Mawlana Bashani Science & technology University
Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology
Report No: 03
Report Name: TCP and router queues.
Course Title: Wireless and Mobile Communication.
Course Code: ICT-4201

Submitted By	Submitted To
Name: **Ismayel Hossen** ID: **IT-16054** Session: 2015-16 4th Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	Nazrul Islam Assistant Professor Dept. of Information & Communication Technology, MBSTU.

Submission Date: 11-09-2020

Objective :

1.As in previous exercise, Create a simple dumbbell topology, two client Node1 and Node2 on the left side of the dumbbell and server nodes Node3 and Node4 on the right side of the dumbbell. Let Node5 and Node6 form the bridge of the dumbbell. Use point to point links.
2.Add drop tail queues of size QueueSize5 and QueueSize6 to Node5 and Node6, respectively.
3.Install a TCP socket instance on Node1 that will connect to Node3.
4.Install a TCP socket instance on Node2 that will connect to Node3.
5.Install a TCP socket instance on Node2 that will connect to Node4.
6.Start Node1--Node3 flow at time 1s, then measure it's throughput. How long does it take to fill link's entire capacity?
7.Start Node2--Node3 and Node2--Node4 flows at time 15s, measure their throughput.
8.Measure packet loss and cwnd size, and plot graphs throughput/time, cwnd/time and packet loss/time for each of the flows.
9.Plot graph throughput/cwnd and packet loss/cwnd for the first flow. Is there an optimal value for cwnd?
10.Vary QueueSize5 and QueueSize6. Which one has immediate effect on cwnd size of the first flow

Source Code:

```
#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/internet-apps-module.h"
#include "ns3/traffic-control-module.h"
```

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

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BenchmarkQueueDiscs");

void
LimitsTrace (Ptr<OutputStreamWrapper> stream, uint32_t oldVal,
uint32_t newVal)
{
  *stream->GetStream () << Simulator::Now ().GetSeconds () << " "
<< newVal << std::endl;
}

void
BytesInQueueTrace (Ptr<OutputStreamWrapper> stream, uint32_t
oldVal, uint32_t newVal)
{
  *stream->GetStream () << Simulator::Now ().GetSeconds () << " "
<< newVal << std::endl;
}

static void
GoodputSampling (std::string fileName, ApplicationContainer app,
Ptr<OutputStreamWrapper> stream, float period)
{
  Simulator::Schedule (Seconds (period), &GoodputSampling,
fileName, app, stream, period);
  double goodput;
```

```
  uint64_t totalPackets = DynamicCast<PacketSink> (app.Get (0))-
>GetTotalRx ();
  goodput = totalPackets * 8 / (Simulator::Now ().GetSeconds () *
1024); // Kbit/s
  *stream->GetStream () << Simulator::Now ().GetSeconds () << " "
<< goodput << std::endl;
}

static void PingRtt (std::string context, Time rtt)
{
  std::cout << context << "=" << rtt.GetMilliSeconds () << " ms" <<
std::endl;
}

int main (int argc, char *argv[])
{
  std::string bandwidth = "10Mbps";
  std::string delay = "5ms";
  std::string queueDiscType = "PfifoFast";
  uint32_t queueDiscSize = 1000;
  uint32_t netdevicesQueueSize = 50;
  bool bql = false;

  std::string flowsDatarate = "20Mbps";
  uint32_t flowsPacketsSize = 1000;

  float startTime = 0.1f; // in s
  float simDuration = 60;
  float samplingPeriod = 1;
```

```
  CommandLine cmd;
  cmd.AddValue ("bandwidth", "Bottleneck bandwidth", bandwidth);
  cmd.AddValue ("delay", "Bottleneck delay", delay);
  cmd.AddValue ("queueDiscType", "Bottleneck queue disc type in
{PfifoFast, ARED, CoDel, FqCoDel, PIE, prio}", queueDiscType);
  cmd.AddValue ("queueDiscSize", "Bottleneck queue disc size in
packets", queueDiscSize);
  cmd.AddValue ("netdevicesQueueSize", "Bottleneck netdevices
queue size in packets", netdevicesQueueSize);
  cmd.AddValue ("bql", "Enable byte queue limits on bottleneck
netdevices", bql);
  cmd.AddValue ("flowsDatarate", "Upload and download flows
datarate", flowsDatarate);
  cmd.AddValue ("flowsPacketsSize", "Upload and download flows
packets sizes", flowsPacketsSize);
  cmd.AddValue ("startTime", "Simulation start time", startTime);
  cmd.AddValue ("simDuration", "Simulation duration in seconds",
simDuration);
  cmd.AddValue ("samplingPeriod", "Goodput sampling period in
seconds", samplingPeriod);
  cmd.Parse (argc, argv);

  float stopTime = startTime + simDuration;

