

Lab no : 02

Lab name : 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");
```

```
//
```

```
=====
```

```
//
```

```
//      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));
```

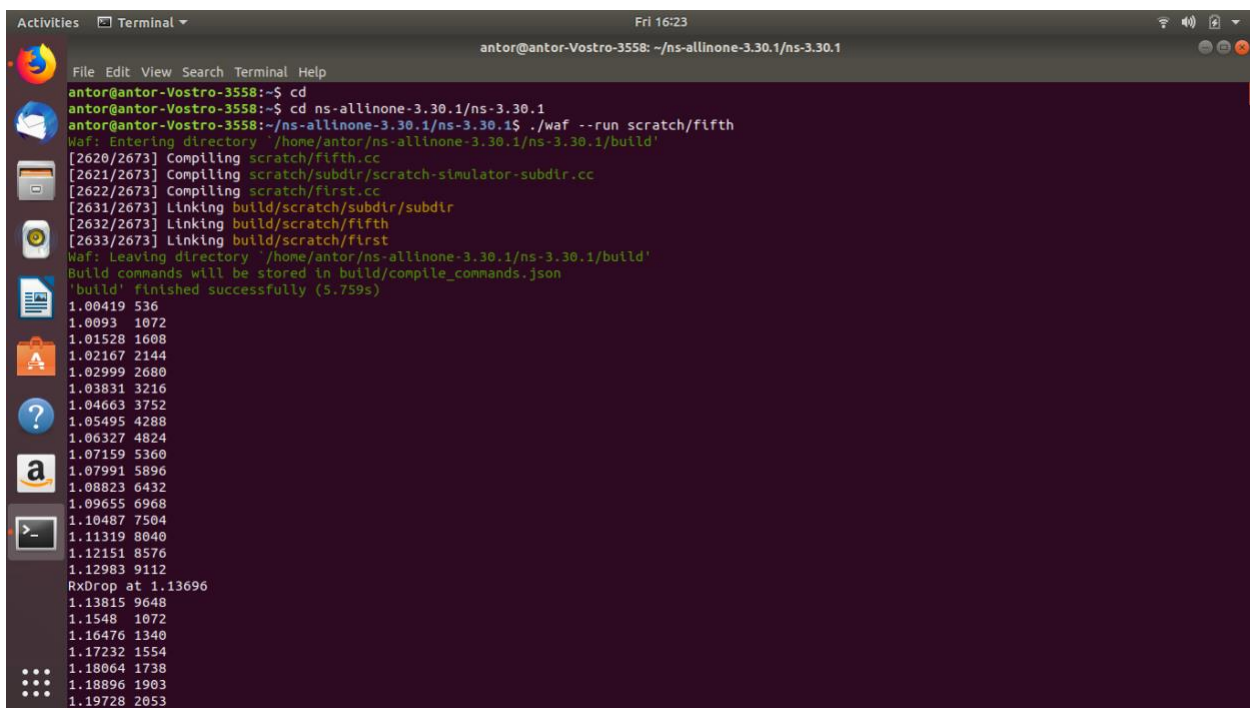
```
Simulator::Run ();
```

```
Simulator::Destroy ();
```

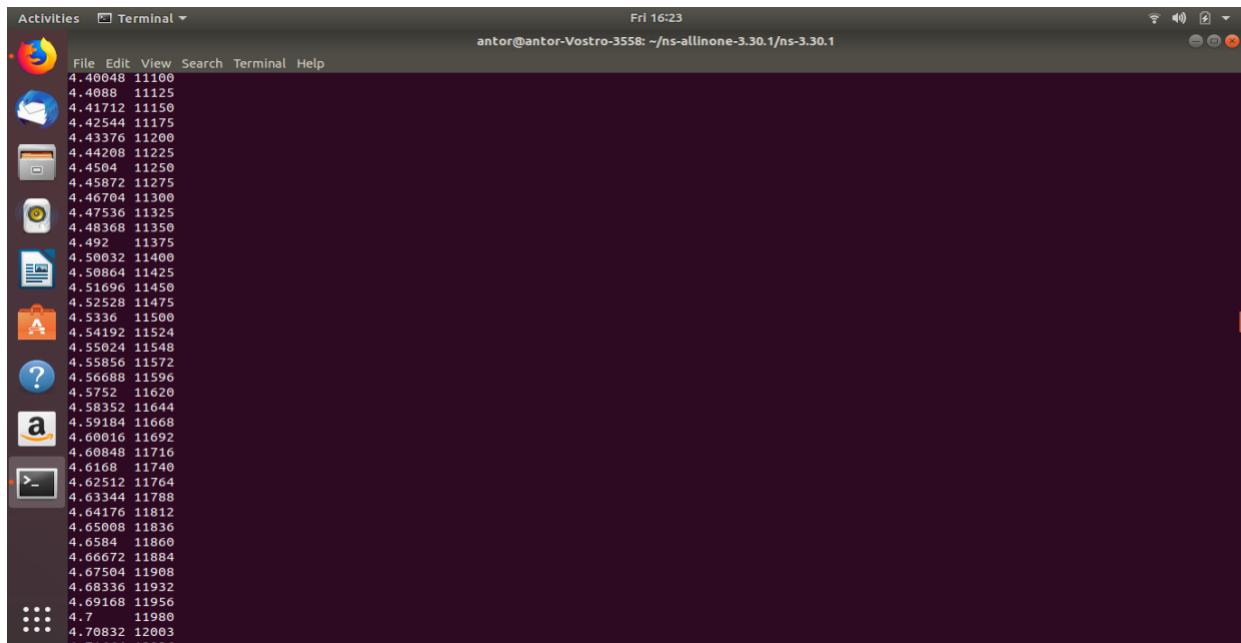
```
return 0;
```

