CS342 Assignment 4 Report

Group 34

Aditya Trivedi 190101005
Atharva Vijay Varde 190101018
Shashwat Sharma 190123055

ns3 Version Used: 3.28.1
Drive link to .xml and .PCAP files :
https://iitgoffice-my.sharepoint.com/:f:/g/personal/atrivedi_iitg_ac_in/EhYxKWTVqnBIuUqc4o3ObisBM5OmS8fZiXkasW_UrufGQg?e=3nNlKg

File Structure :

- simulation.cc : file to be run in ns3
- scripts/
  - data/ : contains all scripts to generate data
  - graphs/ : contains all scripts to generate graphs
- data/ : contains all data generated by data scripts
- graphs/ : contains all graph images
- xml/ : contains the .xml files generated by ns3 simulation.cc

Simulation Setup :

Three nodes, Node 0, 1, 2 at locations 250*i, in a straight line.
Node 0 and Node 2 both have TCP traffic to Node 1 - started randomly within 1 to 5 seconds of starting the simulation.
TCP Westwood+ is used for TCP agents at n0 and n2, respectively.

Explanation of simulation.cc

Code for setting up legacy channel and physical layer

```
WifiMacHelper mac;
WifiHelper wifiHelper;
wifiHelper.SetStandard(WIFI_PHY_STANDARD_80211n_5GHZ);

// Set up Legacy Channel
YansWifiChannelHelper wifichannel;
wifichannel.SetPropagationDelay("ns3::Consta
ntSpeedPropagationDelayModel");
wifichannel.AddPropagationLoss("ns3::FriisPropagationLossModel");

// Setup Physical Layer
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.SetChannel (wifichannel.Create ());
wifiPhy.Set ("TxPowerStart", DoubleValue (100.0));
wifiPhy.Set ("TxPowerEnd", DoubleValue (100.0));
wifiPhy.Set ("TxPowerLevels", UintegerValue (1));
wifiPhy.Set ("TxGain", DoubleValue (0));
wifiPhy.Set ("RxGain", DoubleValue (0));
wifiPhy.Set ("RxNoiseFigure", DoubleValue (10));
wifiPhy.Set ("CcaMode1Threshold", DoubleValue (-79));
```

```
wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue (-79 + 3));
wifiPhy.SetErrorRateModel ("ns3::YansErrorRateModel");
wifiHelper.SetRemoteStationManager ("ns3::ConstantRateWifiManager","DataMode", StringValue
(phyRate),"ControlMode", StringValue ("HtMcs0"));
```

Setting up physical locations (250*i)

```
Ptr<ListPositionAllocator> position = CreateObject<ListPositionAllocator>
();
position->Add (Vector (5.0, 0.0, 0.0));
position->Add (Vector (5.0, 250.0, 0.0));
position->Add (Vector (5.0, 500.0, 0.0));
```

At each station, a TCP transmitter is installed

```
ipv4=apInterface.GetAddress(0);
inetaddress=Address(InetSocketAddress (ipv4,portno));
OnOffHelper server (protocol,inetaddress);
server.SetAttribute ("PacketSize", UintegerValue (payload));
server.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
server.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
server.SetAttribute ("DataRate", DataRateValue (DataRate (datarate)));
ApplicationContainer serverApp = server.Install (staWifiNodes);
```

Installing TCP receiver

```
uint16_t portno=9;
Ipv4Address ipv4=Ipv4Address::GetAny();
std::string protocol="ns3::TcpSocketFactory";
Address inetaddress=Address(InetSocketAddress(ipv4,portno));
PacketSinkHelper sinkHelper (protocol,inetaddress);
ApplicationContainer sinkApp = sinkHelper.Install (apWifiNode);
sink = StaticCast<PacketSink> (sinkApp.Get (0));
```

Then a random number is generated from 1 to 5, after which the TCP traffic starts :

```
int GenerateRandomNumber()
{
 std::srand(time(NULL));
 int x=(std::rand()%5);
 return x+1;
}
```

Starting server at a randomly generated time

```
int x= GenerateRandomNumber();
double x_double=(double)x*1.0;
serverApp.Start (Seconds (x_double));
```

But before starting the server, we must start the simulation and also specify when it should end.

```
serverApp.Start (Seconds (x_double));
Simulator::Stop (Seconds (simulationTime + x_double));
Simulator::Run ();
fm->SerializeToXmlFile("Westwood_256.xml",true,true);
Simulator::Destroy ();
```

Obtaining Data :

The sink monitor class available in ns3 only allows recording link-layer statistics such as RTS, CTS, ACK, etc. However the question also asks for transport layer statistics, hence we also need traces generated by ns3.

