**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: **Shamsunnahar**<br>ID: **IT-16034**<br>Session: 2015-16<br>4th Year 2nd Semester<br>Dept. of Information & Communication Technology, MBSTU. | Nazrul Islam<br>Assistant Professor<br>Dept. of Information & Communication Technology, MBSTU. |

Submission Date: 11-09-2020

**Objective :**

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 visualise 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;
}
```

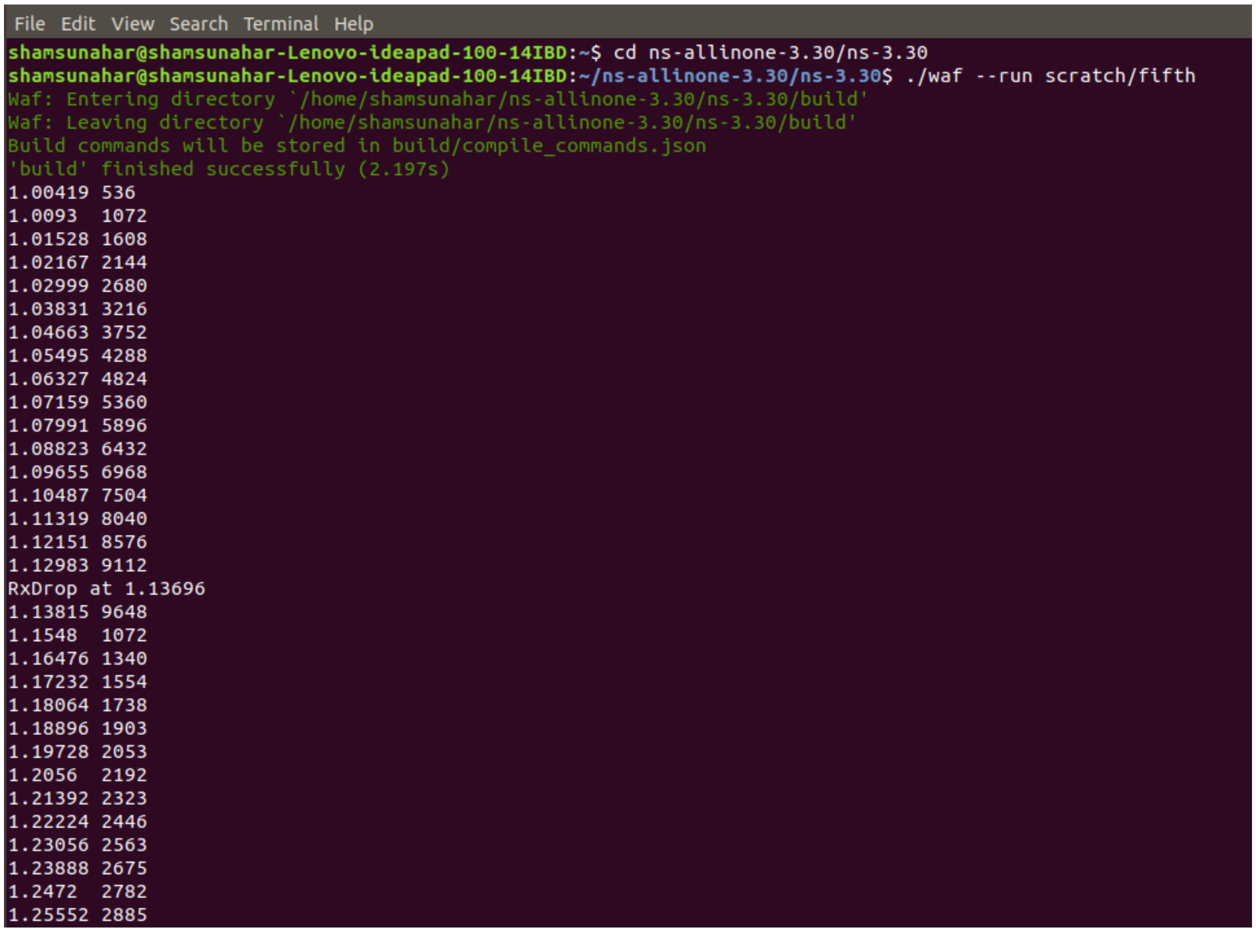
```
void

