MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

LAB-REPORT

Report No : 02

Course Code : ICT-4202

Course Title : Wireless and Mobile Communication Lab.

Date of Performance :08-09-2020

Date of Submission : 18-09-2020

Submitted By

Name: Md. Fahim Al Mamun

ID: IT-15006

4th year 2nd Semester

Session: 2014-15/2015-16

Dept. of ICT

MBSTU.

Submitted To

Nazrul Islam

Assistant Professor

Dept. of ICT

MBSTU.

Objective: We have to 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.Install a TCP socket instance on Node1 that will connect to Node3.install a UDP socket instance on Node2 that will connect to Node4.

Source Code:

```
#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"

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, DataRatedataRate);

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;

DataRatem_dataRate;

EventIdm_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, DataRatedataRate)

{

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 ())


    }
  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);
    }
```

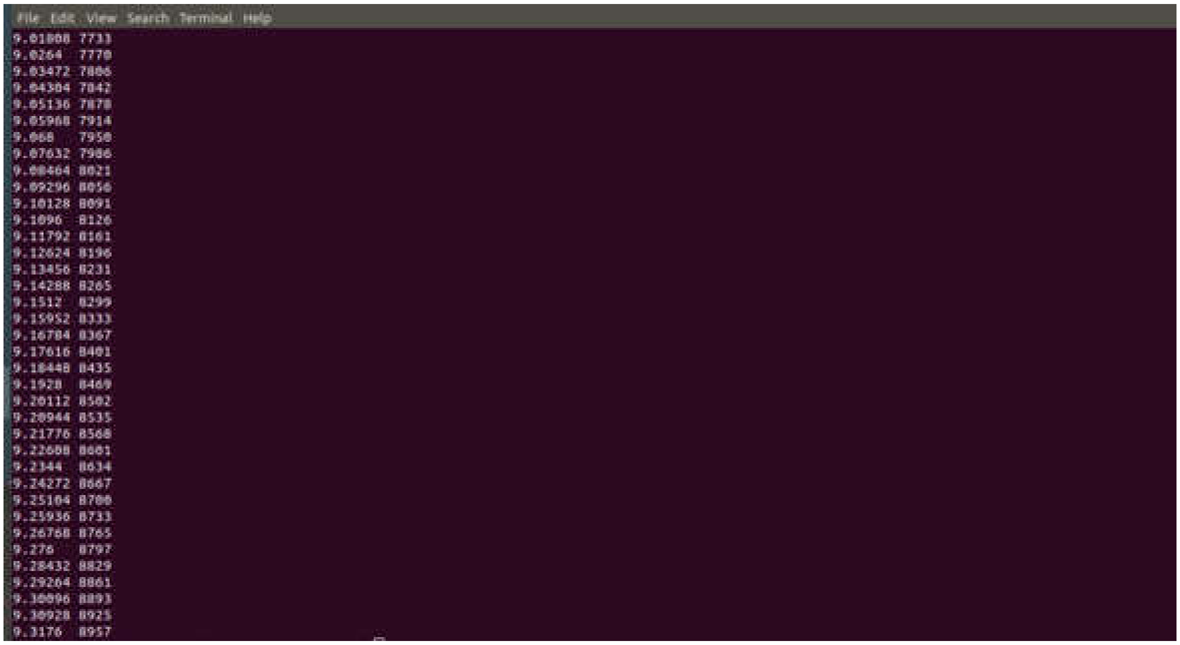
```
}static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" <<newCwnd);
}
static void
RxDrop (Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " <<Simulator::Now ().GetSeconds ());
}
int
main (intargc, char *argv[])
{
CommandLinecmd;
cmd.Parse (argc, argv);
NodeContainer nodes;
nodes.Create (2);
PointToPointHelperpointToPoint;
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));

PacketSinkHelperpacketSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort));

ApplicationContainersinkApps = 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));

Simulator::Stop (Seconds (20));

Simulator::Run ();

Simulator::Destroy ();

  return 0;

}
```

<u>Output:</u>

```
Waf: Entering directory `/home/habib/ns-allinone-3.29/ns-3.29/build'
Waf: Leaving directory `/home/habib/ns-allinone-3.29/ns-3.29/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (15.838s)
1.00419 536
1.0093  1072
1.01528 1608
1.02167 2144
1.02999 2680
1.03831 3216
1.04663 3752
1.05495 4288
1.06327 4824
1.07159 5360
1.07991 5896
1.08823 6432
1.09655 6968
1.10487 7504
1.11319 8040
1.12151 8576
1.12983 9112
RxDrop at 1.13696
1.13815 9648
1.1548  1072
1.16476 1340
1.17232 1554
1.18064 1738
1.18896 1903
1.19728 2053
1.2056  2192
1.21392 2323
1.22224 2446
1.23056 2563
1.23888 2675
1.2472  2782
1.25552 2885
```



<u>Conclusion:</u>

Transmission Control Protocol (TCP) uses a network congestion-avoidance algorithm that includes various aspects of an additive increase/multiplicative decrease(AIMD) scheme, along with other schemes including slow start and congestion window, to achieve congestion avoidance. The TCP congestion-avoidance algorithm is the primary basis for congestion controlin the Internet. Per the end-to-end principle congestion control is largely a function of internet hosts, not the network itself. There are several variations and versions of the algorithm implemented in protocol stacks of operating systems of computers that connect to the Internet.