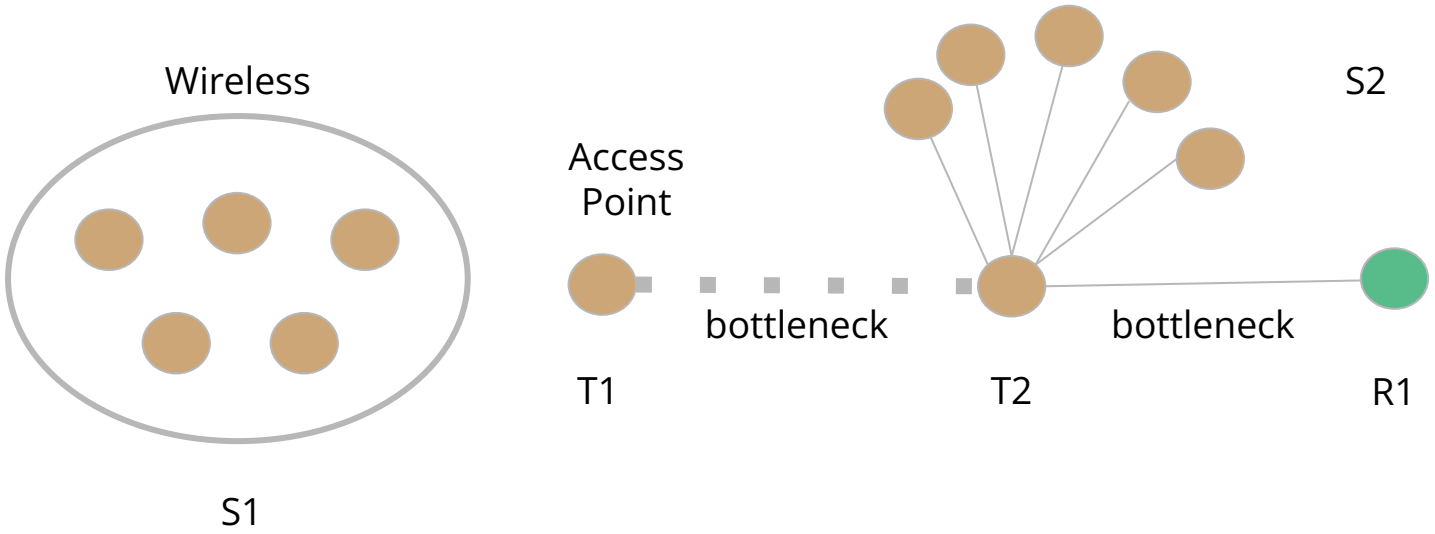
NS3 Project Update 2

Swift: Delay is Simple and Effective for Congestion Control in the Datacenter

Student ID : 1705037

Topology for Datacenters

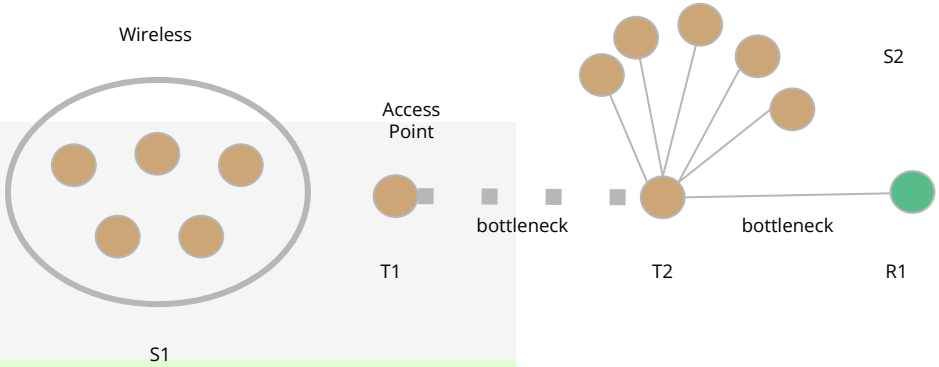


- S1 and S2 each have 5 senders sending to receiver R1 (10 flows in total)
- This sets up two bottlenecks:
  - T1 -> T2 interface (5 senders)
  - T2 -> R1 (5+5 senders)

Setting Congestion Control Algorithm - DCTCP

```
int main (int argc, char *argv[])
{
  std::string tcpTypeId = "TcpDctcp";
  Config::SetDefault ("ns3::TcpL4Protocol::SocketType", StringValue ("ns3::" + tcpTypeId));
```

Building the Topology

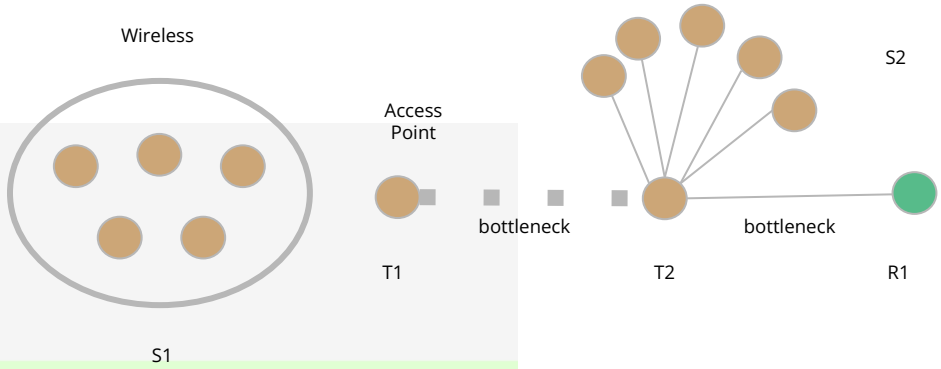



```
NodeContainer S1, S2;
Ptr<Node> T1 = CreateObject<Node> ();
Ptr<Node> T2 = CreateObject<Node> ();
Ptr<Node> R1 = CreateObject<Node> ();
S1.Create (5);
S2.Create (5);
```

```
PointToPointHelper pointToPointSR;
pointToPointSR.SetDeviceAttribute ("DataRate", StringValue (bottleneck_datarate));
pointToPointSR.SetChannelAttribute ("Delay", StringValue ("10us"));

PointToPointHelper pointToPointT;
pointToPointT.SetDeviceAttribute ("DataRate", StringValue ("1Gbps"));
pointToPointT.SetChannelAttribute ("Delay", StringValue ("10us"));

std::vector<NetDeviceContainer> S1T1;
S1T1.reserve (5);

std::vector<NetDeviceContainer> S2T2;
S2T2.reserve (5);
```

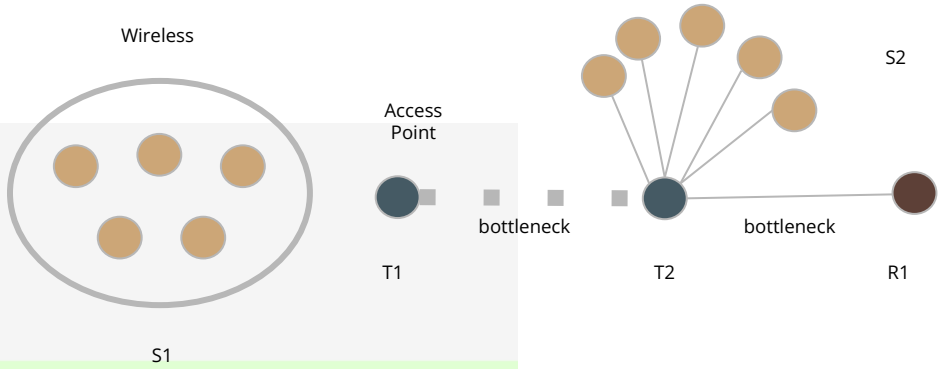
Creating Nodes

```
NodeContainer S1, S2;
Ptr<Node> T1 = CreateObject<Node> ();
Ptr<Node> T2 = CreateObject<Node> ();
Ptr<Node> R1 = CreateObject<Node> ();
S1.Create (5);
S2.Create (5);

PointToPointHelper pointToPointSR;
pointToPointSR.SetDeviceAttribute ("DataRate", StringValue (bottleneck_datarate));
pointToPointSR.SetChannelAttribute ("Delay", StringValue ("10us"));

PointToPointHelper pointToPointT;
pointToPointT.SetDeviceAttribute ("DataRate", StringValue ("1Gbps"));
pointToPointT.SetChannelAttribute ("Delay", StringValue ("10us"));

std::vector<NetDeviceContainer> S1T1;
S1T1.reserve (5);

std::vector<NetDeviceContainer> S2T2;
S2T2.reserve (5);
```



