Networks Assignment 5

Network Simulation Using NS-3

Prepared by:

Nikhil Agarwal
11012323

Himanshu Upreti
11012315

Rohit Rangan
11012333

Instructors:

Dr. Sukumar Nandi

T. Venkatesh

April 15, 2014

1. NS-3 (www.nsnam.org) is a discrete event, packet level network simulator for Internet systems. Download and install NS-3. Create a simple topology of two nodes – Node1 and Node2, separated by a point-to-point link. Setup a UdpClient on Node1 and UdpServer on Node2. Start the client application, and measure end to end throughput while varying the latency of the link. Now add another client application to Node1 and a server instance to Node2. What you need to configure to ensure that there is no conflict? Measure end-to-end throughput with the extra client and server application instances. Show screenshots of pcap traces which indicate that delivery is made to the appropriate server instance.

```
/* -*- 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/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Question1");

int main (int argc, char *argv[])
{
  double lat = 2.0;
  uint64_t rate = 5000000; // Data rate in bps
  double interval = 0.05;

  CommandLine cmd;
  cmd.AddValue ("latency", "P2P link Latency in miliseconds", lat);
  cmd.AddValue ("rate", "P2P data rate in bps", rate);
  cmd.AddValue ("interval", "UDP client packet interval", interval);

  cmd.Parse (argc, argv);
```

```
//
// Enable logging for UdpClient and UdpServer
//
  //LogComponentEnable ("UdpClient", LOG_LEVEL_INFO);
  //LogComponentEnable ("UdpServer", LOG_LEVEL_INFO);

//
// Explicitly create the nodes required by the topology (shown above).
//
  NS_LOG_INFO ("Create nodes.");
  NodeContainer n;
  n.Create (2);

  NS_LOG_INFO ("Create channels.");
//
// Explicitly create the channels required by the topology (shown above).
//
  PointToPointHelper p2p;
  p2p.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (lat)));
  p2p.SetDeviceAttribute ("DataRate", DataRateValue (DataRate (rate)));
  p2p.SetDeviceAttribute ("Mtu", UintegerValue (1400));

  NetDeviceContainer dev;
  dev = p2p.Install (n);

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

//
// Install Internet Stack
//
  InternetStackHelper internet;
  internet.Install (n);
  Ipv4AddressHelper ipv4;

  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (dev);

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  NS_LOG_INFO ("Create Applications.");
//
// Create one udpServer application on node one.
//
  uint16_t port1 = 8000; // Need different port numbers to ensure there is
      no conflict
  uint16_t port2 = 8001;
  UdpServerHelper server1 (port1);
  UdpServerHelper server2 (port2);
  ApplicationContainer servapps;
  servapps = server1.Install (n.Get (1));
  //apps = server2.Install (n.Get (1));
```

```
  servapps.Start (Seconds (1.0));
  servapps.Stop (Seconds (10.0));

//
// Create one UdpClient application to send UDP datagrams from node zero
   to
// node one.
//
  uint32_t MaxPacketSize = 1024;
  Time interPacketInterval = Seconds (interval);
  uint32_t maxPacketCount = 320;
  ApplicationContainer cliapps;
  UdpClientHelper client1 (i.GetAddress (1), port1);
  //UdpClientHelper client2 (i2.GetAddress (1), port2);

  client1.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  client1.SetAttribute ("Interval", TimeValue (interPacketInterval));
  client1.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));

  //client2.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  //client2.SetAttribute ("Interval", TimeValue (interPacketInterval));
  //client2.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));

  cliapps = client1.Install (n.Get (0));
  //apps = client2.Install (n.Get (0));

  cliapps.Start (Seconds (2.0));
  cliapps.Stop (Seconds (10.0));

//
// Tracing
//
  AsciiTraceHelper ascii;
  p2p.EnableAscii(ascii.CreateFileStream ("Question1.tr"), dev);
  p2p.EnablePcap("Question1", dev, false);

//
// Calculate Throughput using Flowmonitor
//
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll ();

//
// Now, do the actual simulation.
//
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds(11.0));
  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=="10.1.1.1" && t.destinationAddress == "
         10.1.1.2"))
      {
          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";
          double thpt = i->second.rxBytes * 8.0 / (i->second.
             timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.
             GetSeconds ());
          thpt /= (1024 * 1024);
          std::cout << "  Throughput: " << thpt  << " Mbps\n";
      }
     }

  monitor->SerializeToXmlFile("Question1.flowmon", true, true);

  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

Listing 1: Question 1, without extra client and server instances

```
/* -*- 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/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
```

```
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Question1");

int main (int argc, char *argv[])
{
  double lat = 2.0;
  uint64_t rate = 5000000; // Data rate in bps
  double interval = 0.05;

  CommandLine cmd;
  cmd.AddValue ("latency", "P2P link Latency in miliseconds", lat);
  cmd.AddValue ("rate", "P2P data rate in bps", rate);
  cmd.AddValue ("interval", "UDP client packet interval", interval);

  cmd.Parse (argc, argv);

//
// Enable logging for UdpClient and UdpServer
//
  //LogComponentEnable ("UdpClient", LOG_LEVEL_INFO);
  //LogComponentEnable ("UdpServer", LOG_LEVEL_INFO);

//
// Explicitly create the nodes required by the topology (shown above).
//
  NS_LOG_INFO ("Create nodes.");
  NodeContainer n;
  n.Create (2);

  NS_LOG_INFO ("Create channels.");
//
// Explicitly create the channels required by the topology (shown above).
//
  PointToPointHelper p2p;
  p2p.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (lat)));
  p2p.SetDeviceAttribute ("DataRate", DataRateValue (DataRate (rate)));
  p2p.SetDeviceAttribute ("Mtu", UintegerValue (1400));

  NetDeviceContainer dev;
  dev = p2p.Install (n);

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

//
// Install Internet Stack
//
  InternetStackHelper internet;
  internet.Install (n);
  Ipv4AddressHelper ipv4;

  NS_LOG_INFO ("Assign IP Addresses.");
```

```
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (dev);

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  NS_LOG_INFO ("Create Applications.");
//
// Create one udpServer application on node one.
//
  uint16_t port1 = 8000; // Need different port numbers to ensure there is
      no conflict
  uint16_t port2 = 8001;
  UdpServerHelper server1 (port1);
  UdpServerHelper server2 (port2);
  ApplicationContainer servapps;
  servapps = server1.Install (n.Get (1));
  servapps = server2.Install (n.Get (1));

  servapps.Start (Seconds (1.0));
  servapps.Stop (Seconds (10.0));

//
// Create one UdpClient application to send UDP datagrams from node zero
    to
// node one.
//
  uint32_t MaxPacketSize = 1024;
  Time interPacketInterval = Seconds (interval);
  uint32_t maxPacketCount = 320;
  ApplicationContainer cliapps;
  UdpClientHelper client1 (i.GetAddress (1), port1);
  UdpClientHelper client2 (i.GetAddress (1), port2);

  client1.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  client1.SetAttribute ("Interval", TimeValue (interPacketInterval));
  client1.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));

  client2.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
  client2.SetAttribute ("Interval", TimeValue (interPacketInterval));
  client2.SetAttribute ("PacketSize", UintegerValue (MaxPacketSize));

  cliapps = client1.Install (n.Get (0));
  cliapps = client2.Install (n.Get (0));

  cliapps.Start (Seconds (2.0));
  cliapps.Stop (Seconds (10.0));