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)
```

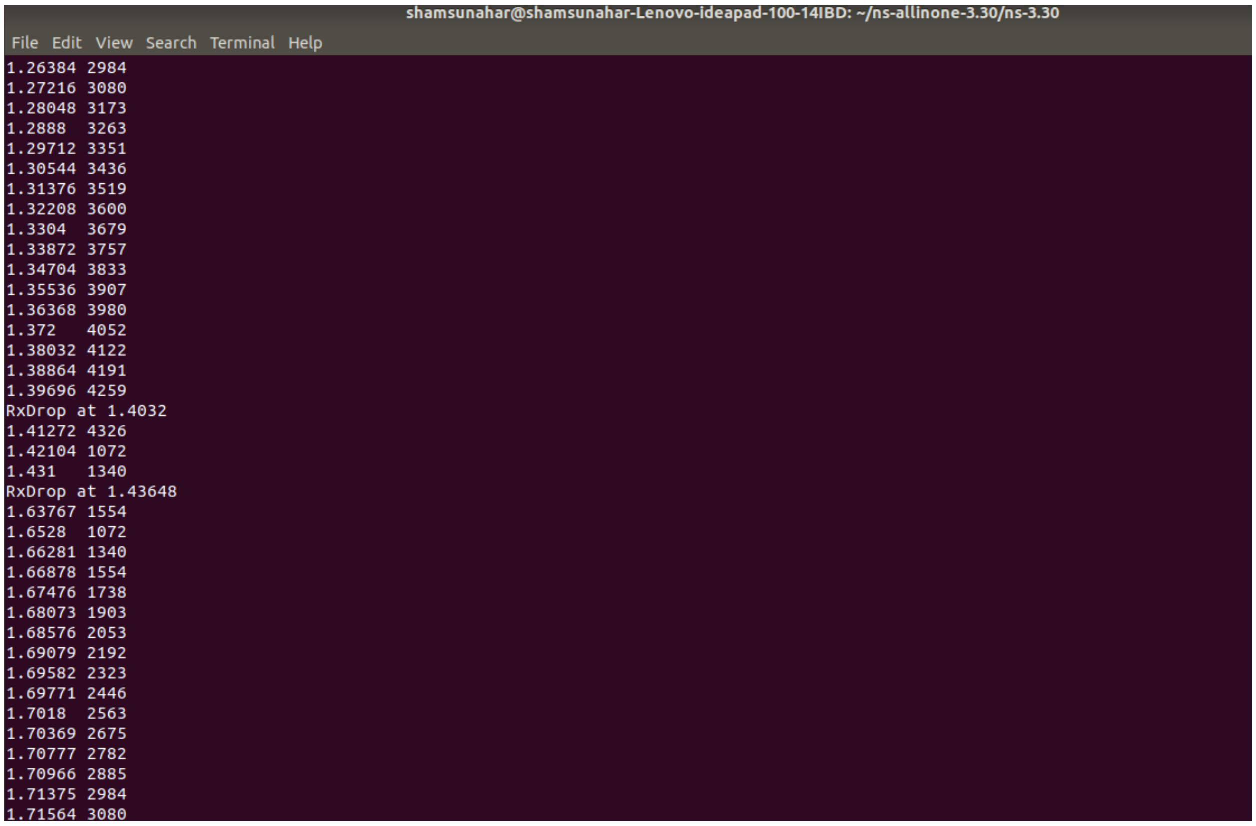
```
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}

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;