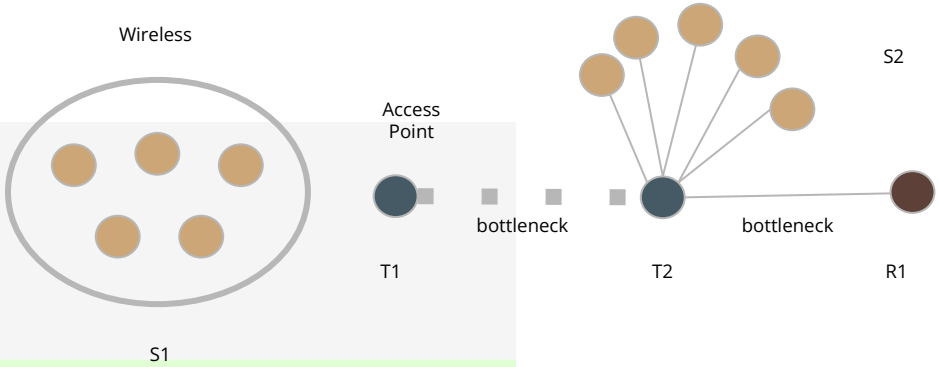
Creating Nodes

```
NodeContainer S1, S2;
Ptr<Node> T1 = CreateObject<Node> ();
Ptr<Node> T2 = CreateObject<Node> ();
Ptr<Node> R1 = CreateObject<Node> ();
S1.Create (5);
S2.Create (5);

PointToPointHelper pointToPointSR;
pointToPointSR.SetDeviceAttribute ("DataRate", StringValue (bottleneck_datarate));
pointToPointSR.SetChannelAttribute ("Delay", StringValue ("10us"));

PointToPointHelper pointToPointT;
pointToPointT.SetDeviceAttribute ("DataRate", StringValue ("1Gbps"));
pointToPointT.SetChannelAttribute ("Delay", StringValue ("10us"));

std::vector<NetDeviceContainer> S1T1;
S1T1.reserve (5);

std::vector<NetDeviceContainer> S2T2;
S2T2.reserve (5);
```



Creating Nodes

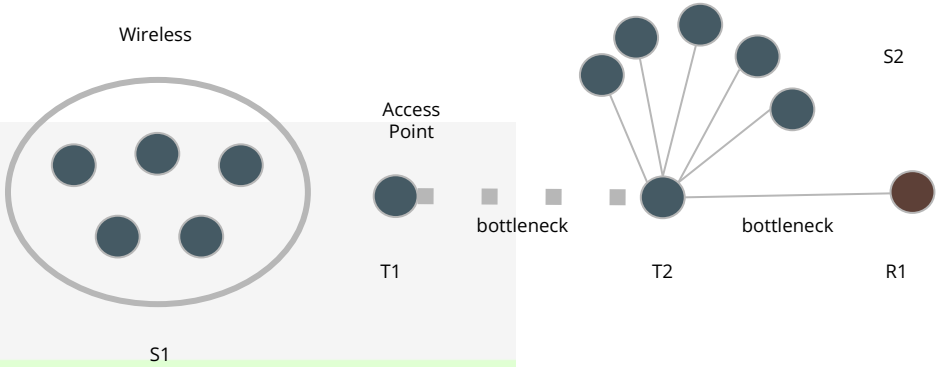
```
NodeContainer S1, S2;
Ptr<Node> T1 = CreateObject<Node> ();
Ptr<Node> T2 = CreateObject<Node> ();
Ptr<Node> R1 = CreateObject<Node> ();
S1.Create (5);
S2.Create (5);
```




```
PointToPointHelper pointToPointSR;
pointToPointSR.SetDeviceAttribute ("DataRate", StringValue (bottleneck_datarate));
pointToPointSR.SetChannelAttribute ("Delay", StringValue ("10us"));

PointToPointHelper pointToPointT;
pointToPointT.SetDeviceAttribute ("DataRate", StringValue ("1Gbps"));
pointToPointT.SetChannelAttribute ("Delay", StringValue ("10us"));

std::vector<NetDeviceContainer> S1T1;
S1T1.reserve (5);

std::vector<NetDeviceContainer> S2T2;
S2T2.reserve (5);
```

Creating Nodes

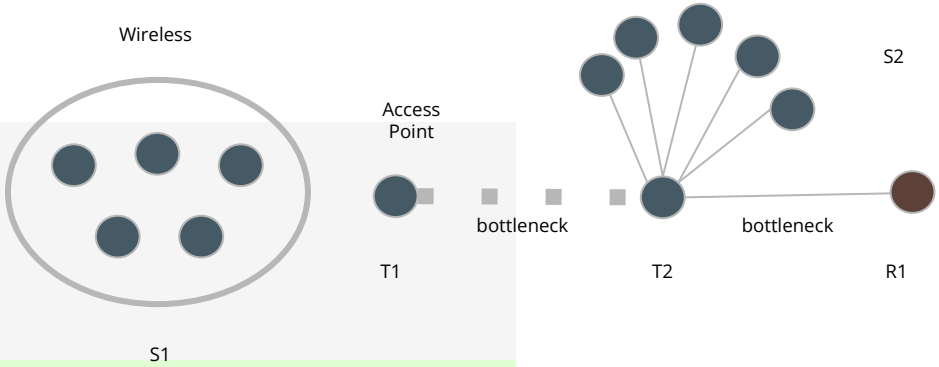
```
NodeContainer S1, S2;
Ptr<Node> T1 = CreateObject<Node> ();
Ptr<Node> T2 = CreateObject<Node> ();
Ptr<Node> R1 = CreateObject<Node> ();
S1.Create (5);
S2.Create (5);
```




```
PointToPointHelper pointToPointSR;
pointToPointSR.SetDeviceAttribute ("DataRate", StringValue (bottleneck_datarate));
pointToPointSR.SetChannelAttribute ("Delay", StringValue ("10us"));

PointToPointHelper pointToPointT;
pointToPointT.SetDeviceAttribute ("DataRate", StringValue ("1Gbps"));
pointToPointT.SetChannelAttribute ("Delay", StringValue ("10us"));

std::vector<NetDeviceContainer> S1T1;
S1T1.reserve (5);

std::vector<NetDeviceContainer> S2T2;
S2T2.reserve (5);
```

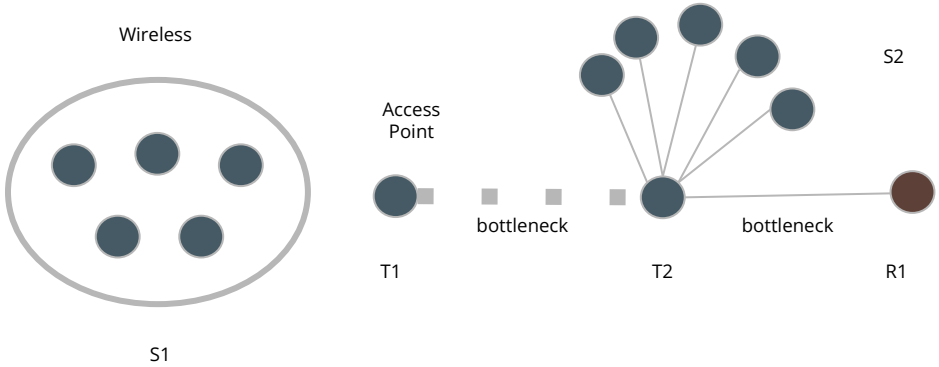
NetDeviceContainer

```
NodeContainer S1, S2;
Ptr<Node> T1 = CreateObject<Node> ();
Ptr<Node> T2 = CreateObject<Node> ();
Ptr<Node> R1 = CreateObject<Node> ();
S1.Create (5);
S2.Create (5);
```