Code for flow monitor

```
FlowMonitorHelper fph;
Ptr<FlowMonitor> fm = fph.InstallAll();
```

To generate traces and obtain relevant data, we must first install “sources” on corresponding nodes/devices. Whenever we want the data, we initialise a “sink” and connect it to the relevant “source”.

ns3 classes expose relevant sources through the objects themselves, so we can utilise the SinkHelper class to create and connect sinks directly

Eg : Here we install a sink at apWifiNode

```
PacketSinkHelper sinkHelper (protocol,inetaddress);
ApplicationContainer sinkApp = sinkHelper.Install (apWifiNode);
sink = StaticCast<PacketSink> (sinkApp.Get (0));
```

Then we can obtain the required data as follows :

```
double averageThroughput =
((sink->GetTotalRx () * 8) / (1e6  * simulationTime));
```

PCAP is also generated by ns3, and analysed using python scripts in “scripts” folder. **NOT generated or analysed using Wireshark or any other PCAP tracing software**

```
    wifiPhy.SetPcapDataLinkType (WifiPhyHelper::DLT_IEEE802_11_RADIO);
    wifiPhy.EnablePcap ("AccessPoint", apDevice);
    wifiPhy.EnablePcap ("Station", staDevices);
```

We ran the simulation.cc 4 times, with RTS set as 0, 256, 512 and 1000.

3 PCAP files were generated :

1. AccessPoint - Node 1
2. Station 0 - Node 0
3. Station 1 - Node 2

Here the AccessPoint refers to the middle node which is receiving traffic from nodes on the endpoints.

To analyse the traffic, we use 3 scripts, 1 for each node, with minor differences. These scripts are present in the “scripts/data” folder. This will dump the relevant data in a .txt file.

These python scripts use the dpkt package, which can parse the ns3 PCAP file and give information for each package. The following code checks each package and determines if it is an ACK packet

```
for ts,packet in pcap:
      tap = dpkt.radiotap.Radiotap(packet)
      t_len = binascii.hexlify(packet[2:3])
      t_len = int(t_len, 16)
      ieee80211Frame = packet[t_len:]
      sequenceControl = packet[t_len + 22:t_len + 24]
      wlan = dpkt.ieee80211.IEEE80211(ieee80211Frame)

      if wlan.subtype == dpkt.ieee80211.C_ACK:
          frame_rts_total  = frame_rts_total + len(packet)
          frame_rts = frame_rts + 1
```

Later, we find the proportion of ACKs using frame_rts_total variable.

Similarly, we calculate RTS and CTS statistics.

Then, we use the data from .txt obtained from above, to generate graphs. These are in “scripts/graphs” folder. The graphs are attached below.

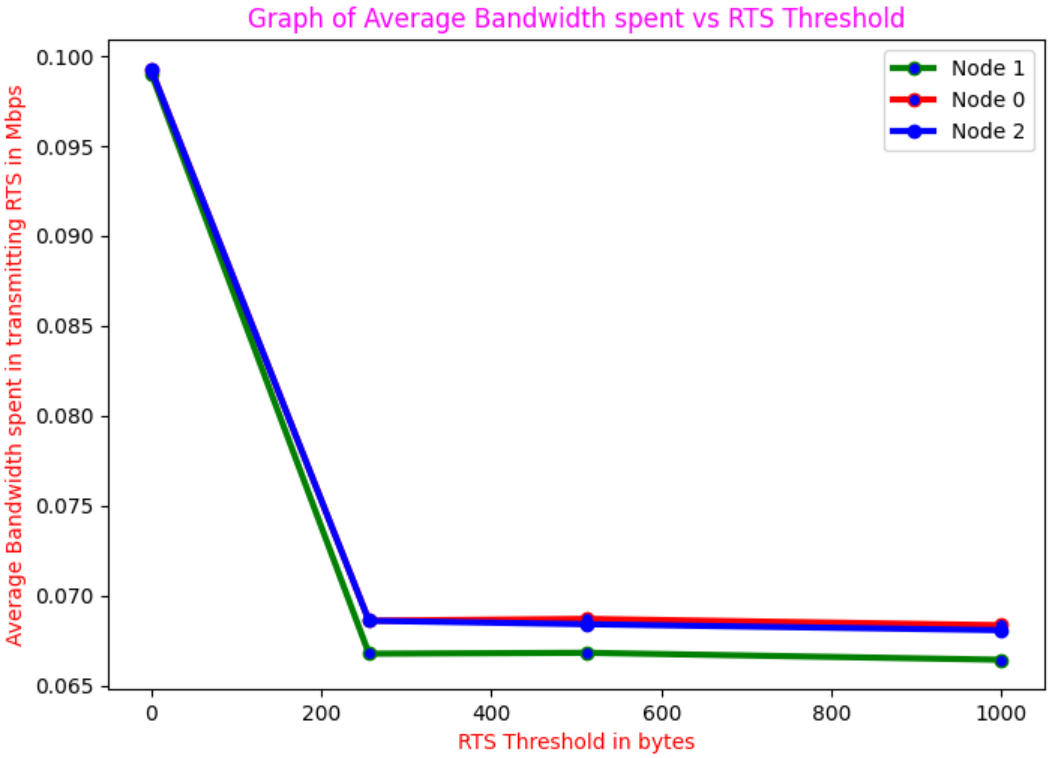
For analysing the throughput we use the flow monitor inbuilt parser, which is available in src/flow-monitor/examples/flowmon-parse-results.py