//
// Tracing
//
  AsciiTraceHelper ascii;
  p2p.EnableAscii(ascii.CreateFileStream ("Question1-Extra.tr"), dev);
  p2p.EnablePcap("Question1-Extra", dev, false);
```

```
//
// Calculate Throughput using Flowmonitor
//
  FlowMonitorHelper flowmon;
  Ptr<FlowMonitor> monitor = flowmon.InstallAll ();

//
// Now, do the actual simulation.
//
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds(11.0));
  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=="10.1.1.1" && t.destinationAddress == "
         10.1.1.2"))
      {
          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";
          double thpt = i->second.rxBytes * 8.0 / (i->second.
             timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.
             GetSeconds ());
          thpt /= (1024 * 1024);
          std::cout << "  Throughput: " << thpt  << " Mbps\n";
      }
     }

  monitor->SerializeToXmlFile("Question1-Extra.flowmon", true, true);

  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
}
```

Listing 2: Question 1, with extra client and server instances

Latency	Throughput
2ms	4.25Mbps
4ms	3.84Mbps
8ms	3.22Mbps
16ms	1.64Mbps
32ms	1.64Mbps
2 server processes, 2ms	4.25

We can see that if the latency is increased the throughput decreases.
While running two instances of server on node 1, we need to make sure the two instances run on different ports.

2. Make the following simulations in NS-3

Make a topology of 4 nodes in the following way.

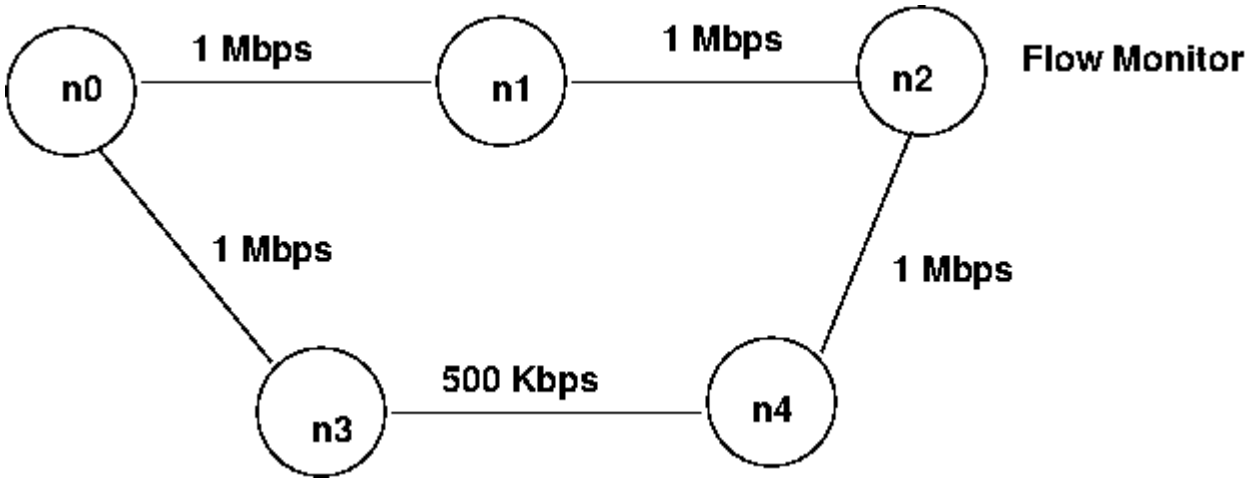


Figure 1: Topology of four nodes

n0 starts CBR traffic at time 1.0 of rate 900 Kbps destined for n2. n0 starts another CBR traffic at time 1.5 of rate 300 Kbps destined for node n1. At time 2.0, link from n0 to n1 goes down. Use a dynamic routing protocol so that path n0-n3-n2 is used now At time 2.7, link n0-n1 comes up again. At time 3.0, CBR traffic destined for node n1 stops. CBR destined for n2 stops at time 3.5. Use a Flow monitor to monitor losses at n2. Draw a graph of percentage loss as a function of time for the duration of simulation. Give an explanation for results you find.

```
/* -*- 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
 */

/*
    LAB Assignment #2
    1. n0 starts CBR traffic at time 1.0 of rate 900 Kbps destined for n2.
```

```
    2. n0 starts another CBR traffic at time 1.5 of rate 300 Kbps destined
       for node n1.

    3. At time 2.0, link from n0 to n1 goes down. Use a dynamic routing
       protocol so that path n0-n3-n2 is used now.

    4. At time 2.7, link n0-n1 comes up again.

    5. At time 3.0, CBR traffic destined for node n1 stops.

    6. CBR destined for n2 stops at time 3.5.

    7. Use a  Flow monitor to monitor losses at n2.

    8. Draw a graph of percentage loss as a function of time for the
       duration of simulation.

    9. Give an explanation for results you find.
*/

// Network topology
//
//       1Mbps          1Mbps
//        n0 ------------ n1 --------- n2 Flow Monitor
//       \                 /
//        \ 1Mbps            / 1Mbps
//         \           /
//          n3 ------------- n4
//               500Kbps
//

#include <iostream>
#include <fstream>
#include <string>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/ipv4-global-routing-helper.h"

using namespace std;
using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("DynamicRoutingProtocol");

ofstream fp;
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor;

uint32_t txPacketsum = 1;
uint32_t rxPacketsum = 1;
```

```
uint32_t DropPacketsum = 0;
uint32_t LostPacketsum = 0;
double Delaysum = 0;

void calculateLoss () {
  double total=0;

  monitor->CheckForLostPackets ();
  Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (
     flowmon.GetClassifier ());
  map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

  map<FlowId, FlowMonitor::FlowStats>::const_iterator i;
  double j=0,m=0;
  for ( i = stats.begin (); i != stats.end (); ++i)
  {
     Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
      if(t.destinationAddress == "10.1.3.2")
      {
        j += i->second.txPackets;
        m += i->second.lostPackets;
        std::cout<<">>"<<i->second.txPackets<<std::endl;
      }
  }
  total = m/j ;
  fp << Simulator::Now ().GetSeconds () <<" "<< total << endl;
}

int main (int argc, char *argv[])
{
  string latency = "2ms";
  string rate = "1Mbps"; // for all P2P links except n3-n4
  string n3_n4rate = "500Kbps"; // n3n4 P2P link
  bool enableFlowMonitor = true;

  fp.open ("Question2-loss.txt");
  if (!fp) {
    cout << "Cannot open the output file" << endl;
    exit (1);
  }

  // The below value configures the default behavior of global routing.
  // By default, it is disabled.  To respond to interface events, set to
     true
  Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents",
     BooleanValue (true));

  CommandLine cmd;
  cmd.AddValue ("latency", "P2P link Latency in miliseconds", latency);
  cmd.AddValue ("rate", "P2P data rate in bps", rate);
  cmd.AddValue ("n3_n4rate", "P2P data rate for n3-n4 link in bps",
     n3_n4rate);
  cmd.AddValue ("EnableMonitor", "Enable Flow Monitor", enableFlowMonitor)
     ;
```

```
cmd.Parse (argc, argv);

/* Explicitly create the nodes required by the topology (shown above).
   */
NS_LOG_INFO ("Create nodes.");
NodeContainer nodes; // ALL Nodes
nodes.Create(5);

