when it

// is used with Gnuplot, be a PNG file.

gnuplot.SetTerminal ("png");

// Set the labels for each axis.

gnuplot.SetLegend ("Flow", "Throughput");

Gnuplot2dDataset dataset;

dataset.SetTitle (dataTitle);

dataset.SetStyle (Gnuplot2dDataset::LINES\_POINTS);

NS\_LOG\_INFO ("FlowMonitor.");

FlowMonitorHelper flowmon;

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

//

// Now, do the actual simulation.

//

MobilityHelper mobility;

// setup the grid itself: objects are layed out

// started from (-100,-100) with 20 objects per row,

// the x interval between each object is 5 meters

// and the y interval between each object is 20 meters

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",

"MinX", DoubleValue(10.0),

"MinY", DoubleValue(10.0),

"DeltaX", DoubleValue(5.0),

"DeltaY", DoubleValue(20.0),

"GridWidth", UintegerValue(20),

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

mobility.[SetMobilityModel](https://www.nsnam.org/doxygen/classns3_1_1_mobility_helper.html#a030275011b6f40682e70534d30280aba) ("ns3::ConstantPositionMobilityModel");

mobility.[Install](https://www.nsnam.org/doxygen/classns3_1_1_mobility_helper.html#a07737960ee95c0777109cf2994dd97ae) (terminals);

NS\_LOG\_INFO ("Run Simulation.");

Simulator::Stop (Seconds(11.0));

AnimationInterface anim ("animation3to1.xml");

Simulator::Run ();

monitor->CheckForLostPackets ();

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

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

double Throughput=0.0;

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

{

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

std::cout << "Flow " << i->first << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";

std::cout << " Tx Packets = " << i->second.txPackets<< "\n";

std::cout << " Rx Packets = " << i->second.rxPackets<< "\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";

 std::cout << "  Lost Packets= " << i->second.lostPackets << "\n";

 std::cout << "  Delay Sum= " << i->second.delaySum << " \n";

std::cout << "  Jitter Sum= " << i->second.jitterSum << " \n";

Throughput= i->second.rxBytes \* 8.0 / (i->second.timeLastRxPacket.GetSeconds() - i->second.timeFirstTxPacket.GetSeconds())/1024/1024;

dataset.Add((double)i->first,(double) Throughput);

}

//Gnuplot ...continued  
   
    gnuplot.AddDataset (dataset);

// Open the plot file.

std::ofstream plotFile (plotFileName.c\_str());

// Write the plot file.

gnuplot.GenerateOutput (plotFile);

// Close the plot file.

plotFile.close ();

monitor->SerializeToXmlFile("openflow3to1.flowmon", true, true);

Simulator::Destroy ();

NS\_LOG\_INFO ("Done.");

#else

NS\_LOG\_INFO ("NS-3 OpenFlow is not enabled. Cannot run simulation.");

#endif // NS3\_OPENFLOW

}