```
}
```

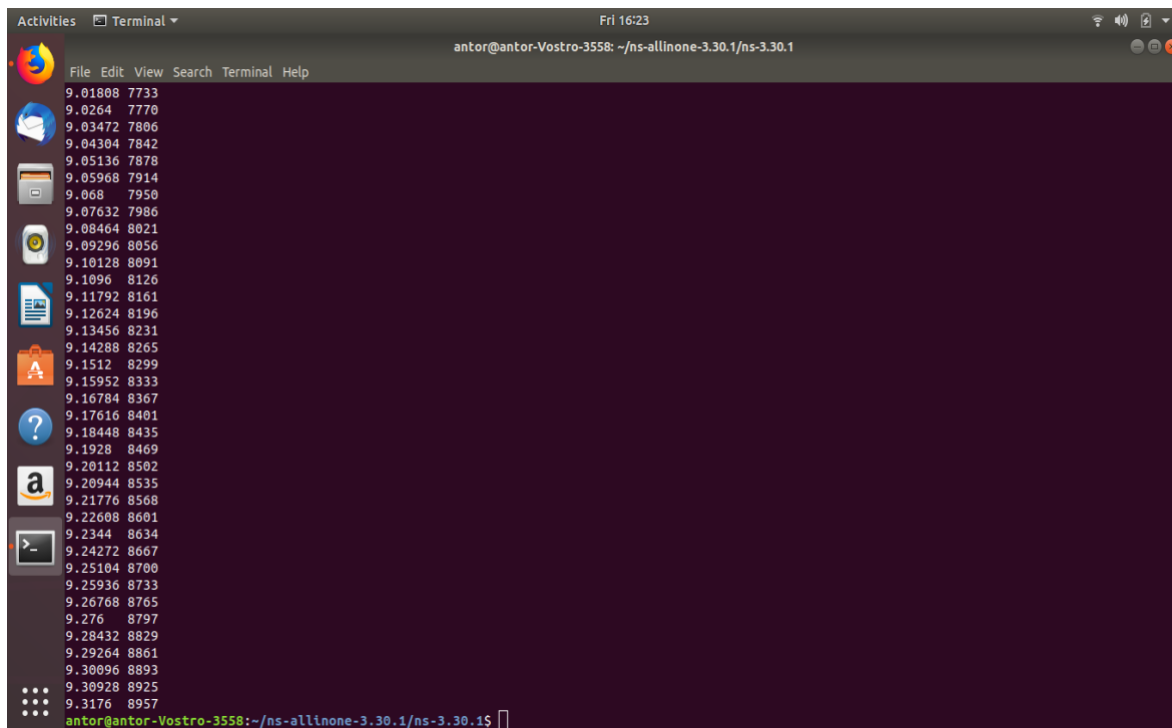
OUTPUT :



```
Activities  Terminal  Fri 16:23
antor@antor-Vostro-3558: ~/ns-allinone-3.30.1/ns-3.30.1
File Edit View Search Terminal Help
antor@antor-Vostro-3558:~$ cd
antor@antor-Vostro-3558:~$ cd ns-allinone-3.30.1/ns-3.30.1
antor@antor-Vostro-3558:~/ns-allinone-3.30.1/ns-3.30.1$ ./waf --run scratch/fifth
Waf: Entering directory '/home/antor/ns-allinone-3.30.1/ns-3.30.1/build'
[2620/2673] Compiling scratch/fifth.cc
[2621/2673] Compiling scratch/subdir/scratch-simulator-subdir.cc
[2622/2673] Compiling scratch/first.cc
[2631/2673] Linking build/scratch/subdir/subdir
[2632/2673] Linking build/scratch/fifth
[2633/2673] Linking build/scratch/first
Waf: Leaving directory '/home/antor/ns-allinone-3.30.1/ns-3.30.1/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (5.759s)
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
```



```
antor@antor-Vostro-3558: ~/ns-allinone-3.30.1/ns-3.30.1
4.40048 11100
4.4088 11125
4.41712 11150
4.42544 11175
4.43376 11200
4.44208 11225
4.4504 11250
4.45872 11275
4.46704 11300
4.47536 11325
4.48368 11350
4.492 11375
4.50032 11400
4.50864 11425
4.51696 11450
4.52528 11475
4.5336 11500
4.54192 11524
4.55024 11548
4.55856 11572
4.56688 11596
4.5752 11620
4.58352 11644
4.59184 11668
4.60016 11692
4.60848 11716
4.6168 11740
4.62512 11764
4.63344 11788
4.64176 11812
4.65008 11836
4.6584 11860
4.66672 11884
4.67504 11908
4.68336 11932
4.69168 11956
4.7 11980
4.70832 12003
```



```
antor@antor-Vostro-3558: ~/ns-allinone-3.30.1/ns-3.30.1
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
antor@antor-Vostro-3558:~/ns-allinone-3.30.1/ns-3.30.1$
```

Conclusion: By creating different networks in ns3 simulator, we could deeply analyze the behavior of the protocols with these TCP variants in the basis of the amount of packet drops in each case. The lesser the amount of drops the better the algorithm. This

paper implicitly analyses which TCP variant has lesser drop rates with which routing protocol .