NodeContainer n0n3 = NodeContainer (nodes.Get (0), nodes.Get (3));
NodeContainer n0n1 = NodeContainer (nodes.Get (0), nodes.Get (1));
NodeContainer n1n2 = NodeContainer (nodes.Get (1), nodes.Get (2));
NodeContainer n2n4 = NodeContainer (nodes.Get (2), nodes.Get (4));
NodeContainer n3n4 = NodeContainer (nodes.Get (3), nodes.Get (4));

/* Install Internet Stack */
InternetStackHelper stack;
stack.Install (nodes);

/* We create the channels first without any IP addressing information */
NS_LOG_INFO ("Create channels.");
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue (rate));
p2p.SetChannelAttribute ("Delay", StringValue (latency));
//p2p.SetDeviceAttribute ("DataRate", StringValue ("0.1kbps"));
NetDeviceContainer d0d3 = p2p.Install (n0n3);
//p2p.SetDeviceAttribute ("DataRate", StringValue (rate));
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d1d2 = p2p.Install (n1n2);
NetDeviceContainer d2d4 = p2p.Install (n2n4);

p2p.SetDeviceAttribute ("DataRate", StringValue (n3_n4rate));
NetDeviceContainer d3d4 = p2p.Install (n3n4);

/* Add IP addresses. */
NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i0i3 = ipv4.Assign (d0d3);

ipv4.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);

ipv4.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer i1i2 = ipv4.Assign (d1d2);

ipv4.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer i2i4 = ipv4.Assign (d2d4);

ipv4.SetBase ("10.1.5.0", "255.255.255.0");
Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);

NS_LOG_INFO ("Enable static global routing.");
/* Turn on global static routing so we can actually be routed across the
```

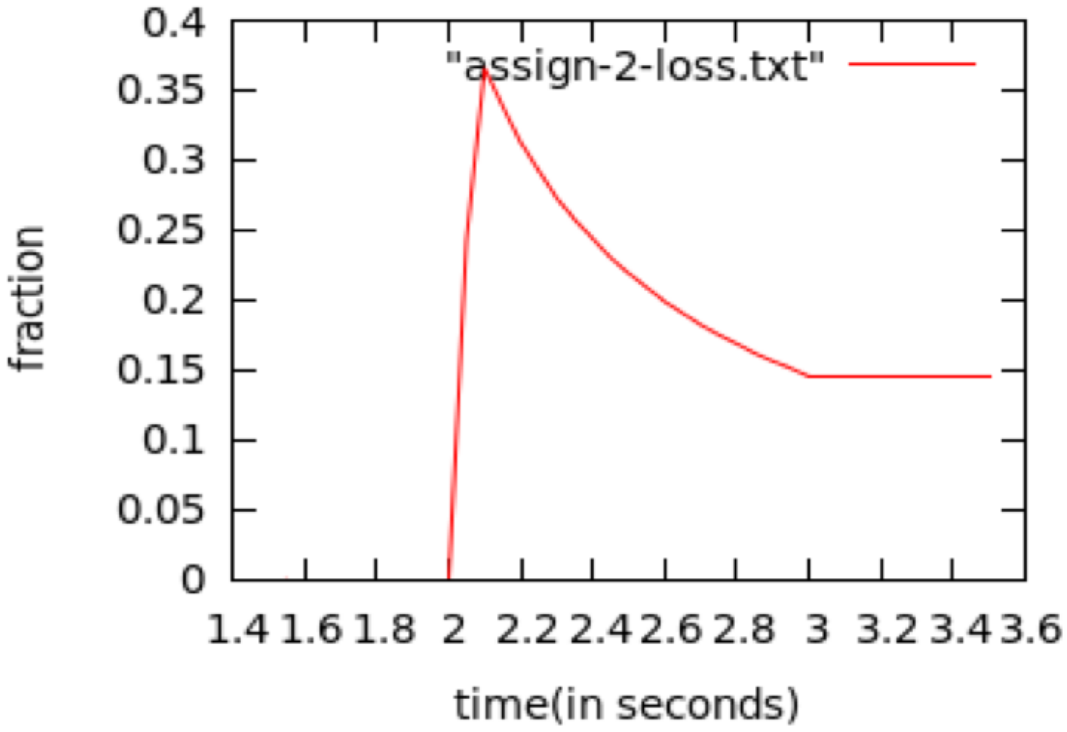
```
     network. */
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  NS_LOG_INFO ("Create Applications.");


  /* This snippet of the code is taken from ns-3/examples/wireless/wifi-
     hidden-terminal.cc  */
  /* OnOffHelper for generating the CBR (Constant Bit Rate) Traffic */
  ApplicationContainer cbrApps;
  uint16_t cbrPort = 12345;
  OnOffHelper onoff ("ns3::UdpSocketFactory",  InetSocketAddress (i1i2.
     GetAddress (1), cbrPort));
  onoff.SetAttribute ("OnTime",  StringValue ("ns3::ConstantRandomVariable
     [Constant=1]"));
  onoff.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable
     [Constant=0]"));
  onoff.SetAttribute ("PacketSize", UintegerValue (200));

  // flow n0 to n2 at 1sec 900Kbps
  onoff.SetAttribute ("DataRate", StringValue ("900Kbps"));
  onoff.SetAttribute ("StartTime", TimeValue (Seconds (1.000000)));
  onoff.SetAttribute ("StopTime", TimeValue (Seconds (3.500000)));
  cbrApps.Add (onoff.Install (nodes.Get (0)));

  // flow n0 to n1 at 1.5sec 300Kbps
  onoff.SetAttribute ("DataRate", StringValue ("300Kbps"));
  onoff.SetAttribute ("StartTime", TimeValue (Seconds (1.500000)));
  onoff.SetAttribute ("StopTime", TimeValue (Seconds (3.000000)));
  onoff.SetAttribute ("Remote",  AddressValue (Address (InetSocketAddress
     (i0i1.GetAddress (1), cbrPort))) );
  cbrApps.Add (onoff.Install (nodes.Get (0)));

  // we also use separate UDP applications that will send a single
  // packet before the CBR flows start.
  // This is a workround for the lack of perfect ARP, see Bug 187
  // http://www.nsnam.org/bugzilla/show_bug.cgi?id=187
  uint16_t  echoPort = 9;

  // ping for n0 to n1
  UdpEchoClientHelper echoClientHelper (i0i1.GetAddress (1), echoPort);
  echoClientHelper.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClientHelper.SetAttribute ("Interval", TimeValue (Seconds (0.1)));
  echoClientHelper.SetAttribute ("PacketSize", UintegerValue (10));
  ApplicationContainer pingApps;

  // again using different start times to workaround Bug 388 and Bug 912
  echoClientHelper.SetAttribute ("StartTime", TimeValue (Seconds (0.001)))
     ;
  pingApps.Add (echoClientHelper.Install (nodes.Get (0)));

  // ping for n0 to n2
  echoClientHelper.SetAttribute ("RemoteAddress", AddressValue (i1i2.
     GetAddress (1)) );
```

```
  echoClientHelper.SetAttribute ("RemotePort", UintegerValue (echoPort));
  echoClientHelper.SetAttribute ("StartTime", TimeValue (Seconds (0.006)))
     ;
  pingApps.Add (echoClientHelper.Install (nodes.Get (0)));


  // Get the IPv4 pointer of node 1, so as to schedule the setDown link
  Ptr<Node> n1 = nodes.Get (1);
  Ptr<Ipv4> ipv41 = n1->GetObject<Ipv4> ();
  // The first ifIndex is 0 for loopback, then the first p2p (n0-n1) is
     numbered 1,
  // then the next p2p (n1-n2) is numbered 2
  uint32_t ipv4ifIndex1 = 1;