  // Create nodes
  NodeContainer n1, n2, n3;
  n1.Create (1);
  n2.Create (1);
  n3.Create (1);

  // Create and configure access link and bottleneck link
```

```
  PointToPointHelper accessLink;
  accessLink.SetDeviceAttribute ("DataRate", StringValue
("100Mbps"));
  accessLink.SetChannelAttribute ("Delay", StringValue ("0.1ms"));

  PointToPointHelper bottleneckLink;
  bottleneckLink.SetDeviceAttribute ("DataRate", StringValue
(bandwidth));
  bottleneckLink.SetChannelAttribute ("Delay", StringValue (delay));

  InternetStackHelper stack;
  stack.InstallAll ();

  // Access link traffic control configuration
  TrafficControlHelper tchPfifoFastAccess;
  tchPfifoFastAccess.SetRootQueueDisc ("ns3::PfifoFastQueueDisc",
"MaxSize", StringValue ("1000p"));

  // Bottleneck link traffic control configuration
  TrafficControlHelper tchBottleneck;

  if (queueDiscType.compare ("PfifoFast") == 0)
    {
      tchBottleneck.SetRootQueueDisc ("ns3::PfifoFastQueueDisc",
"MaxSize",
                                      QueueSizeValue (QueueSize
(QueueSizeUnit::PACKETS, queueDiscSize)));
    }
  else if (queueDiscType.compare ("ARED") == 0)
    {
```

```
      tchBottleneck.SetRootQueueDisc ("ns3::RedQueueDisc");
      Config::SetDefault ("ns3::RedQueueDisc::ARED", BooleanValue
(true));
      Config::SetDefault ("ns3::RedQueueDisc::MaxSize",
                          QueueSizeValue (QueueSize
(QueueSizeUnit::PACKETS, queueDiscSize)));
    }
  else if (queueDiscType.compare ("CoDel") == 0)
    {
      tchBottleneck.SetRootQueueDisc ("ns3::CoDelQueueDisc");
      Config::SetDefault ("ns3::CoDelQueueDisc::MaxSize",
                          QueueSizeValue (QueueSize
(QueueSizeUnit::PACKETS, queueDiscSize)));
    }
  else if (queueDiscType.compare ("FqCoDel") == 0)
    {
      tchBottleneck.SetRootQueueDisc ("ns3::FqCoDelQueueDisc");
      Config::SetDefault ("ns3::FqCoDelQueueDisc::MaxSize",
                          QueueSizeValue (QueueSize
(QueueSizeUnit::PACKETS, queueDiscSize)));
    }
  else if (queueDiscType.compare ("PIE") == 0)
    {
      tchBottleneck.SetRootQueueDisc ("ns3::PieQueueDisc");
      Config::SetDefault ("ns3::PieQueueDisc::MaxSize",
                          QueueSizeValue (QueueSize
(QueueSizeUnit::PACKETS, queueDiscSize)));
    }
  else if (queueDiscType.compare ("prio") == 0)
    {
```

```
    uint16_t handle = tchBottleneck.SetRootQueueDisc
("ns3::PrioQueueDisc", "Priomap",
                                        StringValue ("0 1 0 1 0 1 0 1 0 1 0 1
0 1 0 1"));
    TrafficControlHelper::ClassIdList cid =
tchBottleneck.AddQueueDiscClasses (handle, 2,
"ns3::QueueDiscClass");
    tchBottleneck.AddChildQueueDisc (handle, cid[0],
"ns3::FifoQueueDisc");
    tchBottleneck.AddChildQueueDisc (handle, cid[1],
"ns3::RedQueueDisc");
   }
  else
   {
    NS_ABORT_MSG ("--queueDiscType not valid");
   }

  if (bql)
   {
    tchBottleneck.SetQueueLimits ("ns3::DynamicQueueLimits");
   }

  Config::SetDefault ("ns3::QueueBase::MaxSize", StringValue
("100p"));

  NetDeviceContainer devicesAccessLink = accessLink.Install (n1.Get
(0), n2.Get (0));
  tchPfifoFastAccess.Install (devicesAccessLink);
  Ipv4AddressHelper address;
  address.SetBase ("192.168.0.0", "255.255.255.0");
  address.NewNetwork ();
```

```
  Ipv4InterfaceContainer interfacesAccess = address.Assign
(devicesAccessLink);

  Config::SetDefault ("ns3::QueueBase::MaxSize", StringValue
(std::to_string (netdevicesQueueSize) + "p"));