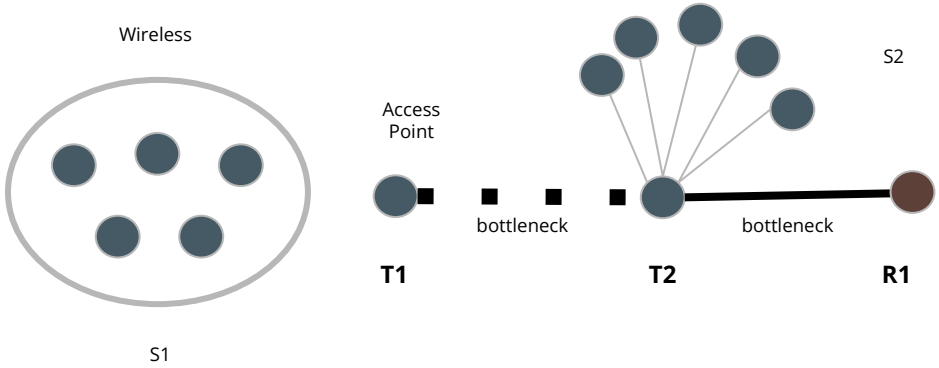

```
PointToPointHelper pointToPointSR;
pointToPointSR.SetDeviceAttribute ("DataRate", StringValue (bottleneck_datarate));
pointToPointSR.SetChannelAttribute ("Delay", StringValue ("10us"));

PointToPointHelper pointToPointT;
pointToPointT.SetDeviceAttribute ("DataRate", StringValue ("1Gbps"));
pointToPointT.SetChannelAttribute ("Delay", StringValue ("10us"));

std::vector<NetDeviceContainer> S1T1;
S1T1.reserve (5);

std::vector<NetDeviceContainer> S2T2;
S2T2.reserve (5);
```

Point to Point Links




```
NetDeviceContainer T1T2 = pointToPointT.Install (T1, T2);
NetDeviceContainer R1T2 = pointToPointSR.Install (R1, T2);
```

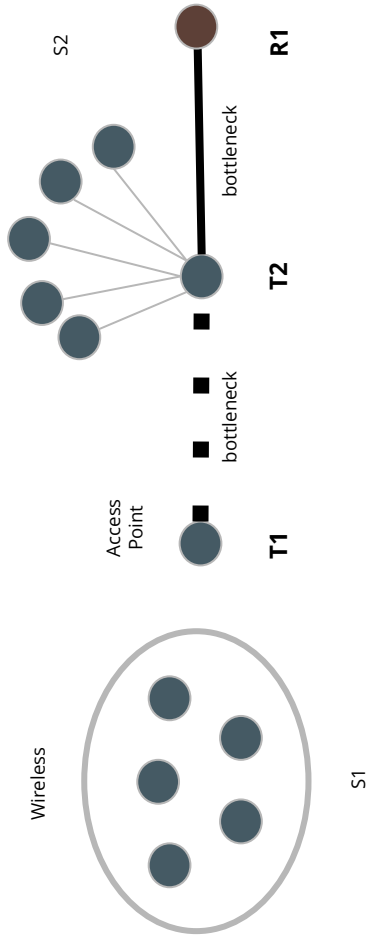
Point to Point Links




```
NetDeviceContainer T1T2 = pointToPointT.Install (T1, T2);
NetDeviceContainer R1T2 = pointToPointSR.Install (R1, T2);
```

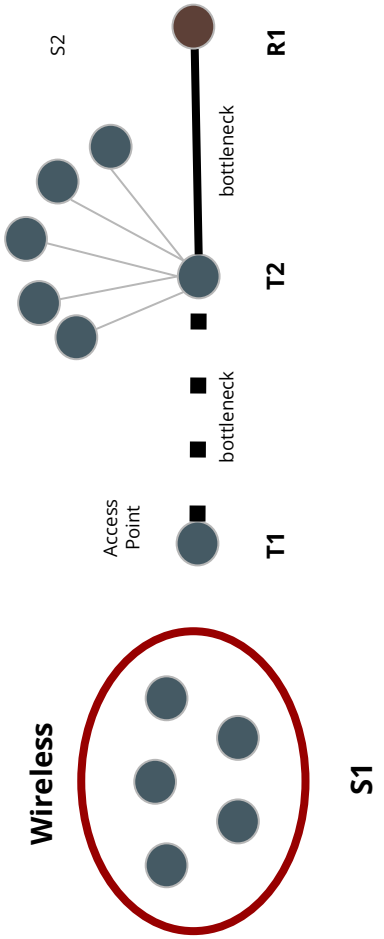
Wireless Nodes

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());
phy.SetErrorRateModel ("ns3::YansErrorRateModel");
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
for (std::size_t i = 0; i < 5; i++)
  {
    Ptr<Node> n = S1.Get (i);
    S1T1.push_back (wifi.Install (phy, mac, n));
  }
  mac.SetType ("ns3::ApWifiMac",
               "Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, T1);
```



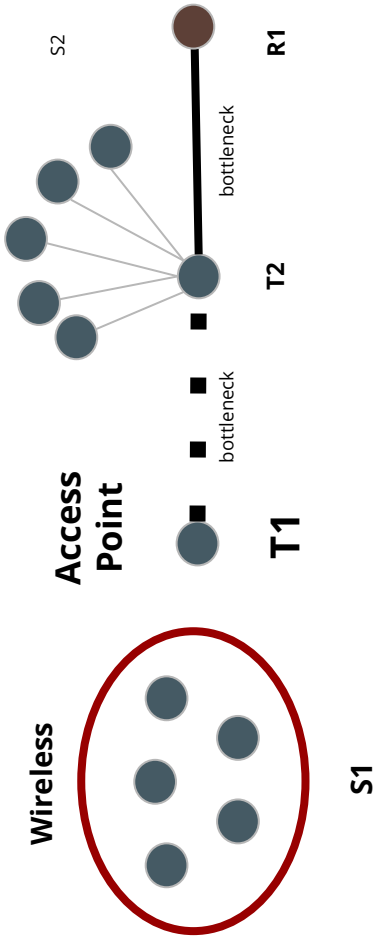
Wireless Nodes

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());
phy.SetErrorRateModel ("ns3::YansErrorRateModel");
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
for (std::size_t i = 0; i < 5; i++)
  {
    Ptr<Node> n = S1.Get (i);
    S1T1.push_back (wifi.Install (phy, mac, n));
  }
  mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, T1);
```



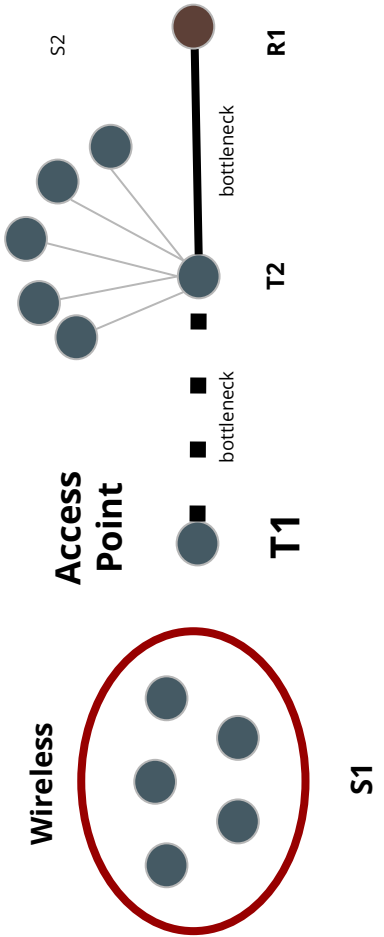
Wireless Nodes

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());
phy.SetErrorRateModel ("ns3::YansErrorRateModel");
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
for (std::size_t i = 0; i < 5; i++)
  {
    Ptr<Node> n = S1.Get (i);
    S1T1.push_back (wifi.Install (phy, mac, n));
  }
  mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, T1);
```

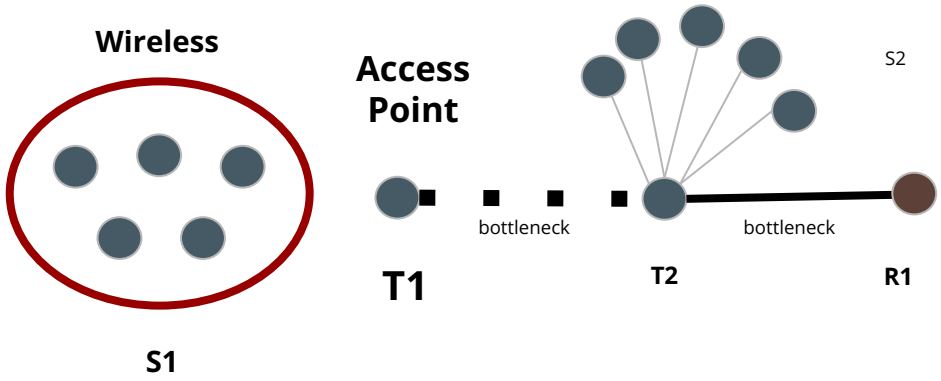


Wireless Nodes

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());
phy.SetErrorRateModel ("ns3::YansErrorRateModel");
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

WifiMacHelper mac;
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
for (std::size_t i = 0; i < 5; i++)
  {
    Ptr<Node> n = S1.Get (i);
    S1T1.push_back (wifi.Install (phy, mac, n));
  }
  mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, T1);
```