  Simulator::Schedule (Seconds (2.0), &Ipv4::SetDown, ipv41, ipv4ifIndex1)
     ;
  Simulator::Schedule (Seconds (2.7), &Ipv4::SetUp, ipv41, ipv4ifIndex1);

  // Increase UDP Rate
  //  Simulator::Schedule (Seconds(30.0), &IncRate, app2, DataRate("500
     kbps"));

  /* Tracing */
  AsciiTraceHelper ascii;
  Ptr<OutputStreamWrapper> stream = ascii.CreateFileStream ("dynamic-
     global-routing.tr");
  p2p.EnableAsciiAll (stream);

  stack.EnableAsciiIpv4All (stream);
  p2p.EnablePcapAll("DynamicRoutingProtocol");

  // Flow Monitor
    if (enableFlowMonitor)
    {
      monitor = flowmon.InstallAll ();
      //monitor = flowmon.Install (nodes.Get (0));
      monitor->Start (Seconds (0.5));
    }

  for(double k=1;k<=7.0;k = k + 0.05)
  {
    Simulator::Schedule (Seconds(k), &calculateLoss);
  }
std::cout<<"fff"<<std::endl;
  /* Now, do the actual simulation. */
  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop (Seconds(8.0));
  Simulator::Run ();
std::cout<<"fff"<<std::endl;
  if (enableFlowMonitor)
    {
      monitor->CheckForLostPackets ();
 // 10. Print per flow statistics
      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)
        {

          // first 2 FlowIds are for ECHO apps, we don't want to display
             them
          if (i->first > 2)
            {
                      Ipv4FlowClassifier::FiveTuple t = classifier->
                         FindFlow (i->first);
                  if(t.destinationAddress == "10.1.3.2")
                  {
                    txPacketsum += i->second.txPackets;
                    rxPacketsum += i->second.rxPackets;
                    LostPacketsum += i->second.lostPackets;
                    DropPacketsum += i->second.packetsDropped.size();
                    Delaysum += i->second.delaySum.GetSeconds();
                  }
            }
        }
        cout << "  All Tx Packets: " << txPacketsum << "\n";
            cout << "  All Rx Packets: " << rxPacketsum << "\n";
            cout << "  All Delay: " << Delaysum / txPacketsum << "\n";
            cout << "  All Lost Packets: " << LostPacketsum << "\n";
            cout << "  All Drop Packets: " << DropPacketsum << "\n";
            cout << "  Packets Delivery Ratio: " << ((rxPacketsum * 100) /
                txPacketsum) << "%" << "\n";
            cout << "  Packets Lost Ratio: " << ((LostPacketsum * 100) /
               txPacketsum) << "%" << "\n";
    monitor->SerializeToXmlFile("Question2.flowmon", true, true);
    }

  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");
  return 0;
}
```

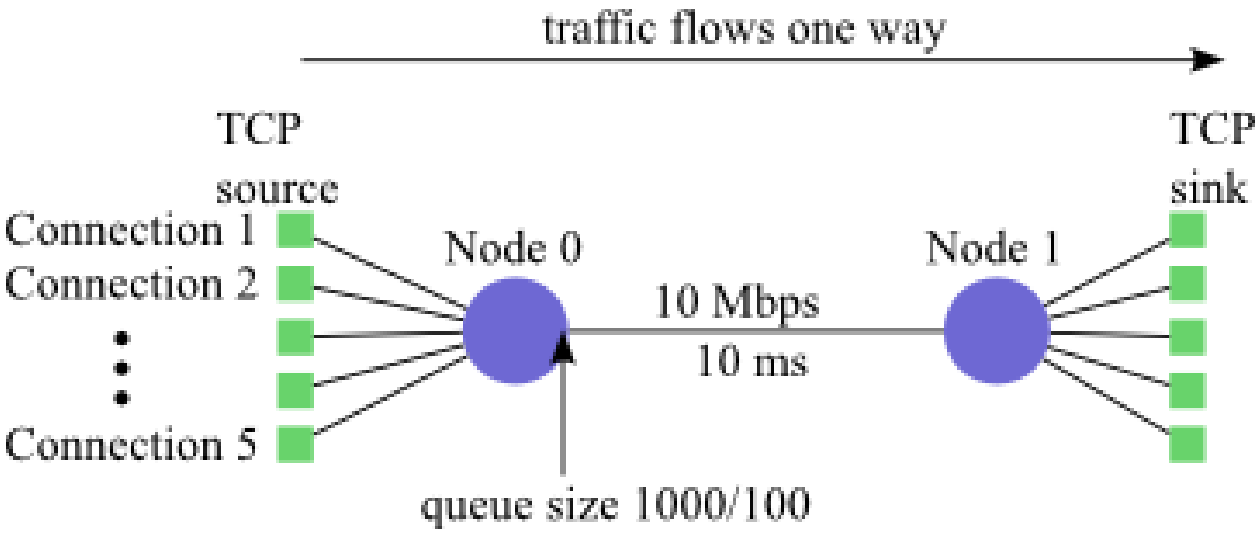
Listing 3: Question 2

While calculating the loss at node n_2, the monitor displays no packet loss. Since we are using dynamic routing protocol for routing, network takes care of the packet routing to n_2 when the P2P link between n_0 and n_1 fails. The packet loss at n_2 is almost 0.



Our statistics are :-
All Tx Packets: 1696
All Rx Packets: 1554
All Delay: 0.073852
All Lost Packets: 142
All Drop Packets: 9
Packets Delivery Ratio: 91.63
Packets Lost Ratio: 8.37

3. Consider following topology



Connection 1 starts at time 0, Connection 2 at time 5, Connection 3 at time 10, Connection 4 at time 15, and Connection 5 at time 20. End Connection 1 at time 50.1, Connection 2 at time 45, Connection 3 at time 40, Connection 4 at time 35, and Connection 5 at time 30.

Using the generated trace files, produce graphs for the following:

a) The receivers' rates over time. Calculate the rate for each TCP connection separately. Plot the rate for all of the receivers on the same graph. To plot a smooth rate, calculate the rate over a moving window of 10 packets. You can do this by plotting a point every time a packet is received. if $packet_i$ is received at time t_i, plot a point at t_i equal to the sum of the lengths of $packet_{i-10}$ through $packet_i$, divided by the t_i - t_{i-10}. You will need to modify this slightly to handle the start of the connection, when there are fewer than 10 previous packets.

b) The queue size over time and each packet drop event. You can calculate the queue size at any time by observing all packet enqueue, dequeue, and drop events. Plot each drop event at the maximum queue size when the drop occurs using an "X" symbol.

c) The Congestion Window over time for each TCP connection.

Use only TCP NewReno for this experiment. Turn in all of your graphs and then answers to the following questions

a) What do you observe about fairness among the various TCP connections?
The receiver rate as visible from the graph is fair for all the connections. The peak(highest) rate in all these connections is around 1000000 Bps except for receiver 5 which has very less time(only

10 seconds) to stabalize.For those having maximum time do reach their peak.Thus TCP is quite fair.Also note the application on receiver 1 (which runs for the maximum amount of time) does not get all the bandwidth.Though the longest running application has maximum congestion window but has almost equal average banwidth as others.

b) What do you observe about the queue size and packet drops?
Most of the time the queue is 30 percent full with rare peaks in between(Continous on majority occasions).maximum capacity between 20-30 seconds when all the 5 connections are running.

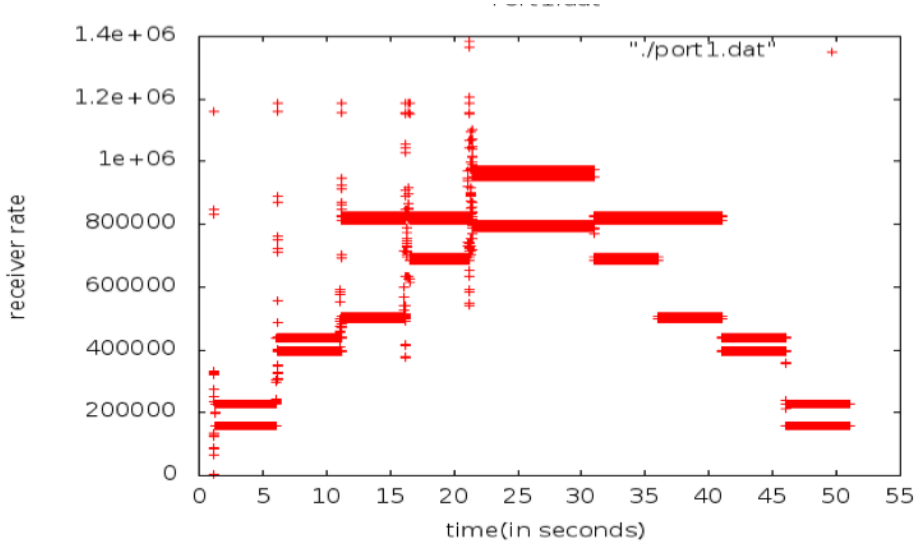
```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the 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 "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/tcp-l4-protocol.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Question3");