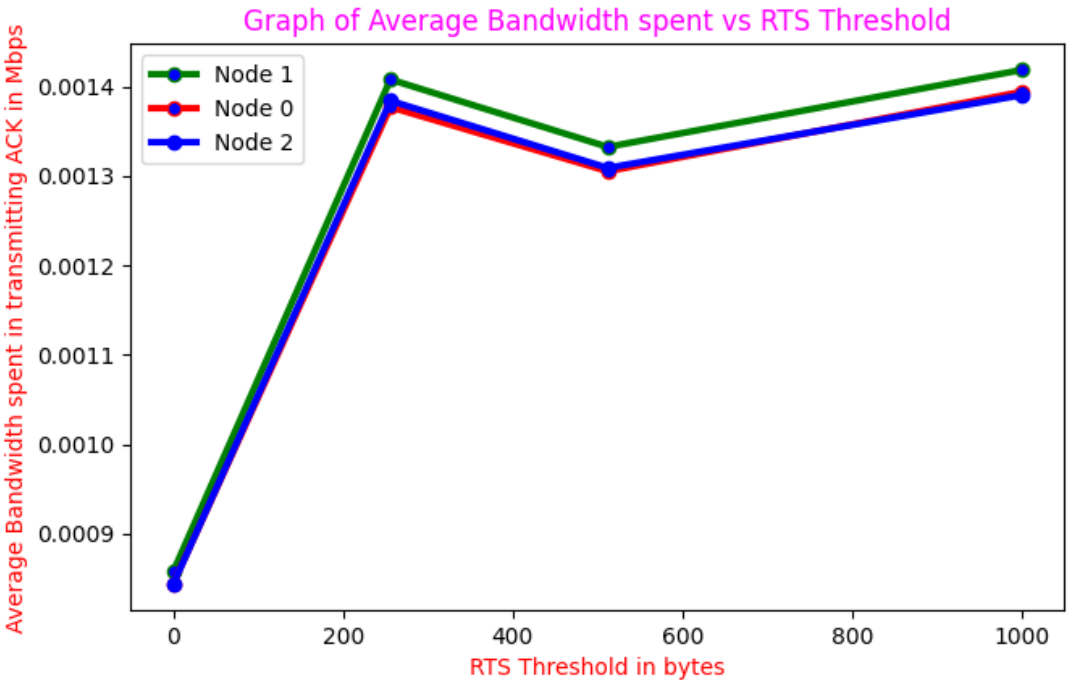
Keep all the xml files and this script in one directory, then run :