}
```

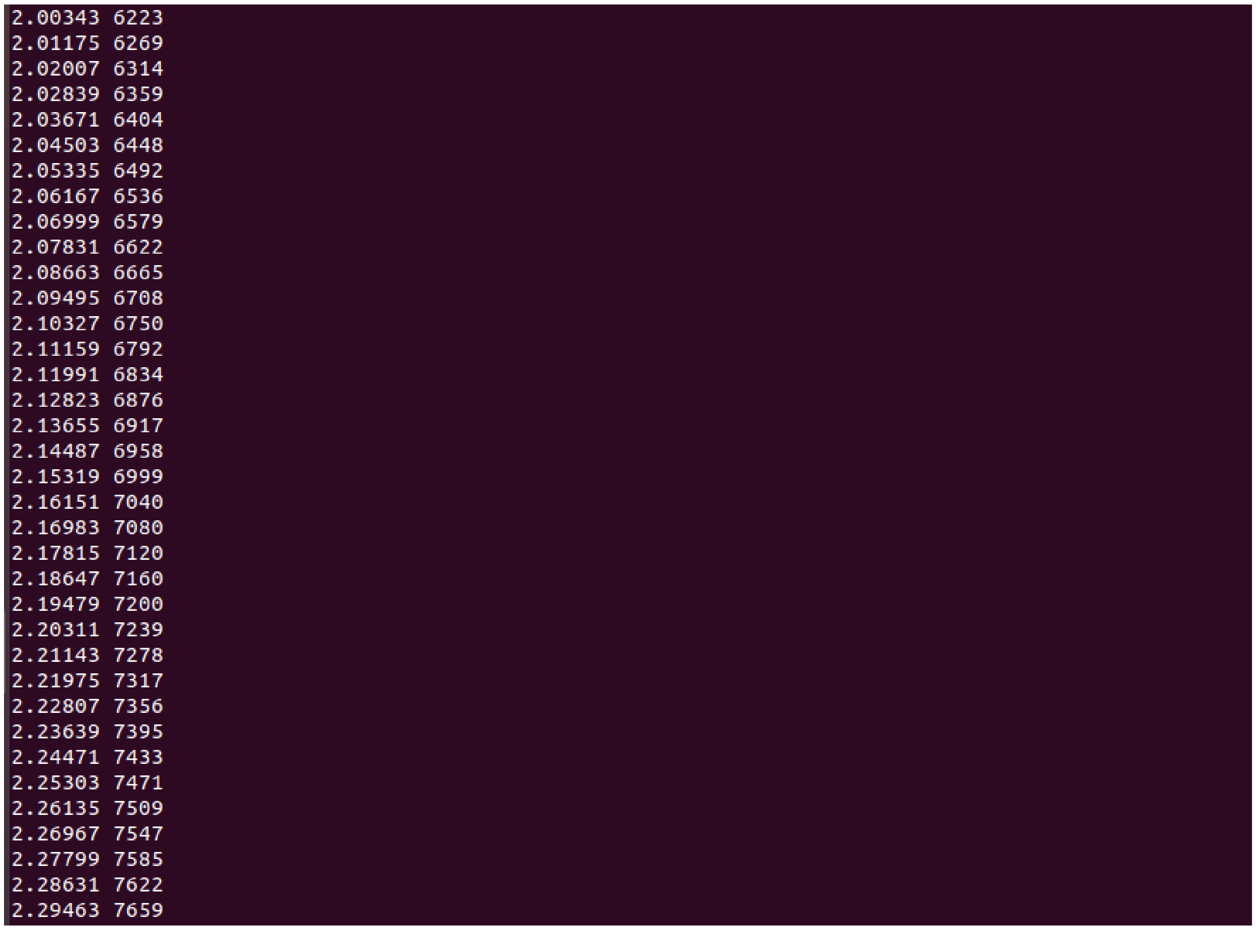
**OUTPUT :**



```
shamsunahar@shamsunahar-Lenovo-ideapad-100-14IBD:~$ cd ns-allinone-3.30/ns-3.30
shamsunahar@shamsunahar-Lenovo-ideapad-100-14IBD:~/ns-allinone-3.30/ns-3.30$ ./waf --run scratch/fifth
Waf: Entering directory `/home/shamsunahar/ns-allinone-3.30/ns-3.30/build'
Waf: Leaving directory `/home/shamsunahar/ns-allinone-3.30/ns-3.30/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (2.197s)
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
```


shamsunahar@shamsunahar-Lenovo-ideapad-100-14IBD: ~/ns-allinone-3.30/ns-3.30

```
1.26384 2984
1.27216 3080
1.28048 3173
1.2888  3263
1.29712 3351
1.30544 3436
1.31376 3519
1.32208 3600
1.3304  3679
1.33872 3757
1.34704 3833
1.35536 3907
1.36368 3980
1.372   4052
1.38032 4122
1.38864 4191
1.39696 4259
RxDrop at 1.4032
1.41272 4326
1.42104 1072
1.431   1340
RxDrop at 1.43648
1.63767 1554
1.6528  1072
1.66281 1340
1.66878 1554
1.67476 1738
1.68073 1903
1.68576 2053
1.69079 2192
1.69582 2323
1.69771 2446
1.7018  2563
1.70369 2675
1.70777 2782
1.70966 2885
1.71375 2984
1.71564 3080
```



```
2.00343 6223
2.01175 6269
2.02007 6314
2.02839 6359
2.03671 6404
2.04503 6448
2.05335 6492
2.06167 6536
2.06999 6579
2.07831 6622
2.08663 6665
2.09495 6708
2.10327 6750
2.11159 6792
2.11991 6834
2.12823 6876
2.13655 6917
2.14487 6958
2.15319 6999
2.16151 7040
2.16983 7080
2.17815 7120
2.18647 7160
2.19479 7200
2.20311 7239
2.21143 7278
2.21975 7317
2.22807 7356
2.23639 7395
2.24471 7433
2.25303 7471
2.26135 7509
2.26967 7547
2.27799 7585
2.28631 7622
2.29463 7659
```

**Conclusion:** TCP has automatic recovery from dropped packets, which it interprets as congestion on the network. A TCP protocol is a transport layer that provides a reliable and in-order delivery of data between two hosts. TCP provides the reliable and connection oriented connection oriented network. TCP provides different variants for example:- Tahoe, Reno and New Reno, all of which are available with ns-3.