void
CwndTracer(uint32_t oldval, uint32_t newval)
{
  NS_LOG_UNCOND ("Congestion window moving from " << oldval << " to " <<
     newval << " at " << Simulator::Now().GetSeconds() << " seconds.");
}
void
Enqueue(std::string context, Ptr<const Packet> p)
{
  NS_LOG_INFO (context <<
                 " Packet Enqueued at " << Simulator::Now ().GetSeconds())
                    ;
}
void
Dequeue(std::string context, Ptr<const Packet> p)
```

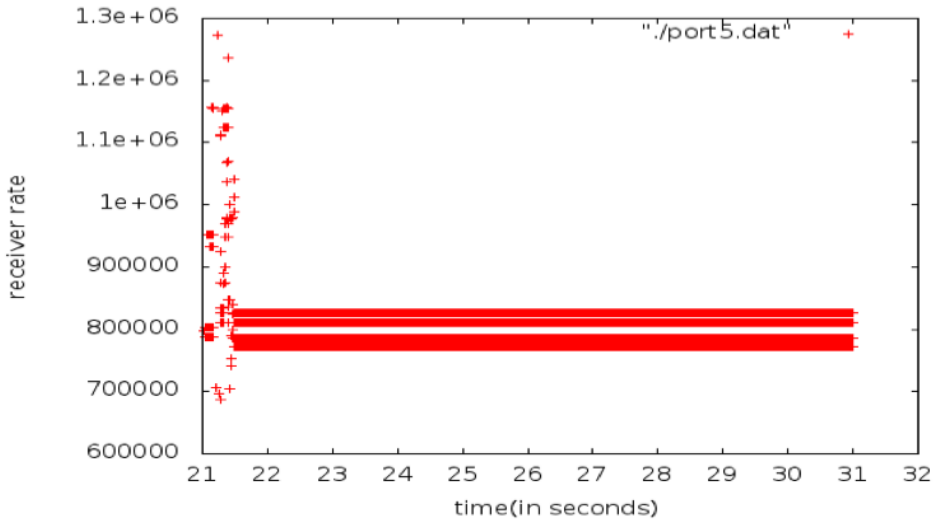
```
{
  NS_LOG_INFO (context <<
               " Packet Dequeued at " << Simulator::Now ().GetSeconds());
}
void
Drop(std::string context, Ptr<const Packet> p)
{
  NS_LOG_INFO (context <<
                 " Packet Dropped at " << Simulator::Now ().GetSeconds());
}

void
ReceivePacket (std::string context, Ptr<const Packet> p, const Address&
    addr)
{
  NS_LOG_INFO (context <<
                 " Packet Received at " << Simulator::Now ().GetSeconds()
                    << "from " << InetSocketAddress::ConvertFrom(addr).
                    GetIpv4 ());
}

int
main (int argc, char *argv[])
{
  // disable fragmentation
  Config::SetDefault ("ns3::WifiRemoteStationManager::
     FragmentationThreshold", StringValue ("2200"));
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
     StringValue ("2200"));
  // Set the size of the sending queue
  Config::SetDefault ("ns3::DropTailQueue::MaxPackets", UintegerValue(
     uint32_t(10)));
  //LogComponentEnable("Question3", LOG_LEVEL_INFO);

  uint32_t nFlows = 5;
  std::string tcpType = "NewReno";
  uint16_t port = 50000;
  CommandLine cmd;

  cmd.AddValue ("Flows", "Number of flows through two nodes", nFlows);
  cmd.AddValue ("Tcp", "Tcp type: 'NewReno', 'Tahoe', 'Reno', or 'Rfc793'"
     , tcpType);
  cmd.Parse (argc, argv);

  if(tcpType != "NewReno" && tcpType != "Tahoe" && tcpType != "Reno" &&
     tcpType != "Rfc793"){
    NS_LOG_UNCOND ("The Tcp type must be either 'NewReno', 'Tahoe', 'Reno
       ', or 'Rfc793'.");
    return 1;
  }

  // Set default Socket type to one of the Tcp Sockets
  Config::SetDefault ("ns3::TcpL4Protocol::SocketType", TypeIdValue(TypeId
     ::LookupByName ("ns3::Tcp" + tcpType)));
```

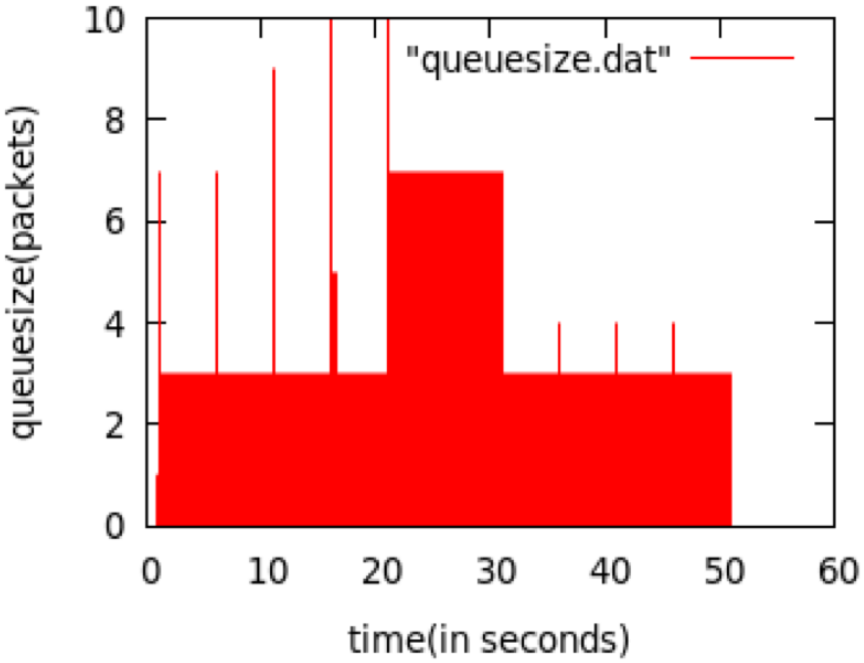
```
  NS_LOG_INFO ("Creating Topology");

