Hybrid Topology

```
/* -*- 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
 */

#include <fstream>
#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/netanim-module.h"
#include "ns3/mobility-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FifthScriptExample");

//
=======================================================================
====
//
//         node 0                 node 1
//   +----------------+    +----------------+
//   |    ns-3 TCP    |    |    ns-3 TCP    |
//   +----------------+    +----------------+
//   |    10.1.1.1    |    |    10.1.1.2    |
//   +----------------+    +----------------+
//   | point-to-point |    | point-to-point |
//   +----------------+    +----------------+
//           |                     |
//           +---------------------+
//                5 Mbps, 2 ms
//
//
// We want to look at changes in the ns-3 TCP congestion window.  We need
// to crank up a flow and hook the CongestionWindow attribute on the socket
// of the sender.  Normally one would use an on-off application to generate a
// flow, but this has a couple of problems.  First, the socket of the on-off
// application is not created until Application Start time, so we wouldn't be
// able to hook the socket (now) at configuration time.  Second, even if we
// could arrange a call after start time, the socket is not public so we
// couldn't get at it.
//
// So, we can cook up a simple version of the on-off application that does what
// we want.  On the plus side we don't need all of the complexity of the on-off
// application.  On the minus side, we don't have a helper, so we have to get
// a little more involved in the details, but this is trivial.
//
// So first, we create a socket and do the trace connect on it; then we pass
// this socket into the constructor of our simple application which we then
// install in the source node.
//
=======================================================================
====
//
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 (__FILE__);
  cmd.Parse (argc, argv);

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
  em->SetAttribute ("ErrorRate", DoubleValue (0.00001));
  devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));

  InternetStackHelper stack;
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.252");
  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  uint16_t sinkPort = 8080;
  Address sinkAddress (InetSocketAddress (interfaces.GetAddress (1), sinkPort));
  PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress
(Ipv4Address::GetAny (), sinkPort));
  ApplicationContainer sinkApps = packetSinkHelper.Install (nodes.Get (1));
  sinkApps.Start (Seconds (0.));
  sinkApps.Stop (Seconds (20.));

  Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0), TcpSocketFactory::GetTypeId
());
  ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback
(&CwndChange));

  Ptr<MyApp> app = CreateObject<MyApp> ();
  app->Setup (ns3TcpSocket, sinkAddress, 1040, 1000, DataRate ("1Mbps"));
  nodes.Get (0)->AddApplication (app);
  app->SetStartTime (Seconds (1.));
  app->SetStopTime (Seconds (20.));

  devices.Get (1)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback (&RxDrop));

MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(nodes);
AnimationInterface anim("fifth.xml");
AnimationInterface::SetConstantPosition(nodes.Get(0),10,25);
AnimationInterface::SetConstantPosition(nodes.Get(1),30,50);
anim.EnablePacketMetadata(true);
pointToPoint.EnablePcapAll("fifth");

  Simulator::Stop (Seconds (20));
  Simulator::Run ();
  Simulator::Destroy ();

  return 0;
}
```