

## **Experiment N0:02**

### **Name of Experiments : 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 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 :

```
'build' finished successfully (13.860s)
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
```

2.49431 8502  
2.50263 8535  
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

9.01808 7733  
9.0264 7770  
9.03472 7806  
9.04304 7842  
9.05136 7878  
9.05968 7914  
9.068 7950  
9.07632 7986  
9.08464 8021  
9.09296 8056  
9.10128 8091  
9.1096 8126  
9.11792 8161  
9.12624 8196  
9.13456 8231  
9.14288 8265  
9.1512 8299  
9.15952 8333  
9.16784 8367  
9.17616 8401  
9.18448 8435  
9.1928 8469  
9.20112 8502  
9.20944 8535  
9.21776 8568  
9.22608 8601  
9.2344 8634  
9.24272 8667  
9.25104 8700  
9.25936 8733  
9.26768 8765  
9.276 8797  
9.28432 8829  
9.29264 8861  
9.30096 8893  
9.30928 8925  
9.3176 8957