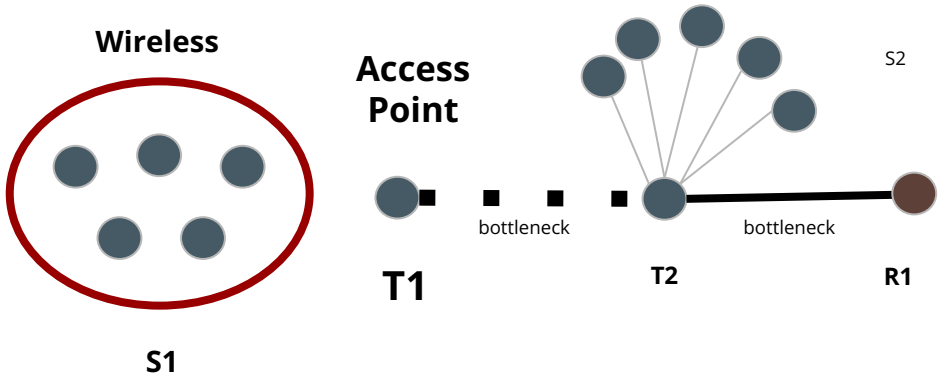
Mobility Model




```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (S1); //Wireless static nodes
mobility.Install (T1); //Apnode
```

Static

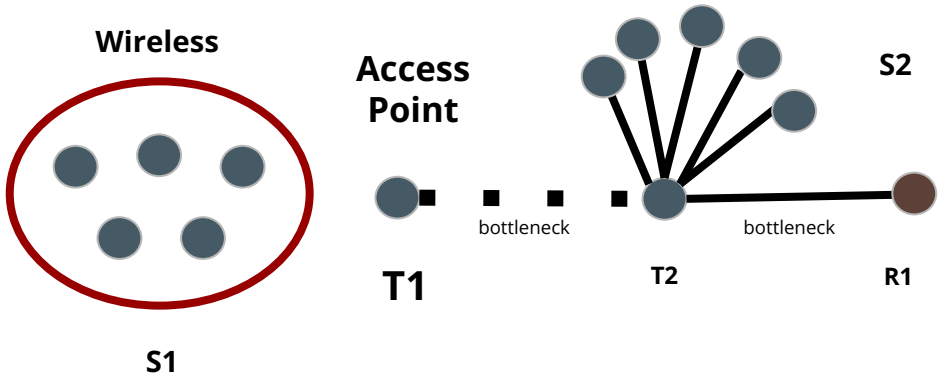
Wired Nodes




```
for (std::size_t i = 0; i < 5; i++)
  {
    Ptr<Node> n = S2.Get (i);
    S2T2.push_back (pointToPointSR.Install (n, T2));
  }
```

```
InternetStackHelper stack;
  stack.InstallAll ();
```

Wired Nodes




```
for (std::size_t i = 0; i < 5; i++)
  {
    Ptr<Node> n = S2.Get (i);
    S2T2.push_back (pointToPointSR.Install (n, T2));
  }
```

```
InternetStackHelper stack;
  stack.InstallAll ();
```

Introducing Error To Drop Packets

```
Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
em->SetAttribute ("ErrorRate", DoubleValue (0.00001));

//Adding Error to wired nodes
for (std::size_t i = 0; i < 7; i++)
{
    std::cout << "\t" << i << std::endl;
    All[i].Get (0)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
    All[i].Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));

}
```

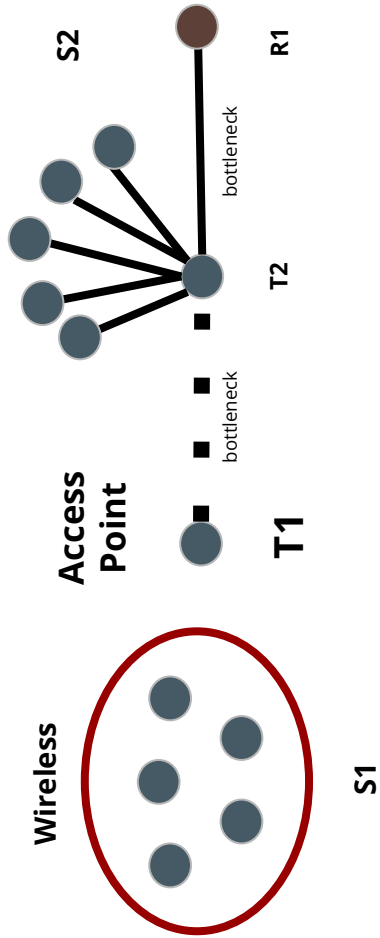
```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy;
phy.SetChannel (channel.Create ());
phy.SetErrorRateModel ("ns3::YansErrorRateModel");
```

Traffic Control for DCTCP

```
TrafficControlHelper tchRed10;
// MinTh = 50, MaxTh = 150 recommended in ACM SIGCOMM 2010 DCTCP Paper
// This yields a target (MinTh) queue depth of 60us at 10 Gb/s
tchRed10.SetRootQueueDisc ("ns3::RedQueueDisc",
                           "LinkBandwidth", StringValue ("1Gbps"),
                           "LinkDelay", StringValue ("10us"),
                           "MinTh", DoubleValue (50),
                           "MaxTh", DoubleValue (150));
QueueDiscContainer queueDiscs1 = tchRed10.Install (T1T2);

TrafficControlHelper tchRed1;
// MinTh = 20, MaxTh = 60 recommended in ACM SIGCOMM 2010 DCTCP Paper
// This yields a target queue depth of 250us at 1 Gb/s
tchRed1.SetRootQueueDisc ("ns3::RedQueueDisc",
                          "LinkBandwidth", StringValue ("500Mbps"),
                          "LinkDelay", StringValue ("10us"),
                          "MinTh", DoubleValue (20),
                          "MaxTh", DoubleValue (60));
QueueDiscContainer queueDiscs2 = tchRed1.Install (R1T2.Get (1));

for (std::size_t i = 0; i < 5; i++)
{
    tchRed1.Install (S1T1[i]);
}
for (std::size_t i = 0; i < 5; i++)
{
    tchRed1.Install (S2T2[i].Get (1));
}
```