  NodeContainer nodes;
  nodes.Create (2);

  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute("DataRate", DataRateValue(DataRate("1.5
      Mbps")));
  pointToPoint.SetChannelAttribute("Delay", TimeValue(Time("10ms")));

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  ApplicationContainer serverApp[nFlows];
  ApplicationContainer sinkApp[nFlows];

  for(unsigned int i = 0;i < nFlows; i++){
    PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory",
        InetSocketAddress(interfaces.GetAddress (1), port + i));
    sinkApp[i] = packetSinkHelper.Install (nodes.Get (1));
    sinkApp[i].Start(Seconds (1.0));
    sinkApp[i].Stop(Seconds (60.0));
  }

  for(unsigned int i = 0; i < nFlows; i++){
    OnOffHelper server("ns3::TcpSocketFactory", InetSocketAddress(
        interfaces.GetAddress (1), port + i));
    server.SetAttribute ("OnTime", StringValue("ns3::
        ConstantRandomVariable[Constant=50]"));
    server.SetAttribute ("OffTime", StringValue("ns3::
        ConstantRandomVariable[Constant=0]"));
    server.SetAttribute ("DataRate", DataRateValue (DataRate ("1.5Mbps")))
        ;
    server.SetAttribute ("PacketSize", UintegerValue (2000));

    serverApp[i] = server.Install (nodes.Get (0));
    serverApp[i].Start(Seconds (1 + (i * 5)));
    serverApp[i].Stop (Seconds (51.0 - (i * 5)));
  }

  NS_LOG_INFO ("Enable static global routing.");
  //
  // Turn on global static routing so we can actually be routed across the
      star.
  //
```

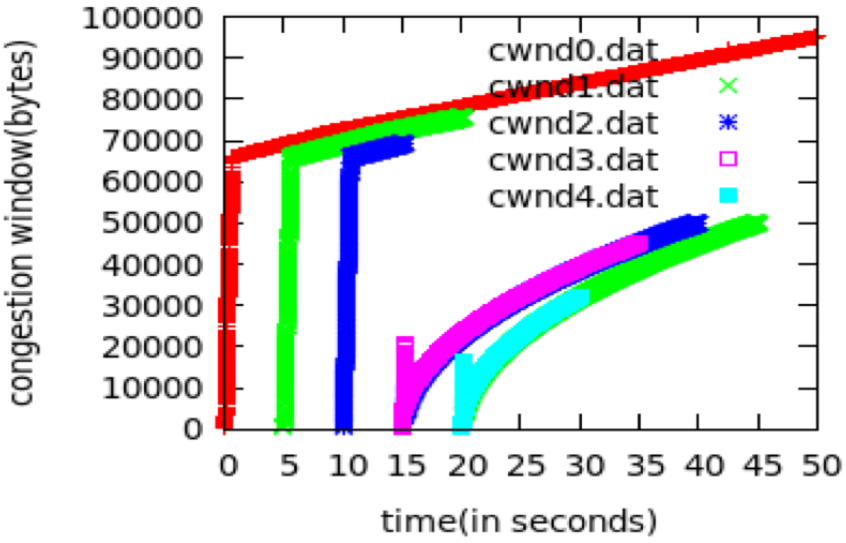
```
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  AsciiTraceHelper ascii;
  pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("Question3-nflow.tr
      "));
  pointToPoint.EnablePcapAll("Question3");

  std::string context = "/NodeList/0/DeviceList/0/$ns3::
      PointToPointNetDevice/TxQueue/";

  Config::Connect (context + "Enqueue", MakeCallback (&Enqueue));
  Config::Connect (context + "DeQueue", MakeCallback (&Dequeue));
  Config::Connect (context + "Drop", MakeCallback (&Drop));

  context = "/NodeList/1/ApplicationList/*/$ns3::PacketSink/Rx";
  Config::Connect (context, MakeCallback(&ReceivePacket));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Listing 4: `Question 3`

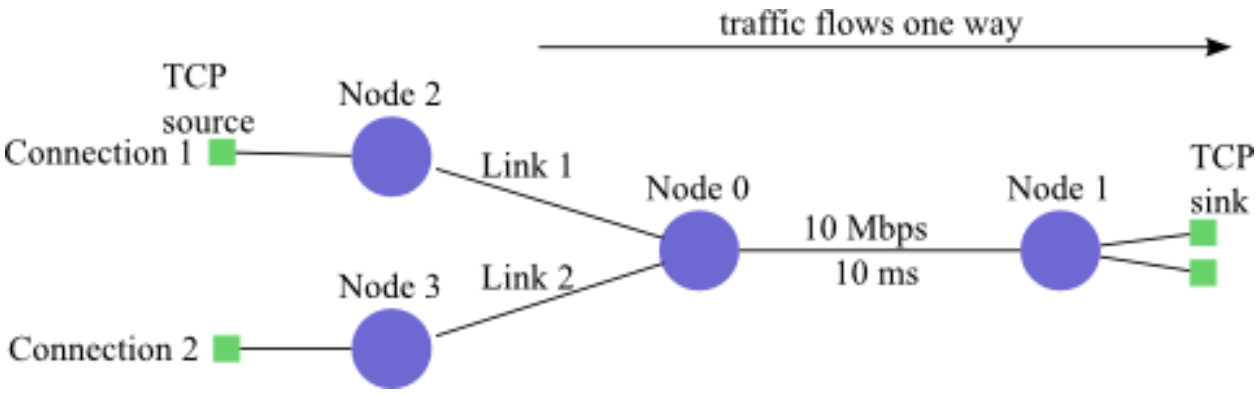



"./port5.dat"
receiver rate
1.3e+06
1.2e+06
1.1e+06
1e+06
900000
800000
700000
600000
21
22
23
24
25
26
27
28
29
30
31
32
time(in seconds)


"queuesize.dat"
queuesize(packets)
10
8
6
4
2
0
0
10
20
30
40
50
60
time(in seconds)


congestion window(bytes)
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0
5
10
15
20
25
30
35
40
45
50
time(in seconds)
cwnd0.dat
cwnd1.dat
cwnd2.dat
cwnd3.dat
cwnd4.dat

4. Consider the following topology



The queue size limit on all nodes is 10. The bandwidth on Links 1 and 2 is 1.5 Mbps. The propagation delay on Links 1 and 2 is initially 10 ms, but you will be varying the propagation delay on Link 2 for your experiments.

Connection 1 starts at time 0, using Link 1. Connection 2 starts at time 0. Both connections run for 5 seconds.

Plot a graph to show the throughput for each connection over time and observe that they get approximately the same amount. Now run a set of experiments, varying the propagation delay on Link 2 so that Connection 2 has an increasingly longer RTT. Produce the following graph:

The receiver's total rate (averaged over the length of the connection) versus the propagation delay on Link 2. Plot each connection separately.