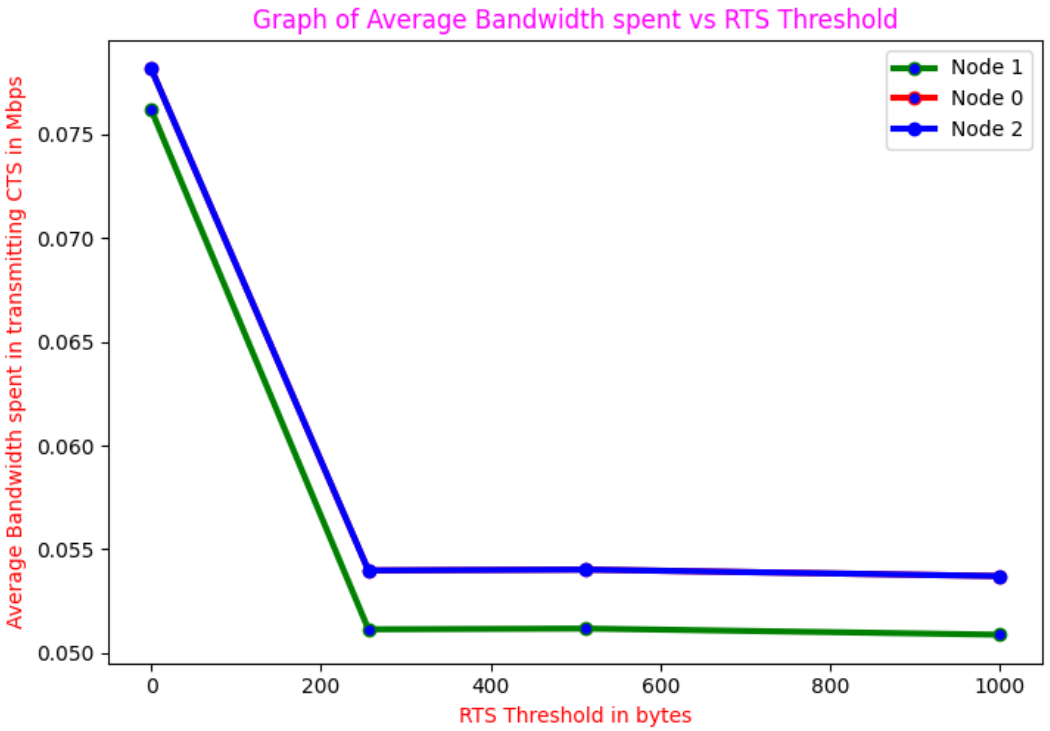
python flowmon-parse-results.py myxml.xml

Observations :