  NetDeviceContainer devicesBottleneckLink = bottleneckLink.Install
(n2.Get (0), n3.Get (0));
  QueueDiscContainer qdiscs;
  qdiscs = tchBottleneck.Install (devicesBottleneckLink);

  address.NewNetwork ();
  Ipv4InterfaceContainer interfacesBottleneck = address.Assign
(devicesBottleneckLink);

  Ptr<NetDeviceQueueInterface> interface =
devicesBottleneckLink.Get (0)-
>GetObject<NetDeviceQueueInterface> ();
  Ptr<NetDeviceQueue> queueInterface = interface->GetTxQueue (0);
  Ptr<DynamicQueueLimits> queueLimits =
StaticCast<DynamicQueueLimits> (queueInterface->GetQueueLimits
());

  AsciiTraceHelper ascii;
  if (bql)
    {
      queueDiscType = queueDiscType + "-bql";
      Ptr<OutputStreamWrapper> streamLimits = ascii.CreateFileStream
(queueDiscType + "-limits.txt");
      queueLimits->TraceConnectWithoutContext
("Limit",MakeBoundCallback (&LimitsTrace, streamLimits));
```

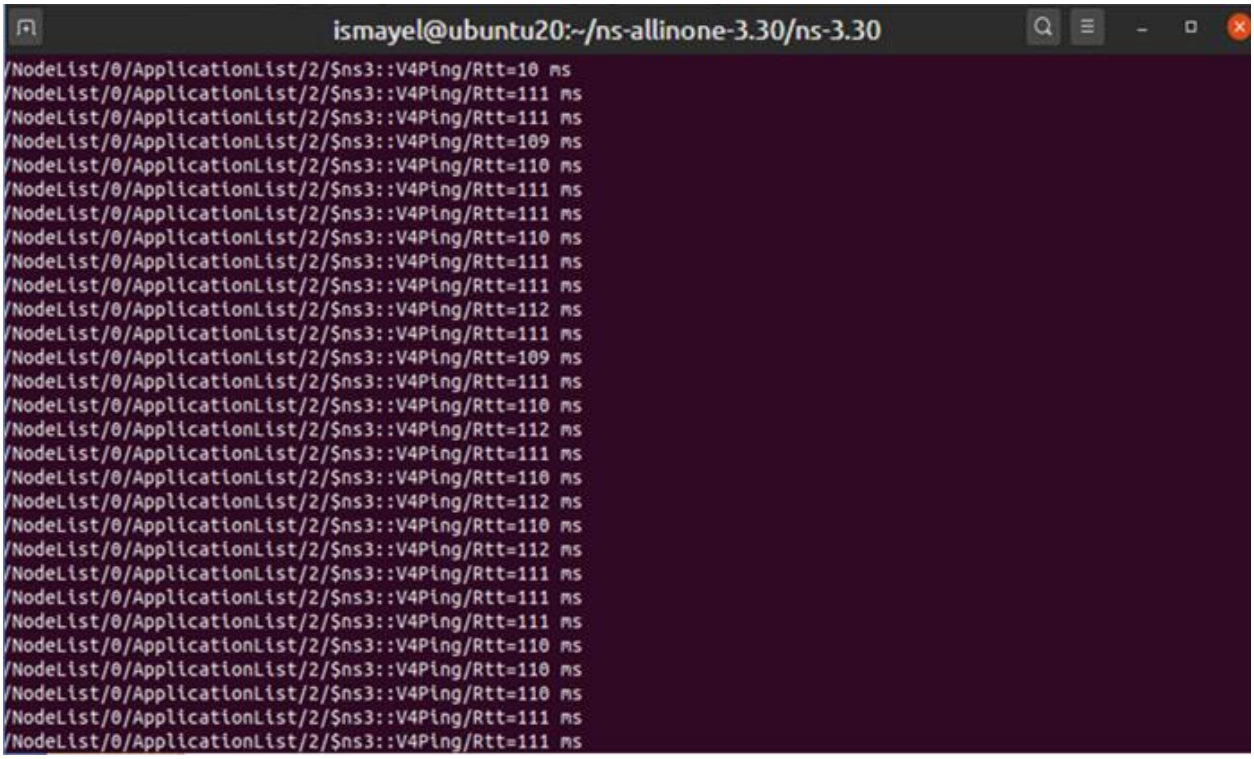
```
    }
  Ptr<Queue<Packet> > queue = StaticCast<PointToPointNetDevice>
(devicesBottleneckLink.Get (0))->GetQueue ();
  Ptr<OutputStreamWrapper> streamBytesInQueue =
ascii.CreateFileStream (queueDiscType + "-bytesInQueue.txt");
  queue->TraceConnectWithoutContext
("BytesInQueue",MakeBoundCallback (&BytesInQueueTrace,
streamBytesInQueue));

