Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology

Report No: 02
Report Name: TCP Variants.

Course Title: Wireless and Mobile Communication Lab
Course Code: ICT-4202

Submitted By	Submitted To
Name: **Jahid Tanvir** ID: **IT-15041** Session: 2014-15/15-16 4th Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	**Nazrul Islam** Assistant Professor Dept. of Information & Communication Technology, MBSTU.

Submission Date: 28-08-2020

Experiment No: 02
Name of Experiment: TCP Variants.

Objectives:

1. 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. Install a TCP socket instance on Node1 that will connect to Node3.
3. Install a UDP socket instance on Node2 that will connect to Node4.
4. Start the TCP application at time 1s.
5. Start the UDP application at time 20s at rate Rate1 such that it clogs half the dumbbell bridge's link capacity.
6. Increase the UDP application's rate at time 30s to rate Rate2 such that it clogs the whole of the dumbbell bridge's capacity.
7. Use the ns-3 tracing mechanism to record changes in congestion window size of the TCP instance over time. Use gnuplot/matplotlib to visualize plots of cwnd vs. time.
8. Mark points of fast recovery and slow start in the graphs.
9. Perform the above experiment for TCP variants Tahoe, Reno and New Reno, all of which are available with ns-3.

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, 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 ();
    }
}
```

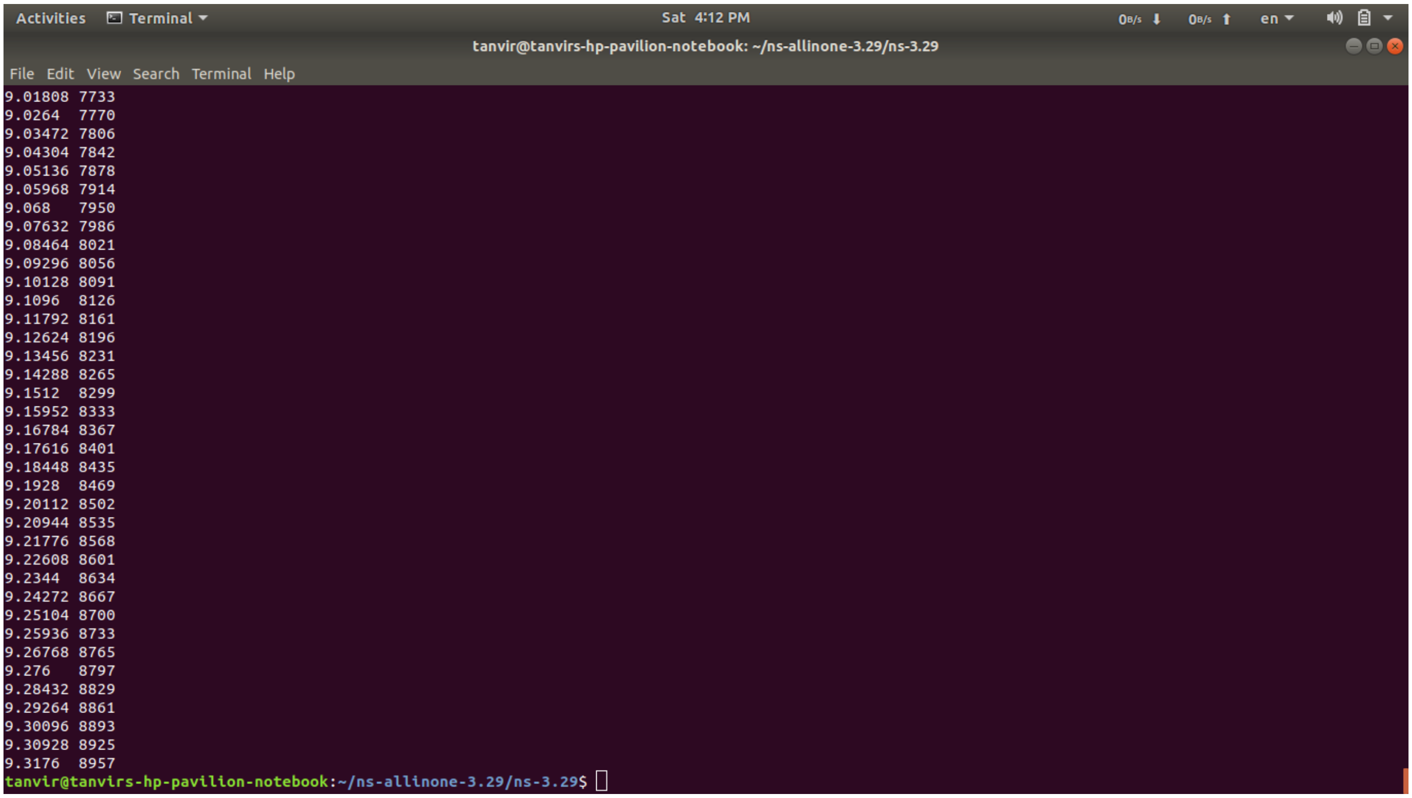
```
void
MyApp::ScheduleTx (void)
{
  if (m_running)
    {
      Time tNext (Seconds (m_packetSize * 8 / static_cast<double> (m_dataRate.GetBitRate ())));
      m_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);
    }
}

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;
  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));

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

  return 0;
}
```

OUTPUT:

```
tanvir@tanvirs-hp-pavilion-notebook:~$ cd ns-allinone-3.29
tanvir@tanvirs-hp-pavilion-notebook:~/ns-allinone-3.29$ cd ns-3.29
tanvir@tanvirs-hp-pavilion-notebook:~/ns-allinone-3.29/ns-3.29$ ./waf --run scratch/fifth
Waf: Entering directory `/home/tanvir/ns-allinone-3.29/ns-3.29/build'
[2561/2616] Compiling scratch/fifth.cc
[2562/2616] Compiling scratch/scratch-simulator.cc
[2563/2616] Compiling scratch/subdir/scratch-simulator-subdir.cc
[2564/2616] Compiling scratch/first.cc
[2573/2616] Linking build/scratch/subdir/subdir
[2574/2616] Linking build/scratch/scratch-simulator
[2575/2616] Linking build/scratch/first
[2576/2616] Linking build/scratch/fifth
Waf: Leaving directory `/home/tanvir/ns-allinone-3.29/ns-3.29/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (50.987s)
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
```

```
2.51095 8568
2.51927 8601
2.52759 8634
RxDrop at 2.53382
2.53682 8667
RxDrop at 2.54304
2.55258 1072
2.57267 1340
2.57865 1554
2.58583 1738
2.59415 1903
2.60247 2053
RxDrop at 2.6087
2.6117  2192
2.62002 1072
2.63003 1340
2.63664 1554
2.64496 1738
2.65328 1903
2.6616  2053
2.66992 2192
2.67824 2323
2.68656 2446
2.69488 2563
2.7032  2675
2.71152 2782
2.71984 2885
2.72816 2984
2.73648 3080
2.7448  3173
2.75312 3263
2.76144 3351
2.76976 3436
2.77808 3519
2.7864  3600
2.79472 3679
2.80304 3757
2.81136 3833
```



Conclusion:

TCP (Transmission Control Protocol) 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 control in 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.