1. **Average bandwidth spent in transmitting RTS, CTS, and ACK.**

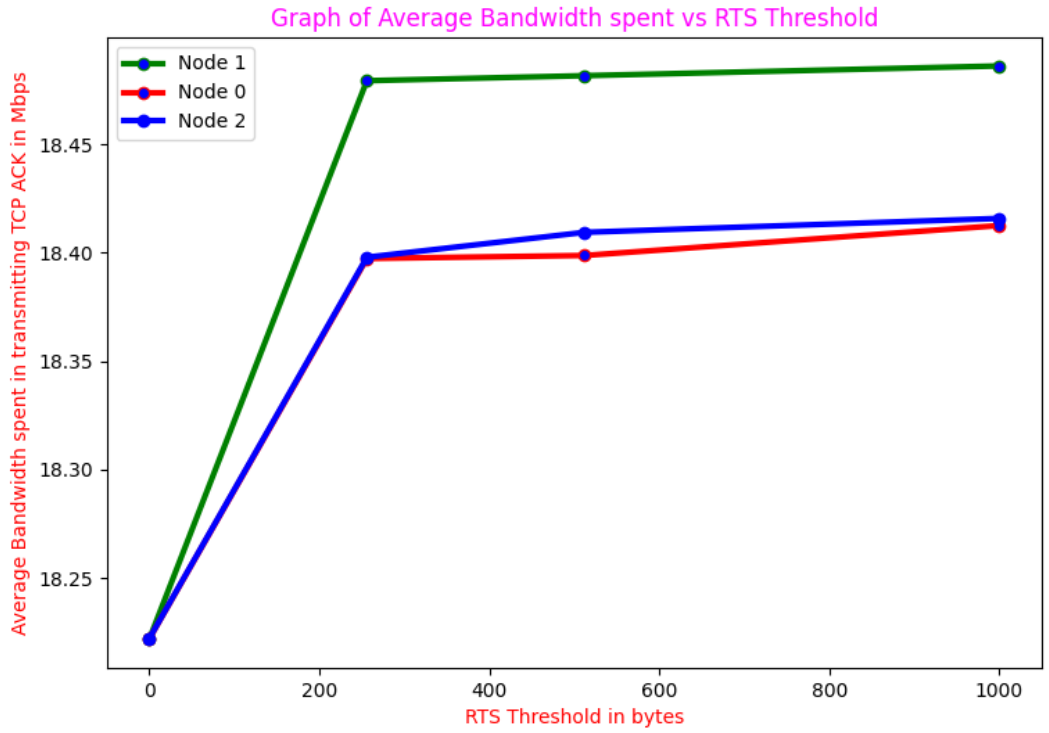


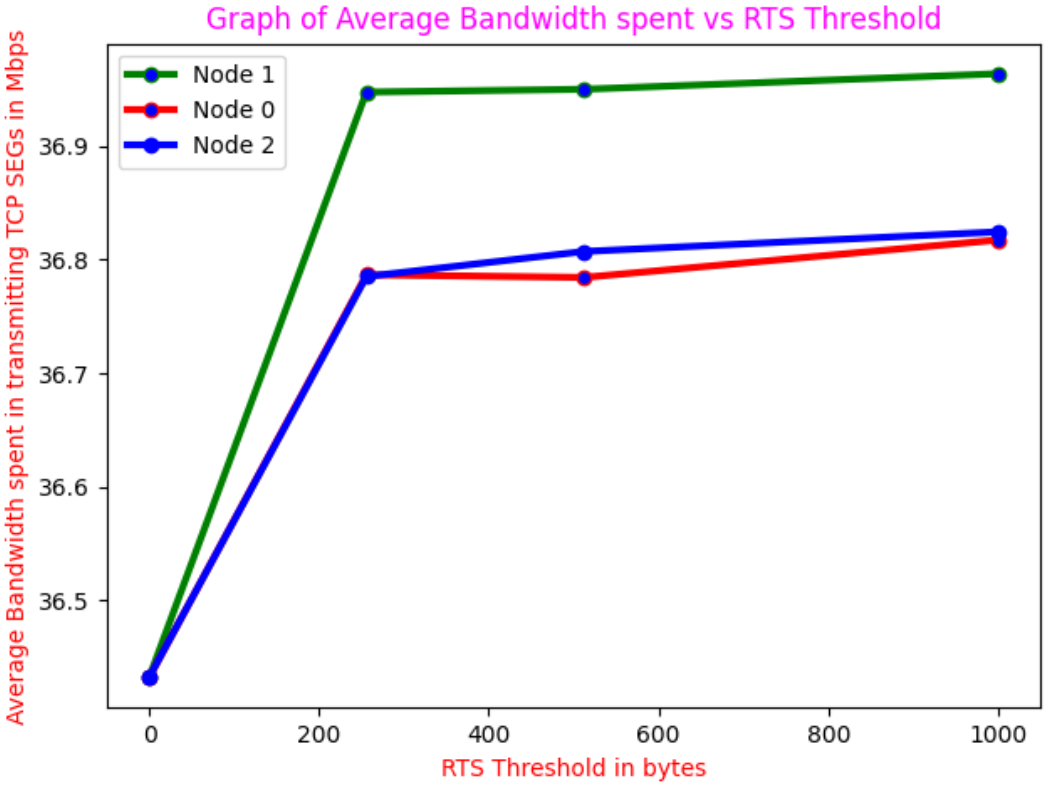




As we can see, Station 0 and Station 2 have very similar statistics as they are both performing identical functions.

2. Average bandwidth spent in transmitting TCP segments and TCP acks





3. Average bandwidth wasted due to collisions

We can see using data collected from the flow monitor (screenshots below) that there is 0.0% packet loss, which means that there were negligible collisions.

4. TCP Throughput at each node

Screenshots of flow monitor for each RTS :
The average throughputs seen at each node are shown flow-wise. These are the tx and rx bitrate (bits/second) values. The IP addresses of the nodes are as follows:
10.1.1.1: Node 1
10.1.1.2: Node 0
10.1.1.3: Node 3.

The bitrates for tx/rx are calculated as the total number of bytes transmitted divided by the time difference between the last tx/rx packet received and the first tx/rx packet received for each case.

RTS = 0

```
aditya@adityapc:~/networks/A4/ns3simulation/xmls$ python2 flowmon-parse-results.py  Westwood_0.xml
Reading XML file . done.
FlowID: 1 (TCP 10.1.1.2/49153 --> 10.1.1.1/9)
        TX bitrate: 27266.61 kbit/s
        RX bitrate: 27253.71 kbit/s
        Mean Delay: 24.59 ms
        Packet Loss Ratio: 0.00 %
FlowID: 2 (TCP 10.1.1.3/49153 --> 10.1.1.1/9)
        TX bitrate: 27439.58 kbit/s
        RX bitrate: 27419.79 kbit/s
        Mean Delay: 24.24 ms
        Packet Loss Ratio: 0.00 %
FlowID: 3 (TCP 10.1.1.1/9 --> 10.1.1.2/49153)
        TX bitrate: 673.59 kbit/s
        RX bitrate: 673.55 kbit/s
        Mean Delay: 12.41 ms
        Packet Loss Ratio: 0.00 %
FlowID: 4 (TCP 10.1.1.1/9 --> 10.1.1.3/49153)
        TX bitrate: 677.69 kbit/s
        RX bitrate: 678.02 kbit/s
        Mean Delay: 12.39 ms
        Packet Loss Ratio: 0.00 %
```