Use only TCP NewReno for these experiments. What do you observe about the relationship ?

```
/* -*- 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
```

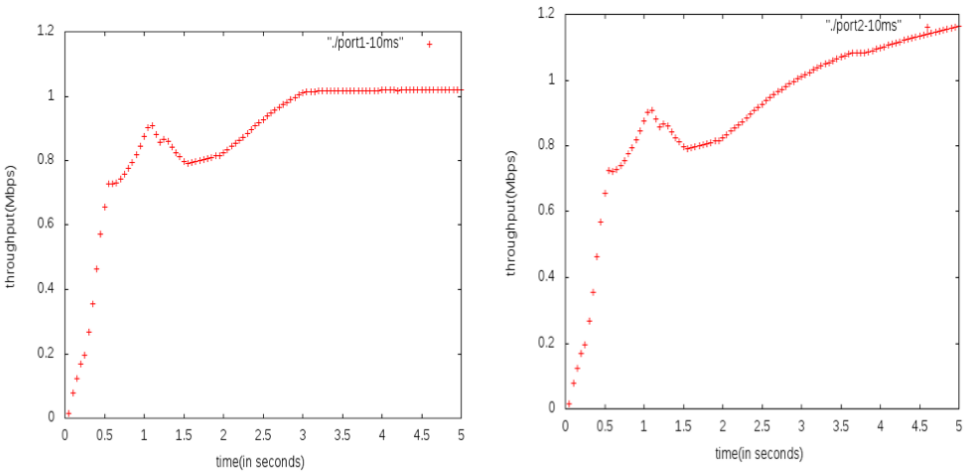
```
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
    USA
 */

#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/tcp-l4-protocol.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Question4");

static const double timeToRun = 50.5;
static double node2BytesRcv = 0;
static double node3BytesRcv = 0;

void
ReceiveNode2Packet (std::string context, Ptr<const Packet> p, const
   Address& addr)
{
  NS_LOG_INFO (context <<
                " Packet Received from Node 2 at " << Simulator::Now ().
                   GetSeconds());
  node2BytesRcv += p->GetSize ();
}

void
ReceiveNode3Packet (std::string context, Ptr<const Packet> p, const
   Address& addr)
{
  NS_LOG_INFO (context <<
                " Packet Received from Node 3 at " << Simulator::Now ().
                   GetSeconds() << "from " << InetSocketAddress::
                   ConvertFrom(addr).GetIpv4 ());
  node3BytesRcv += p->GetSize ();
}

void
ChangeDelay (Ptr<PointToPointRemoteChannel> & linkChannel)
{
  std::ostringstream new_delay;
  TimeValue val(Time("0ms"));

  linkChannel->GetAttribute("Delay", val);
  NS_LOG_INFO("The previous delay of the bottleneck link was " << val.Get
     ().GetMilliSeconds() << "ms");

  new_delay << val.Get().GetMilliSeconds() + 1 << "ms";
  val.Set (Time (new_delay.str()));
  linkChannel->SetAttribute("Delay", val);
```

```
  linkChannel->GetAttribute("Delay", val);
  NS_LOG_INFO("The new delay of the bottleneck link is " << val.Get().
     GetMilliSeconds() << "ms");

  Simulator::Schedule (Seconds (1.0), &ChangeDelay, linkChannel);
}

int
main (int argc, char *argv[])
{
  std::string tcpType = "NewReno";
  uint16_t port = 50000;
  CommandLine cmd;

  cmd.AddValue ("Tcp", "Tcp type: 'NewReno', 'Tahoe', 'Reno', or 'Rfc793'"
     , tcpType);
  cmd.Parse (argc, argv);

  if(tcpType != "NewReno" && tcpType != "Tahoe" && tcpType != "Reno" &&
     tcpType != "Rfc793"){
    NS_LOG_UNCOND ("The Tcp type must be either 'NewReno', 'Tahoe', 'Reno
       ', or 'Rfc793'.");
    return 1;
  }

  // disable fragmentation
  Config::SetDefault ("ns3::WifiRemoteStationManager::
     FragmentationThreshold", StringValue ("2200"));
  Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
     StringValue ("2200"));
  // Set tcp type
  Config::SetDefault ("ns3::TcpL4Protocol::SocketType", TypeIdValue(TypeId
     ::LookupByName ("ns3::Tcp" + tcpType)));
  // Set maximum queue size
  Config::SetDefault ("ns3::DropTailQueue::MaxPackets", UintegerValue(
     uint32_t(10)));
  //LogComponentEnable("Question4", LOG_LEVEL_INFO);

  NS_LOG_INFO ("Creating Topology");

  NodeContainer nodes;
  nodes.Create (4);
  NodeContainer n0n1 (nodes.Get (0), nodes.Get (1));
  NodeContainer n2n0 (nodes.Get (2), nodes.Get (0));
  NodeContainer n3n0 (nodes.Get (3), nodes.Get (0));

  PointToPointHelper link;
  link.SetDeviceAttribute("DataRate", DataRateValue(DataRate("1.5Mbps")));
  link.SetChannelAttribute("Delay", TimeValue(Time("10ms")));

  NetDeviceContainer d0d1 = link.Install (n0n1);
  NetDeviceContainer d2d0 = link.Install(n2n0);
  NetDeviceContainer d3d0 = link.Install(n3n0);
```

```
InternetStackHelper stack;
stack.Install (nodes);

Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer i0i1 = address.Assign (d0d1);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer i2i0 = address.Assign (d2d0);

address.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer i3i0 = address.Assign (d3d0);

ApplicationContainer apps;

OnOffHelper source("ns3::TcpSocketFactory", InetSocketAddress(i0i1.
   GetAddress (1), port));
source.SetAttribute ("OnTime", RandomVariableValue (ConstantVariable
   (50)));
source.SetAttribute ("OffTime", RandomVariableValue (ConstantVariable
   (0)));
source.SetAttribute ("DataRate", DataRateValue (DataRate ("1.5Mbps")));
source.SetAttribute ("PacketSize", UintegerValue (2000));

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

source.SetAttribute ("Remote", AddressValue(InetSocketAddress(i0i1.
   GetAddress (1), port + 1)));
apps.Add (source.Install (nodes.Get (3)));

PacketSinkHelper sink ("ns3::TcpSocketFactory", InetSocketAddress(i0i1.
   GetAddress (1), port));
apps.Add(sink.Install (nodes.Get (1)));

sink.SetAttribute("Local", AddressValue(InetSocketAddress(i0i1.
   GetAddress(1), port + 1)));
apps.Add (sink.Install (nodes.Get (1)));

apps.Start(Seconds (0.5));
apps.Stop(Seconds (timeToRun));

NS_LOG_INFO ("Enable static global routing.");
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Ptr<PointToPointRemoteChannel> linkChannel ((PointToPointRemoteChannel*)
    &(*(nodes.Get(3)->GetDevice (0)->GetChannel ())));
Simulator::Schedule (Seconds (1.5), &ChangeDelay, linkChannel);

std::string context = "/NodeList/1/ApplicationList/0/$ns3::PacketSink/Rx
   ";
Config::Connect (context, MakeCallback(&ReceiveNode2Packet));

context = "/NodeList/1/ApplicationList/1/$ns3::PacketSink/Rx";
```