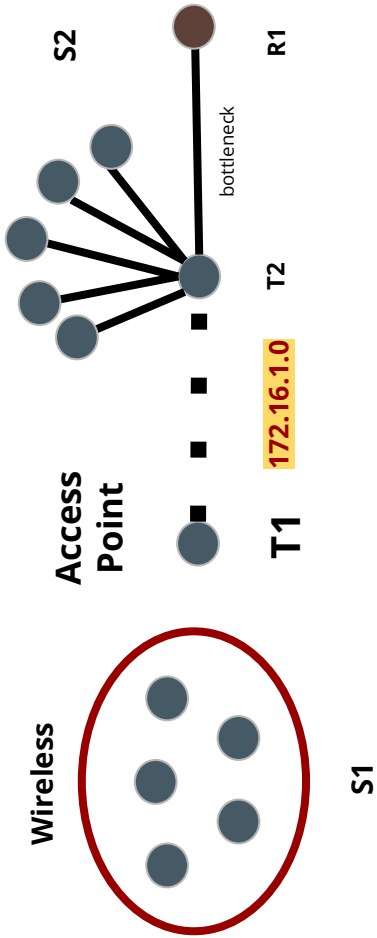
Creating Networks

```
Ipv4AddressHelper address;
std::vector<Ipv4InterfaceContainer> ipS1T1;
ipS1T1.reserve (5);
std::vector<Ipv4InterfaceContainer> ipS2T2;
ipS2T2.reserve (5);
address.SetBase ("172.16.1.0", "255.255.255.0");
Ipv4InterfaceContainer ipT1T2 = address.Assign (T1T2);
address.SetBase ("192.168.0.0", "255.255.255.0");
Ipv4InterfaceContainer ipR1T2 = address.Assign (R1T2);

address.SetBase ("10.1.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS1T1.push_back (address.Assign (S1T1[i]));
}
address.Assign(apDevices);

address.SetBase ("10.2.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS2T2.push_back (address.Assign (S2T2[i]));
    address.NewNetwork ();
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```



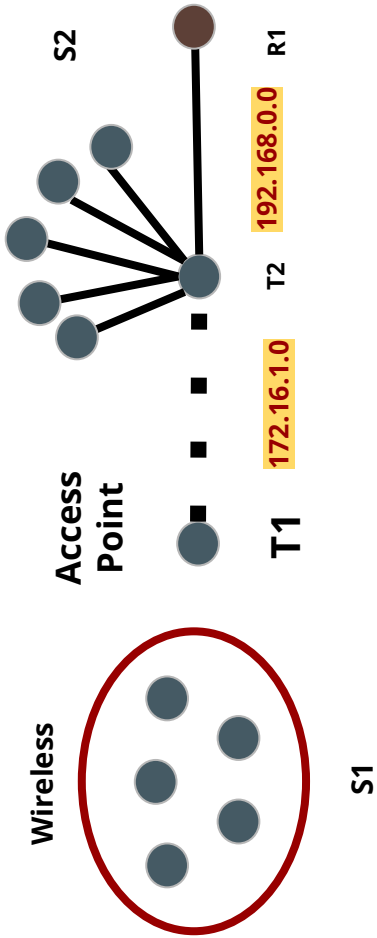
Creating Networks

```
Ipv4AddressHelper address;
std::vector<Ipv4InterfaceContainer> ipS1T1;
ipS1T1.reserve (5);
std::vector<Ipv4InterfaceContainer> ipS2T2;
ipS2T2.reserve (5);
address.SetBase ("172.16.1.0", "255.255.255.0");
Ipv4InterfaceContainer ipT1T2 = address.Assign (T1T2);
address.SetBase ("192.168.0.0", "255.255.255.0");
Ipv4InterfaceContainer ipR1T2 = address.Assign (R1T2);

address.SetBase ("10.1.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS1T1.push_back (address.Assign (S1T1[i]));
}
address.Assign(apDevices);

address.SetBase ("10.2.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS2T2.push_back (address.Assign (S2T2[i]));
    address.NewNetwork ();
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```



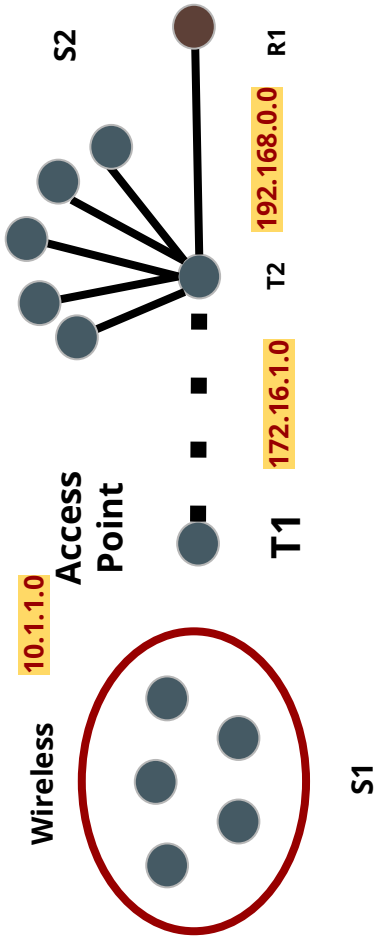
Creating Networks

```
Ipv4AddressHelper address;
std::vector<Ipv4InterfaceContainer> ipS1T1;
ipS1T1.reserve (5);
std::vector<Ipv4InterfaceContainer> ipS2T2;
ipS2T2.reserve (5);
address.SetBase ("172.16.1.0", "255.255.255.0");
Ipv4InterfaceContainer ipT1T2 = address.Assign (T1T2);
address.SetBase ("192.168.0.0", "255.255.255.0");
Ipv4InterfaceContainer ipR1T2 = address.Assign (R1T2);

address.SetBase ("10.1.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS1T1.push_back (address.Assign (S1T1[i]));
}
address.Assign(apDevices);

address.SetBase ("10.2.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS2T2.push_back (address.Assign (S2T2[i]));
    address.NewNetwork ();
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```



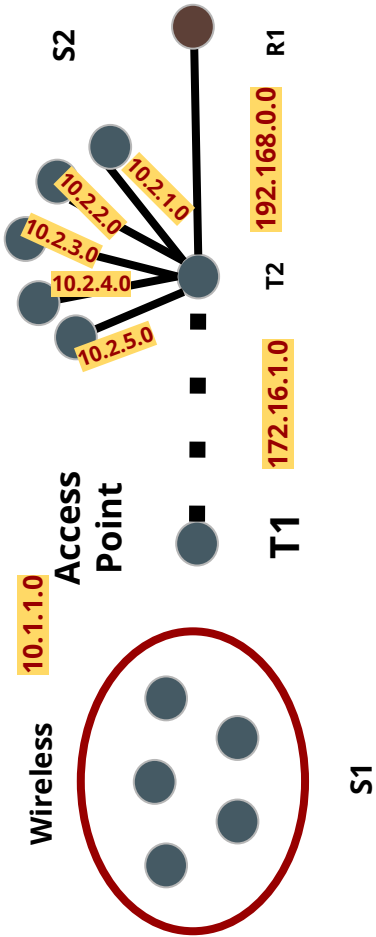
Creating Networks

```
Ipv4AddressHelper address;
std::vector<Ipv4InterfaceContainer> ipS1T1;
ipS1T1.reserve (5);
std::vector<Ipv4InterfaceContainer> ipS2T2;
ipS2T2.reserve (5);
address.SetBase ("172.16.1.0", "255.255.255.0");
Ipv4InterfaceContainer ipT1T2 = address.Assign (T1T2);
address.SetBase ("192.168.0.0", "255.255.255.0");
Ipv4InterfaceContainer ipR1T2 = address.Assign (R1T2);

address.SetBase ("10.1.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS1T1.push_back (address.Assign (S1T1[i]));
}
address.Assign(apDevices);

address.SetBase ("10.2.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS2T2.push_back (address.Assign (S2T2[i]));
    address.NewNetwork ();
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```



Creating Networks

```
Ipv4AddressHelper address;
std::vector<Ipv4InterfaceContainer> ipS1T1;
ipS1T1.reserve (5);
std::vector<Ipv4InterfaceContainer> ipS2T2;
ipS2T2.reserve (5);
address.SetBase ("172.16.1.0", "255.255.255.0");
Ipv4InterfaceContainer ipT1T2 = address.Assign (T1T2);
address.SetBase ("192.168.0.0", "255.255.255.0");
Ipv4InterfaceContainer ipR1T2 = address.Assign (R1T2);

address.SetBase ("10.1.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS1T1.push_back (address.Assign (S1T1[i]));
}
address.Assign(apDevices);

address.SetBase ("10.2.1.0", "255.255.255.0");
for (std::size_t i = 0; i < 5; i++)
{
    ipS2T2.push_back (address.Assign (S2T2[i]));
    address.NewNetwork ();
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```



Creating Multiple Flows

```
// Each sender in S1 and S2 sends to R1
std::vector<Ptr<PacketSink> > s1r1Sinks;
std::vector<Ptr<PacketSink> > s2r1Sinks;
s1r1Sinks.reserve (5);
s2r1Sinks.reserve (5);

//5 flows S1->R1 and 5 flows S2->R1
for (std::size_t i = 0; i < 10; i++)
  {
    uint16_t port = 50000 + i;
    Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
    PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", sinkLocalAddress);
    ApplicationContainer sinkApp = sinkHelper.Install (R1);
    Ptr<PacketSink> packetSink = sinkApp.Get (0)->GetObject<PacketSink> ();
    if (i < 5)
      {
        s1r1Sinks.push_back (packetSink);
      }
    else
      {
        s2r1Sinks.push_back (packetSink);
      }
    sinkApp.Start (startTime);
    sinkApp.Stop (stopTime);
  //....
  }
```

1

Creating Multiple Flows