RTS = 256

```
aditya@adityapc:~/networks/A4/ns3simulation/xmls$ python2 flowmon-parse-results.py  Westwood_256.xml
Reading XML file . done.
FlowID: 1 (TCP 10.1.1.2/49153 --> 10.1.1.1/9)
        TX bitrate: 27527.84 kbit/s
        RX bitrate: 27522.68 kbit/s
        Mean Delay: 22.56 ms
        Packet Loss Ratio: 0.00 %
FlowID: 2 (TCP 10.1.1.3/49153 --> 10.1.1.1/9)
        TX bitrate: 27545.79 kbit/s
        RX bitrate: 27526.52 kbit/s
        Mean Delay: 22.28 ms
        Packet Loss Ratio: 0.00 %
FlowID: 3 (TCP 10.1.1.1/9 --> 10.1.1.2/49153)
        TX bitrate: 680.23 kbit/s
        RX bitrate: 680.07 kbit/s
        Mean Delay: 13.64 ms
        Packet Loss Ratio: 0.00 %
FlowID: 4 (TCP 10.1.1.1/9 --> 10.1.1.3/49153)
        TX bitrate: 680.32 kbit/s
        RX bitrate: 680.66 kbit/s
        Mean Delay: 13.80 ms
        Packet Loss Ratio: 0.00 %
```

RTS = 512

```
aditya@adityapc:~/networks/A4/ns3simulation/xmls$ python2 flowmon-parse-results.py  Westwood_512.xml
Reading XML file . done.
FlowID: 1 (TCP 10.1.1.2/49153 --> 10.1.1.1/9)
        TX bitrate: 26961.35 kbit/s
        RX bitrate: 26946.82 kbit/s
        Mean Delay: 23.26 ms
        Packet Loss Ratio: 0.00 %
FlowID: 2 (TCP 10.1.1.3/49153 --> 10.1.1.1/9)
        TX bitrate: 28127.70 kbit/s
        RX bitrate: 28112.11 kbit/s
        Mean Delay: 21.89 ms
        Packet Loss Ratio: 0.00 %
FlowID: 3 (TCP 10.1.1.1/9 --> 10.1.1.2/49153)
        TX bitrate: 666.00 kbit/s
        RX bitrate: 665.99 kbit/s
        Mean Delay: 13.82 ms
        Packet Loss Ratio: 0.00 %
FlowID: 4 (TCP 10.1.1.1/9 --> 10.1.1.3/49153)
        TX bitrate: 694.80 kbit/s
        RX bitrate: 694.96 kbit/s
        Mean Delay: 13.40 ms
        Packet Loss Ratio: 0.00 %
```

RTS = 1000

```
aditya@adityapc:~/networks/A4/ns3simulation/xmls$ python2 flowmon-parse-results.py  Westwood_1000.xml
Reading XML file . done.
FlowID: 1 (TCP 10.1.1.2/49153 --> 10.1.1.1/9)
        TX bitrate: 27130.33 kbit/s
        RX bitrate: 27114.19 kbit/s
        Mean Delay: 23.07 ms
        Packet Loss Ratio: 0.00 %
FlowID: 2 (TCP 10.1.1.3/49153 --> 10.1.1.1/9)
        TX bitrate: 27978.07 kbit/s
        RX bitrate: 27967.74 kbit/s
        Mean Delay: 22.12 ms
        Packet Loss Ratio: 0.00 %
FlowID: 3 (TCP 10.1.1.1/9 --> 10.1.1.2/49153)
        TX bitrate: 670.13 kbit/s
        RX bitrate: 670.17 kbit/s
        Mean Delay: 13.73 ms
        Packet Loss Ratio: 0.00 %
FlowID: 4 (TCP 10.1.1.1/9 --> 10.1.1.3/49153)
        TX bitrate: 691.23 kbit/s
        RX bitrate: 691.26 kbit/s
        Mean Delay: 13.40 ms
        Packet Loss Ratio: 0.00 %
```