Mawlana Bhashani Science and Technology University

Santosh, Tangail-1902.

Lab Report

Department of Information and Communication Technology

Experiment No: 02
Experiment Name: TCP Variants

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

Submitted By	Submitted To
Name: **Naznin Sultana** ID: IT-16036 Session: 2015-16 4th Year 2nd Semester Dept. of Information & Communication Technology, MBSTU.	Nazrul Islam Assistant Professor Dept. of Information & Communication Technology, MBSTU.

Name: **Naznin Sultana** **ID: IT-16036**

Experiment N0: 02

Name of Experiments : TCP Variants

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");

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);
    }
}
```

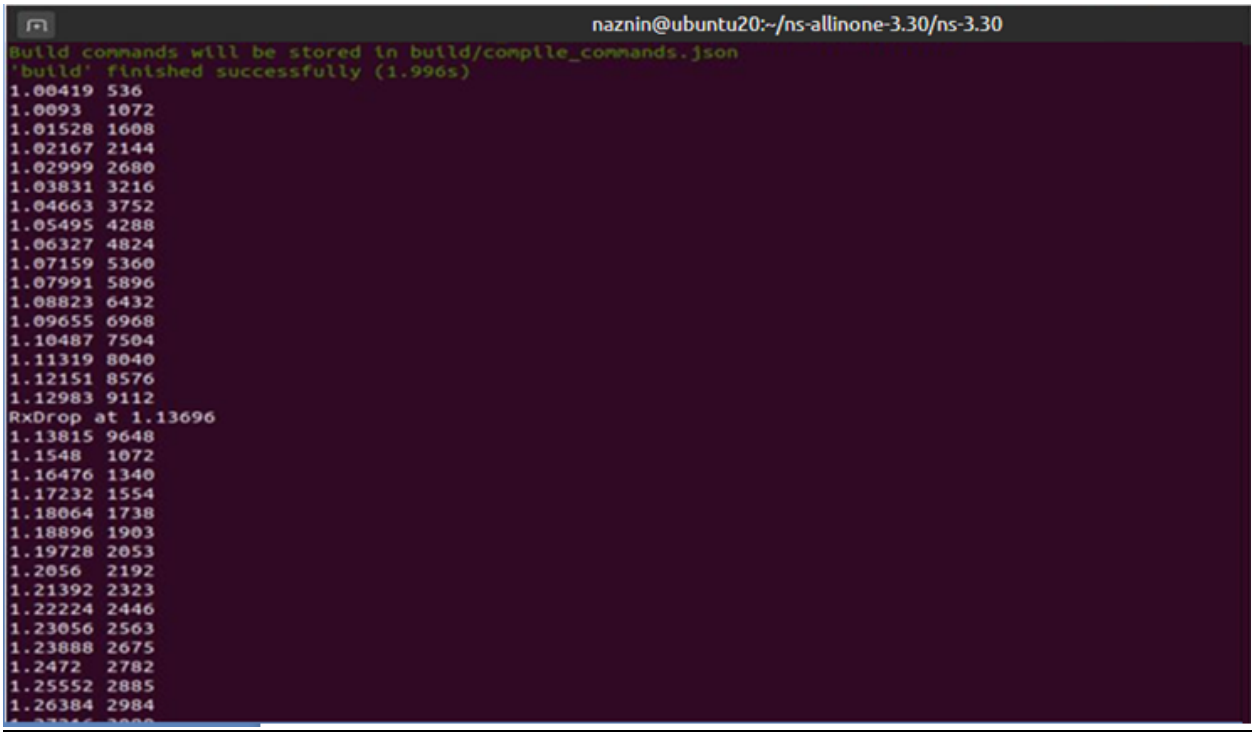
```
static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
  NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}

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;
```

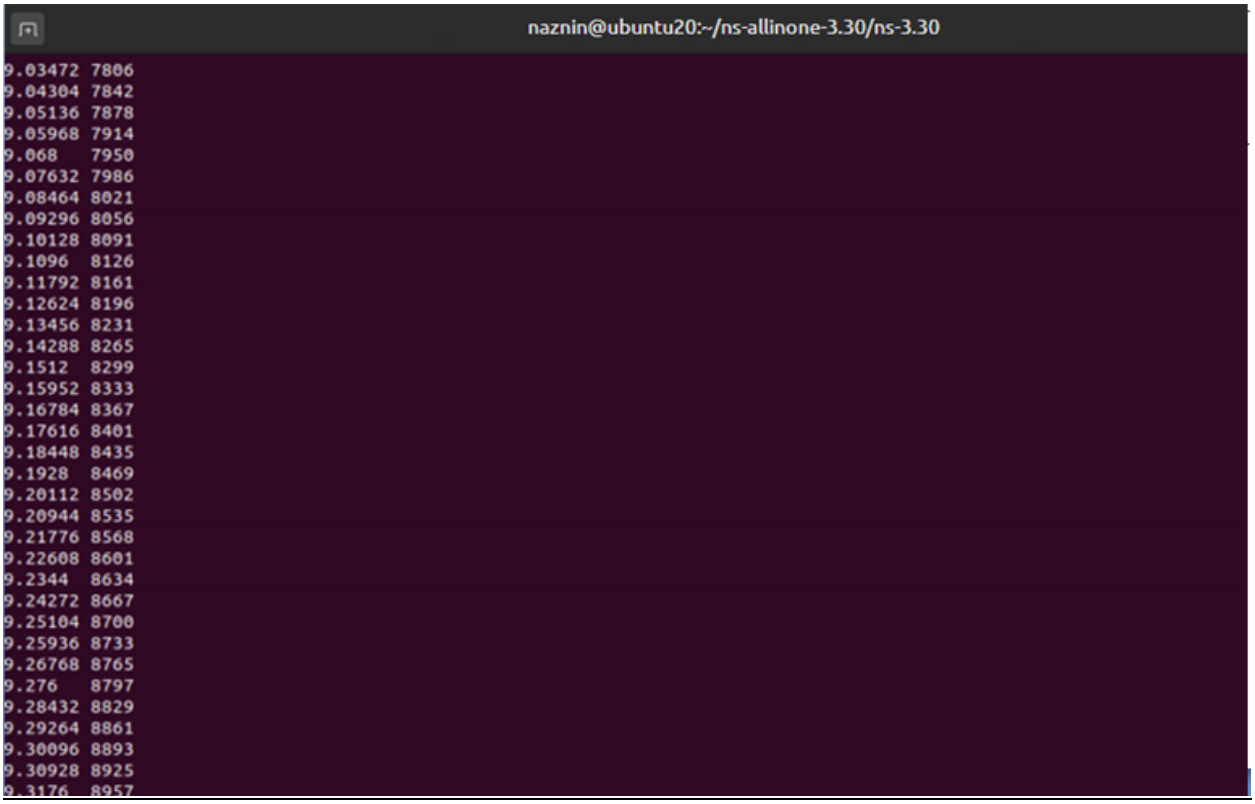
```
nodes.Create (2);

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:





Conclusion:

From this lab, we learned about TCP internals and the difference between each of the variants using NS-3 tracing mechanism. For this we first compile , execute and calculate the throughput and then compare the various TCP protocols in NS3 for wireless network. from the lab we've learnt how to create dumbbell topology, the process of installing TCP & UDP instance & got used to these.We've used the ns-3 tracing mechanism to record changes in congestion window size of the TCP instance over time & used gnuplot/matplotlib to visualise plots of cwnd vs time.