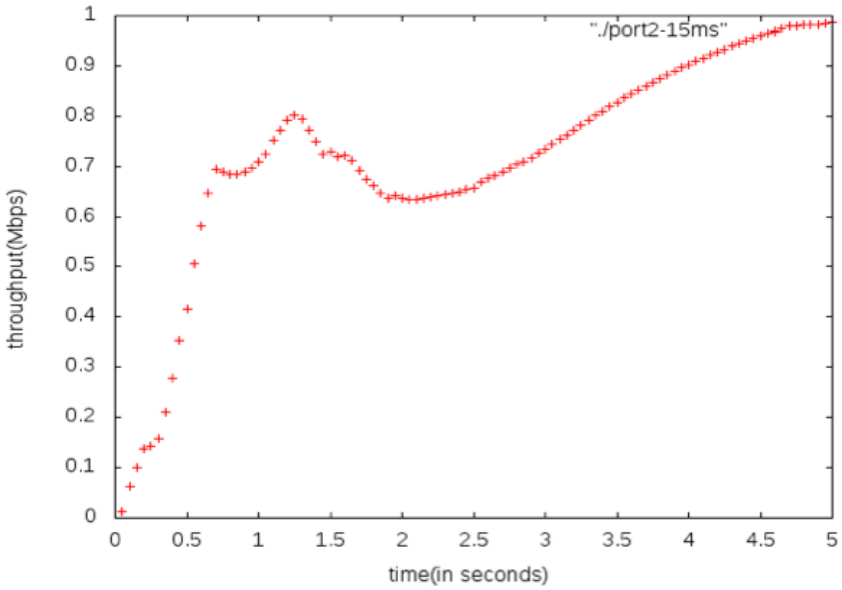
```
  Config::Connect (context, MakeCallback(&ReceiveNode3Packet));

  AsciiTraceHelper ascii;
  link.EnableAsciiAll (ascii.CreateFileStream ("Question4.tr"));
  link.EnablePcapAll ("Question4");

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

  std::cout << "Throughput from Node 2: " << (node2BytesRcv * 8 / 1000000)
      / (timeToRun - 0.5) << " Mbps" << std::endl;
  std::cout << "Throughput from Node 3: " << (node3BytesRcv * 8 / 1000000)
      / (timeToRun - 0.5) << " Mbps" << std::endl;

  return 0;
}
```

Listing 5: `Question 4`



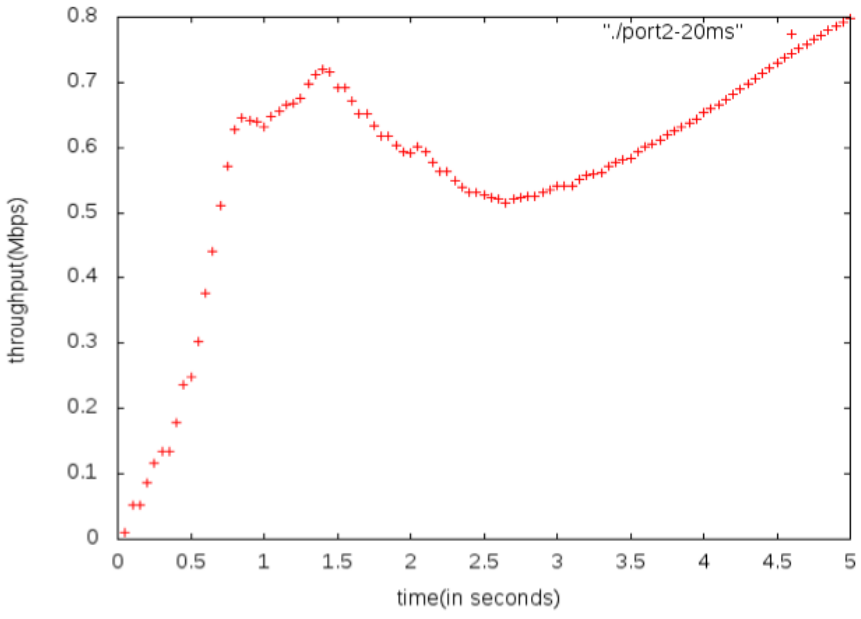
At 10 ms latency both have approximately same throughput
Link 1:- 1.014 Mbps
Link 2:- 1.004 Mbps

When Link 2's latency is changed to 15 ms,



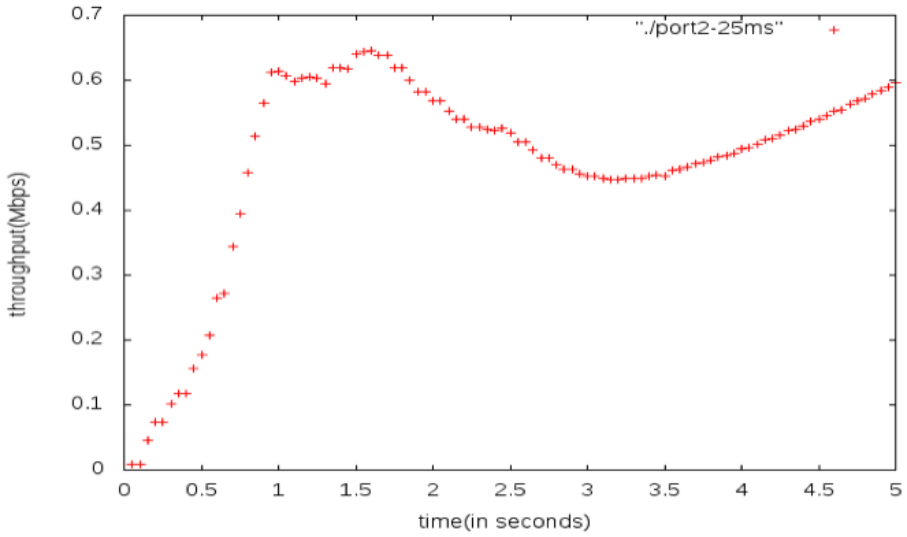
Link 1:- 1.014 Mbps
Link 2:- 0.989 Mbps

Link 2's latency is changed to 20 ms.

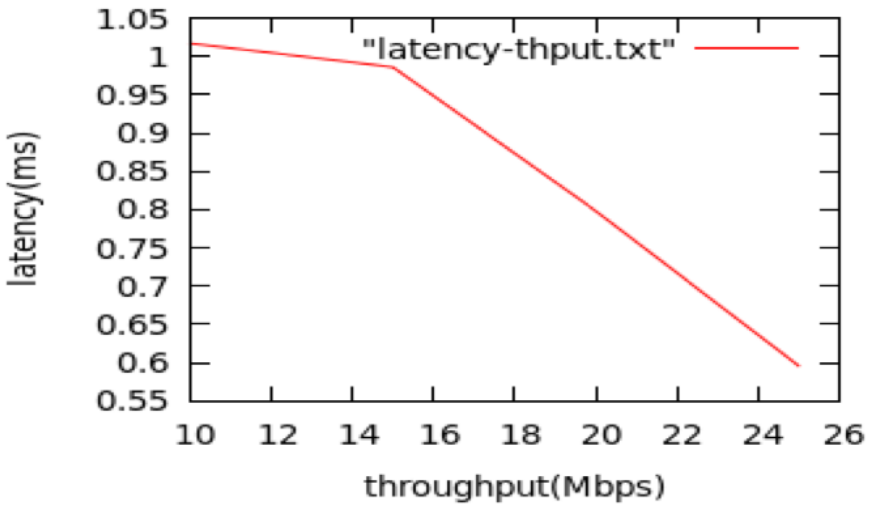


Link 1:- 1.014 Mbps
Link 2:– 0.801 Mbps

Link 2's latency is changed to 25 ms.



Link 1:- 1.019 Mbps
Link 2:- 0.599 Mbps



You can see as the latency increases the throughput decreases.
Hence we can conclude that TCP is unfair to slow connections or those which have large latency.