  Ipv4InterfaceContainer n1Interface;
  n1Interface.Add (interfacesAccess.Get (0));

  Ipv4InterfaceContainer n3Interface;
  n3Interface.Add (interfacesBottleneck.Get (1));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Config::SetDefault ("ns3::TcpSocket::SegmentSize", UintegerValue
(flowsPacketsSize));

  // Flows configuration
  // Bidirectional TCP streams with ping like flent tcp_bidirectional
test.
  uint16_t port = 7;
  ApplicationContainer uploadApp, downloadApp, sourceApps;
  // Configure and install upload flow
  Address addUp (InetSocketAddress (Ipv4Address::GetAny (), port));
  PacketSinkHelper sinkHelperUp ("ns3::TcpSocketFactory", addUp);
  sinkHelperUp.SetAttribute ("Protocol", TypeIdValue
(TcpSocketFactory::GetTypeId ()));
```

```
  uploadApp.Add (sinkHelperUp.Install (n3));

  InetSocketAddress socketAddressUp = InetSocketAddress
(n3Interface.GetAddress (0), port);
  OnOffHelper onOffHelperUp ("ns3::TcpSocketFactory", Address ());
  onOffHelperUp.SetAttribute ("Remote", AddressValue
(socketAddressUp));
  onOffHelperUp.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
  onOffHelperUp.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
  onOffHelperUp.SetAttribute ("PacketSize", UintegerValue
(flowsPacketsSize));
  onOffHelperUp.SetAttribute ("DataRate", StringValue
(flowsDatarate));
  sourceApps.Add (onOffHelperUp.Install (n1));

  port = 8;
  // Configure and install download flow
  Address addDown (InetSocketAddress (Ipv4Address::GetAny (),
port));
  PacketSinkHelper sinkHelperDown ("ns3::TcpSocketFactory",
addDown);
  sinkHelperDown.SetAttribute ("Protocol", TypeIdValue
(TcpSocketFactory::GetTypeId ()));
  downloadApp.Add (sinkHelperDown.Install (n1));

  InetSocketAddress socketAddressDown = InetSocketAddress
(n1Interface.GetAddress (0), port);
  OnOffHelper onOffHelperDown ("ns3::TcpSocketFactory", Address
());
```

```
  onOffHelperDown.SetAttribute ("Remote", AddressValue
(socketAddressDown));
  onOffHelperDown.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
  onOffHelperDown.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
  onOffHelperDown.SetAttribute ("PacketSize", UintegerValue
(flowsPacketsSize));
  onOffHelperDown.SetAttribute ("DataRate", StringValue
(flowsDatarate));
  sourceApps.Add (onOffHelperDown.Install (n3));

  // Configure and install ping
  V4PingHelper ping = V4PingHelper (n3Interface.GetAddress (0));
  ping.Install (n1);

  Config::Connect ("/NodeList/*/ApplicationList/*/$ns3::V4Ping/Rtt",
MakeCallback (&PingRtt));

  uploadApp.Start (Seconds (0));
  uploadApp.Stop (Seconds (stopTime));
  downloadApp.Start (Seconds (0));
  downloadApp.Stop (Seconds (stopTime));

  sourceApps.Start (Seconds (0 + 0.1));
  sourceApps.Stop (Seconds (stopTime - 0.1));

  Ptr<OutputStreamWrapper> uploadGoodputStream =
ascii.CreateFileStream (queueDiscType + "-upGoodput.txt");
  Simulator::Schedule (Seconds (samplingPeriod), &GoodputSampling,
queueDiscType + "-upGoodput.txt", uploadApp,
```

```
                uploadGoodputStream, samplingPeriod);
  Ptr<OutputStreamWrapper> downloadGoodputStream =
ascii.CreateFileStream (queueDiscType + "-downGoodput.txt");
  Simulator::Schedule (Seconds (samplingPeriod), &GoodputSampling,
queueDiscType + "-downGoodput.txt", downloadApp,
                downloadGoodputStream, samplingPeriod);

  // Flow monito
  Ptr<FlowMonitor> flowMonitor;
  FlowMonitorHelper flowHelper;
  flowMonitor = flowHelper.InstallAll();
  Simulator::Stop (Seconds (stopTime));
  Simulator::Run ();
  flowMonitor->SerializeToXmlFile(queueDiscType + "-
flowMonitor.xml", true, true);
  Simulator::Destroy ();
  return 0;
}
```

Output:



Conclusion: from there, we have learnt about TCP & Router Queues, packet drops and their effect on congestion window size.