```
// Each sender in S1 and S2 sends to R1
std::vector<Ptr<PacketSink> > s1r1Sinks;
std::vector<Ptr<PacketSink> > s2r1Sinks;
s1r1Sinks.reserve (5);
s2r1Sinks.reserve (5);

//5 flows S1->R1 and 5 flows S2->R1
for (std::size_t i = 0; i < 10; i++)
  {
    //....
    OnOffHelper clientHelper1 ("ns3::TcpSocketFactory", Address ());
    clientHelper1.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=
    clientHelper1.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant
    clientHelper1.SetAttribute ("DataRate", DataRateValue (DataRate ("500Mbps")));
    clientHelper1.SetAttribute ("PacketSize", UintegerValue (1000));
    //....
  }
```

2

Creating Multiple Flows

```
// Each sender in S1 and S2 sends to R1
std::vector<Ptr<PacketSink> > s1r1Sinks;
std::vector<Ptr<PacketSink> > s2r1Sinks;
s1r1Sinks.reserve (5);
s2r1Sinks.reserve (5);

//5 flows S1->R1 and 5 flows S2->R1
for (std::size_t i = 0; i < 10; i++)
  {
    ApplicationContainer clientApps1;
    AddressValue remoteAddress (InetSocketAddress (ipR1T2.GetAddress (0), port));
    clientHelper1.SetAttribute ("Remote", remoteAddress);
    if (i < 5)
      {
        clientApps1.Add (clientHelper1.Install (S1.Get (i)));
        clientApps1.Start (i * flowStartupWindow / 5 + clientStartTime + MilliSeconds (i * 5)
      }
    else
      {
        clientApps1.Add (clientHelper1.Install (S2.Get (i - 5)));
        clientApps1.Start ((i - 5) * flowStartupWindow / 5 + clientStartTime + MilliSeconds (
      }
    clientApps1.Stop (stopTime);
  }
```

Tracing

- Tracing Congestion Control Related Parameters:

```
//CWND calculation only for node T2
Simulator::Schedule (Seconds (0.00001), &TraceCwnd, dir + "-cwnd.dat");
Simulator::Schedule (Seconds (0.00001), &TraceRtt, dir + "-rtt.dat");
Simulator::Schedule (Seconds (0.00001), &TraceRto, dir + "-rto.dat");
Simulator::Schedule (Seconds (0.00001), &TraceSsThresh, dir + "-ssth.dat");
Simulator::Stop (stopTime + TimeStep(1));
```

Tracing

- List of defined callback functions :

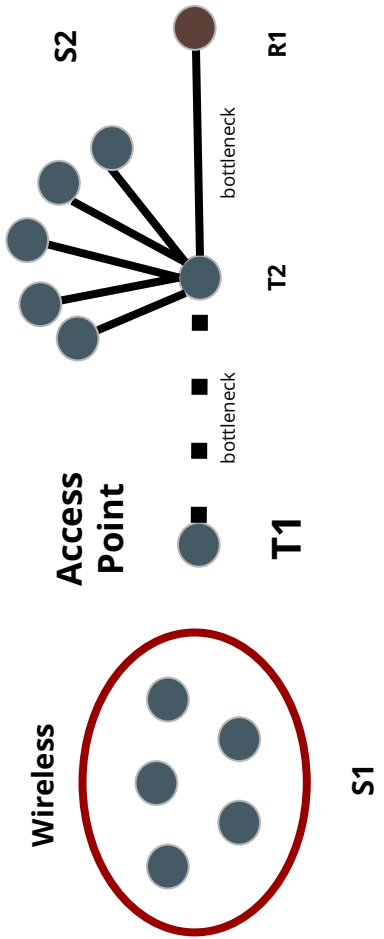
```
static void
CwndTracer (uint32_t oldval, uint32_t newval);
static void
SsThreshTracer (uint32_t oldval, uint32_t newval);
static void
RttTracer (Time oldval, Time newval);
static void
RtoTracer (Time oldval, Time newval);
static void
TraceCwnd (std::string cwnd_tr_file_name);
static void
TraceSsThresh (std::string ssthresh_tr_file_name);
static void
TraceRtt (std::string rtt_tr_file_name);
static void
TraceRto (std::string rto_tr_file_name);
```

Tracing CWND for Node T2

```
//CWND calculation only for node T2
Simulator::Schedule (Seconds (0.00001), &TraceCwnd, dir + "-cwnd.dat");
```

```
static void
TraceCwnd (std::string cwnd_tr_file_name)
{
  AsciiTraceHelper ascii;
  cWndStream = ascii.CreateFileStream (cwnd_tr_file_name.c_str ());
  Config::ConnectWithoutContext (
      "/NodeList/3/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow",
   MakeCallback (&CwndTracer));
}
```

```
static void
CwndTracer (uint32_t oldval, uint32_t newval)
{
  if (firstCwnd)
    {
      *cWndStream->GetStream () << "0.0 " << oldval << std::endl;
      firstCwnd = false;
    }
  *cWndStream->GetStream () << Simulator::Now ().GetSeconds () << " " << newval << std::endl;
  cWndValue = newval;

  if (!firstSshThr)
    {
      *ssThreshStream->GetStream ()
      << Simulator::Now ().GetSeconds () << " " << ssThreshValue << std::endl;
    }
}
```



Trace Files Generated

-cwnd.dat

my-dctcp-3-calculations > bottleneck=500Mbps >

```
0.0 0
4.13387 14480
4.13387 14480
4.19692 14480
4.19692 13032
4.20079 11584
4.20267 10498
4.20543 9654
4.20659 7240
4.24364 10136
4.28262 11584
4.32136 13032
4.34101 14480
4.35657 15928
4.38305 17376
4.38586 17376
4.38586 15928
4.38965 13277
```

-rto.dat

my-dctcp-3-calculations > bottleneck=500Mbps >

```
0.0 3
3 1.001
3 6
4.13387 3.399
4.25678 3.78275
4.28125 3.92309
4.28262 3.89563
4.29614 3.74484
4.31543 3.52526
4.31832 3.29209
4.32136 3.0356
4.32738 2.78064
4.33568 2.53701
4.33972 2.29761
4.34101 2.07023
4.34406 1.85431
4.34946 1.65627
4.35372 1.47873
4.35483 1.31919
4.35657 1.17375
4.3576 1.04383
4.36003 1

```

-rtt.dat

-rtt.dat

```
0.0 0
4.13387 1.133
4.25678 0.99725
4.28125 0.877719
4.28262 0.772879
4.29614 0.682894
4.31543 0.604907
4.31832 0.533919
4.32136 0.472054
4.32738 0.416922
4.33568 0.367307
4.33972 0.324019
4.34101 0.286016
4.34406 0.253139
4.34946 0.224247
4.35372 0.198466
4.35483 0.175533
4.35657 0.155466
4.3576 0.137658
4.36003 0.121826
4.37093 0.108722
4.37606 0.0978821
4.37929 0.0885219
4.38305 0.0808316
4.40796 0.0722277
4.4097 0.0645742
```

FlowMonitor

```
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();
flowmonitorOutput.open (dir+"flowSummary.dat", std::ios::out);

// variables for output measurement
float AvgThroughput = 0;
Time Jitter;
Time Delay;
uint32_t SentPackets = 0;
uint32_t ReceivedPackets = 0;
uint32_t LostPackets = 0;

Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats ();

for (auto iter = stats.begin (); iter != stats.end (); ++iter) {
  Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (iter->first);
      // classifier returns FiveTuple in correspondance to a flowID

  flowmonitorOutput << "----Flow ID:" <<iter->first << std::endl;
  flowmonitorOutput << "Src Addr" <<t.sourceAddress << " -- Dst Addr "<< t.destinationAddress<< std::endl;
  flowmonitorOutput << "Sent Packets=" <<iter->second.txPackets<< std::endl;
  flowmonitorOutput << "Received Packets =" <<iter->second.rxPackets<< std::endl;
  flowmonitorOutput << "Lost Packets =" <<iter->second.txPackets-iter->second.rxPackets<< std::endl;
  flowmonitorOutput << "Packet delivery ratio =" <<iter->second.rxPackets*100.0/iter->second.txPackets << "%"<< std::endl;
  flowmonitorOutput << "Packet loss ratio =" << (iter->second.txPackets-iter->second.rxPackets)*100.0/iter->second.txPackets << "%"<< std::endl;
  // flowmonitorOutput << "Packet lost diff way = "<< iter->second.lostPackets;
  flowmonitorOutput << "Delay =" <<iter->second.delaySum<< std::endl;
  flowmonitorOutput << "Jitter =" <<iter->second.jitterSum<< std::endl;
  flowmonitorOutput << "Throughput =" <<iter->second.rxBytes * 8.0/(iter->second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSec

  SentPackets = SentPackets +(iter->second.txPackets);
  ReceivedPackets = ReceivedPackets + (iter->second.rxPackets);
  LostPackets = LostPackets + (iter->second.txPackets-iter->second.rxPackets);
  AvgThroughput = AvgThroughput + (iter->second.rxBytes * 8.0/(iter->second.timeLastRxPacket.GetSeconds()-iter->second.timeFirstTxPacket.GetSeconds(
  Delay = Delay + (iter->second.delaySum);
  Jitter = Jitter + (iter->second.jitterSum);
  j = j + 1;
}
```

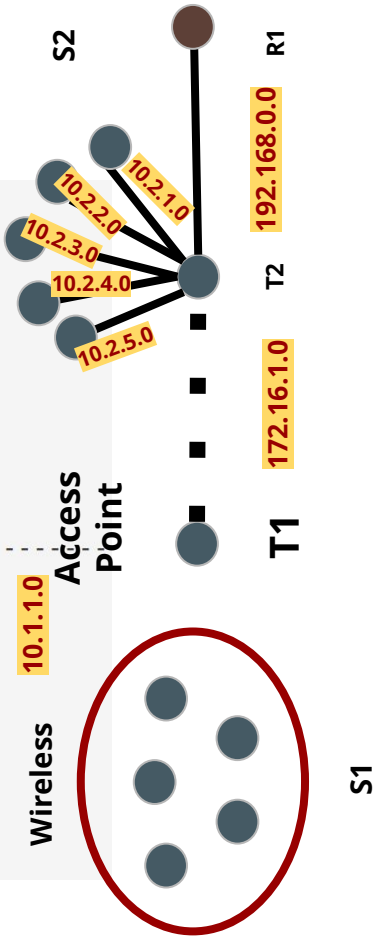
FlowMonitor - Output

flowSummary.dat

my-dctcp-3-calculations > bottleneck=500Mbps > flowSummary.dat

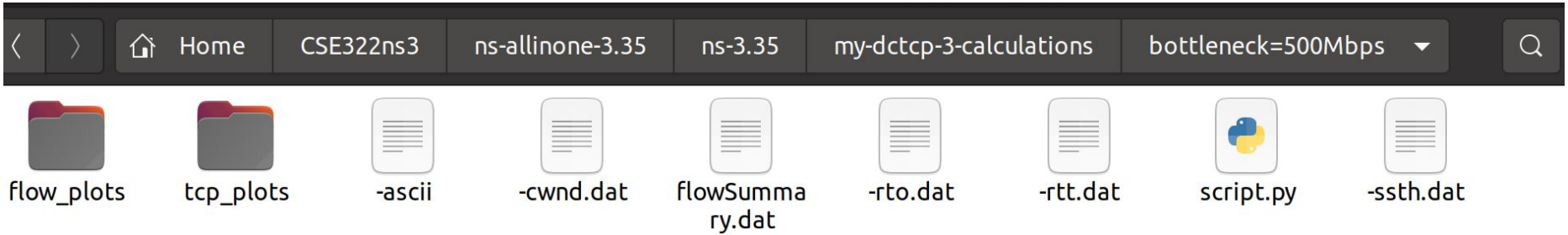
```
1   ----Flow ID:1
2   Src Addr10.1.1.1 -- Dst Addr 192.168.0.1
3   Sent Packets=2086
4   Received Packets =2021
5   Lost Packets =65
6   Packet delivery ratio =96.884%
7   Packet loss ratio =3.11601%
8   Delay =+6.44429e+09ns
9   Jitter =+2.21257e+09ns
10  Throughput =2629.42Kbps
11  ----Flow ID:2
12  Src Addr10.2.1.1 -- Dst Addr 192.168.0.1
13  Sent Packets=8316
14  Received Packets =8058
15  Lost Packets =258
16  Packet delivery ratio =96.8975%
17  Packet loss ratio =3.10245%
18  Delay =+6.2718e+09ns
19  Jitter =+9.99396e+07ns
20  Throughput =10375.8Kbps
```

```
191  ----Flow ID:20
192  Src Addr192.168.0.1 -- Dst Addr 10.1.1.1
193  Sent Packets=1630
194  Received Packets =1594
195  Lost Packets =36
196  Packet delivery ratio =97.7914%
197  Packet loss ratio =2.20859%
198  Delay =+9.36968e+10ns
199  Jitter =+2.52175e+09ns
200  Throughput =127.013Kbps
201  --------Total Results of the simulation----------
202  Total sent packets  =89420
203  Total Received Packets =87747
204  Total Lost Packets =1673
205  Packet Loss ratio =1.87095%
206  Packet delivery ratio =98.1291%
207  Average Throughput =3463.6Kbps
208  End to End Delay =+6.78658e+11ns
209  End to End Jitter delay =+2.5146e+10ns
210  Total Flow id 20
```

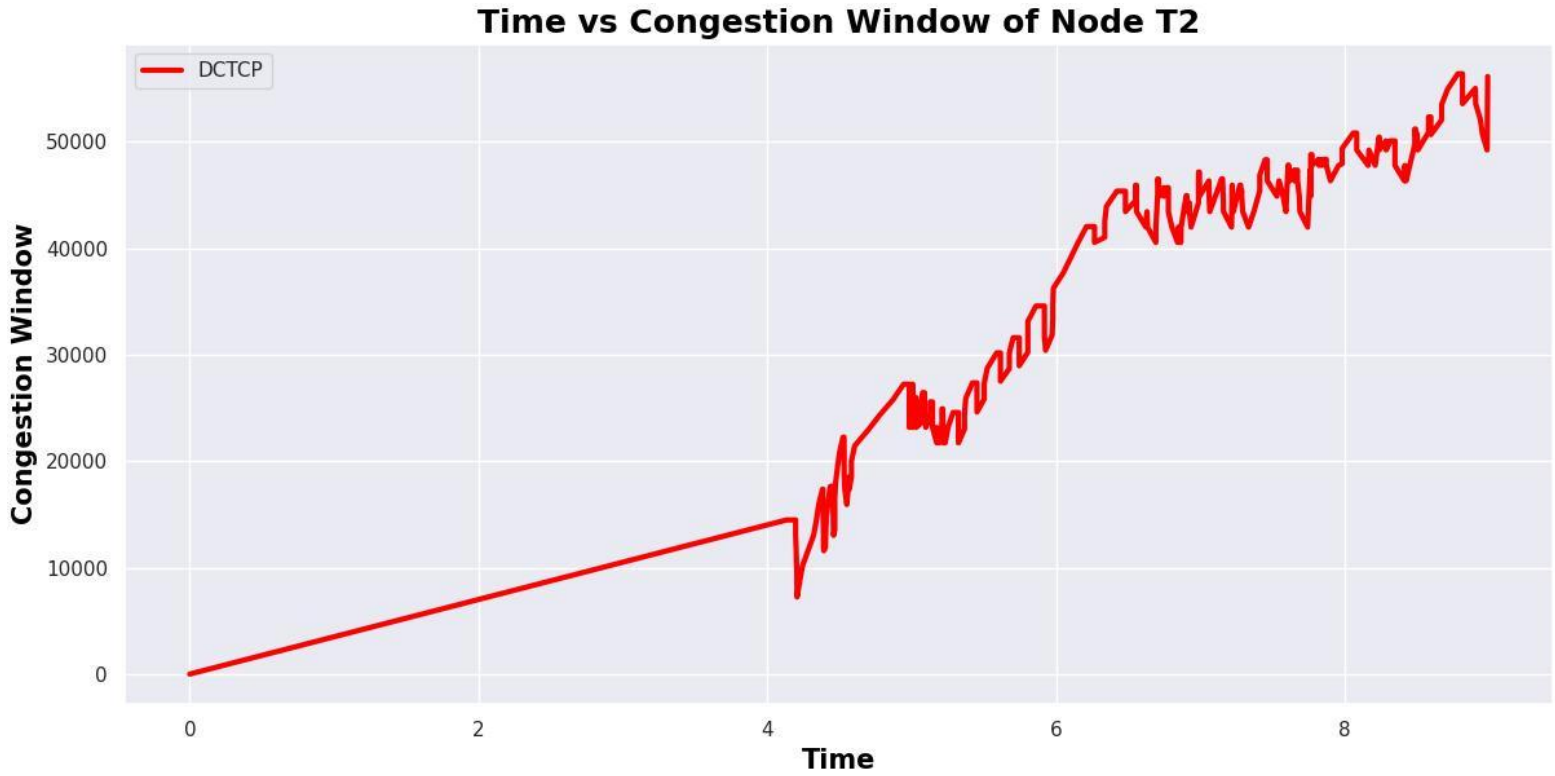


Parser for plotting

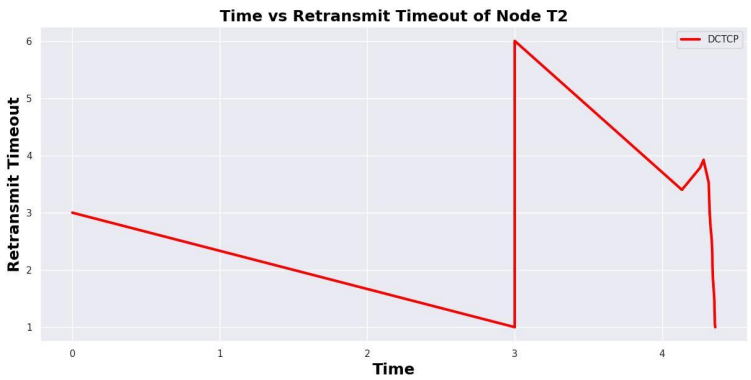
- Wrote a script in python to parse generated .dat files
- Generated plots on all parameters from all outputs using matplotlib

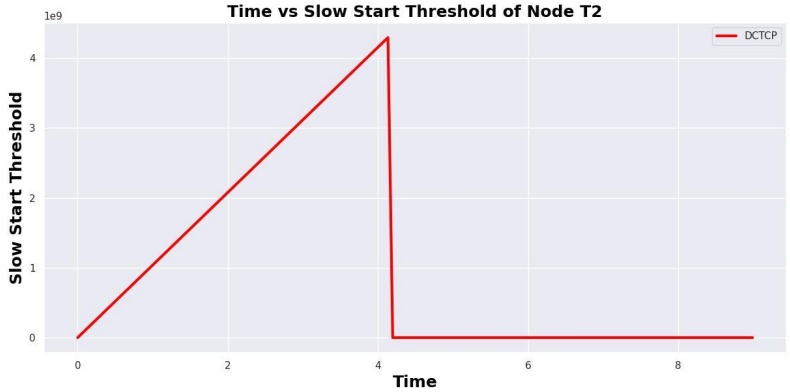


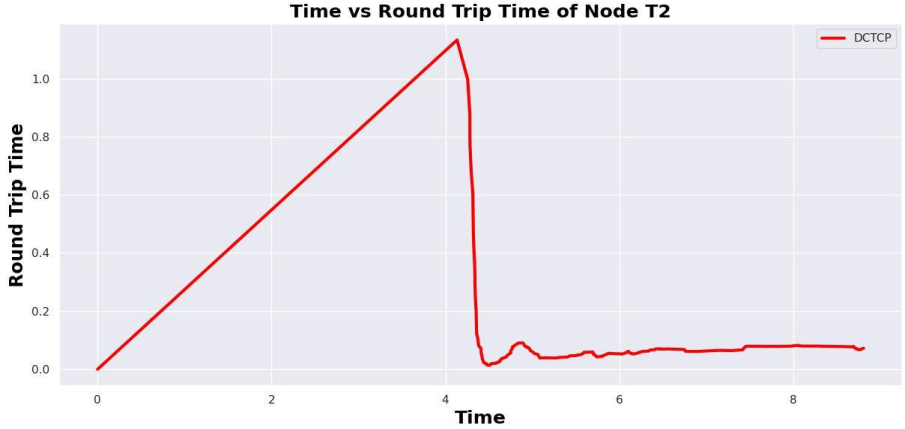
Congestion Window Plot



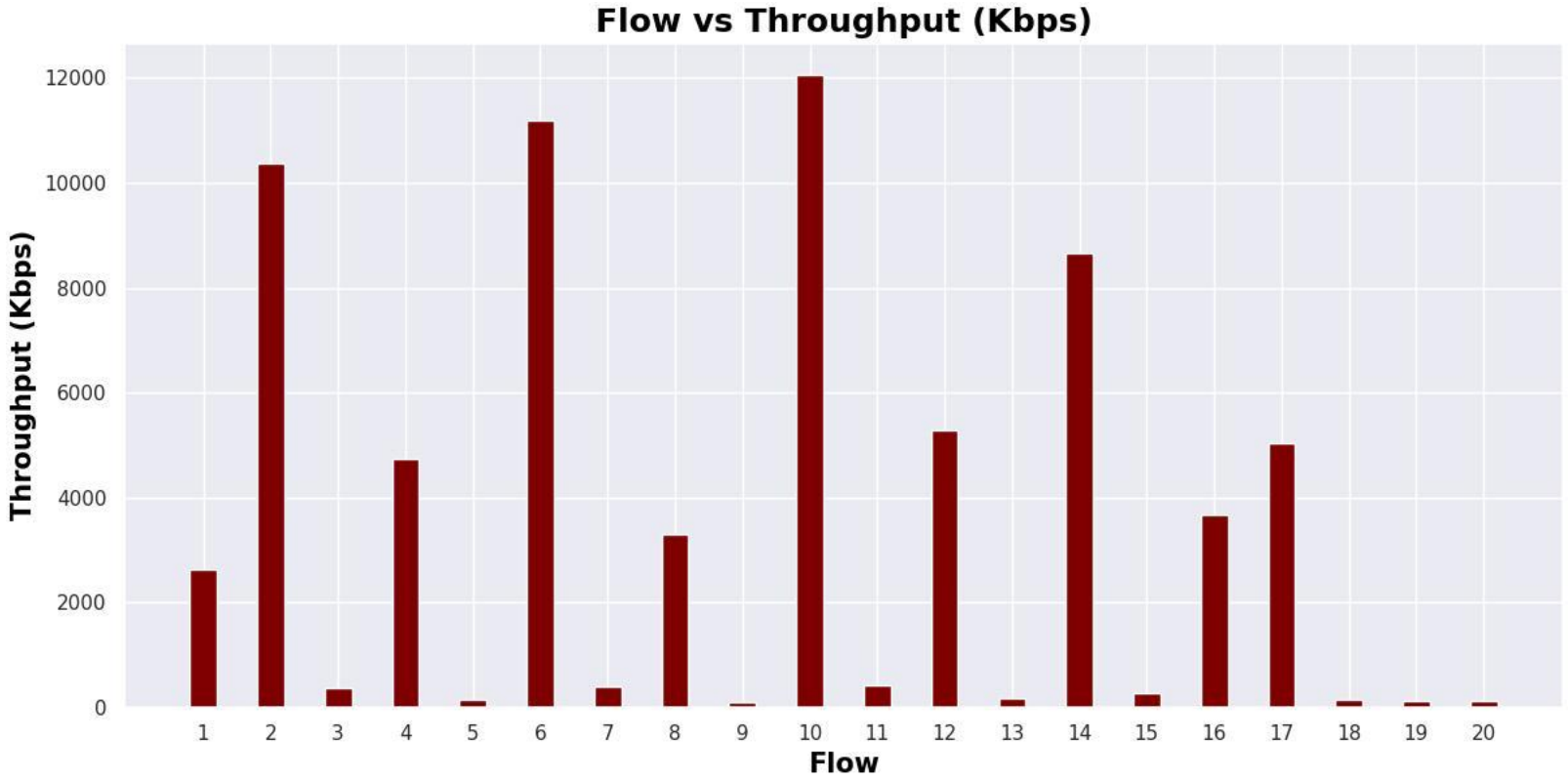
Plots of Other Congestion Parameters



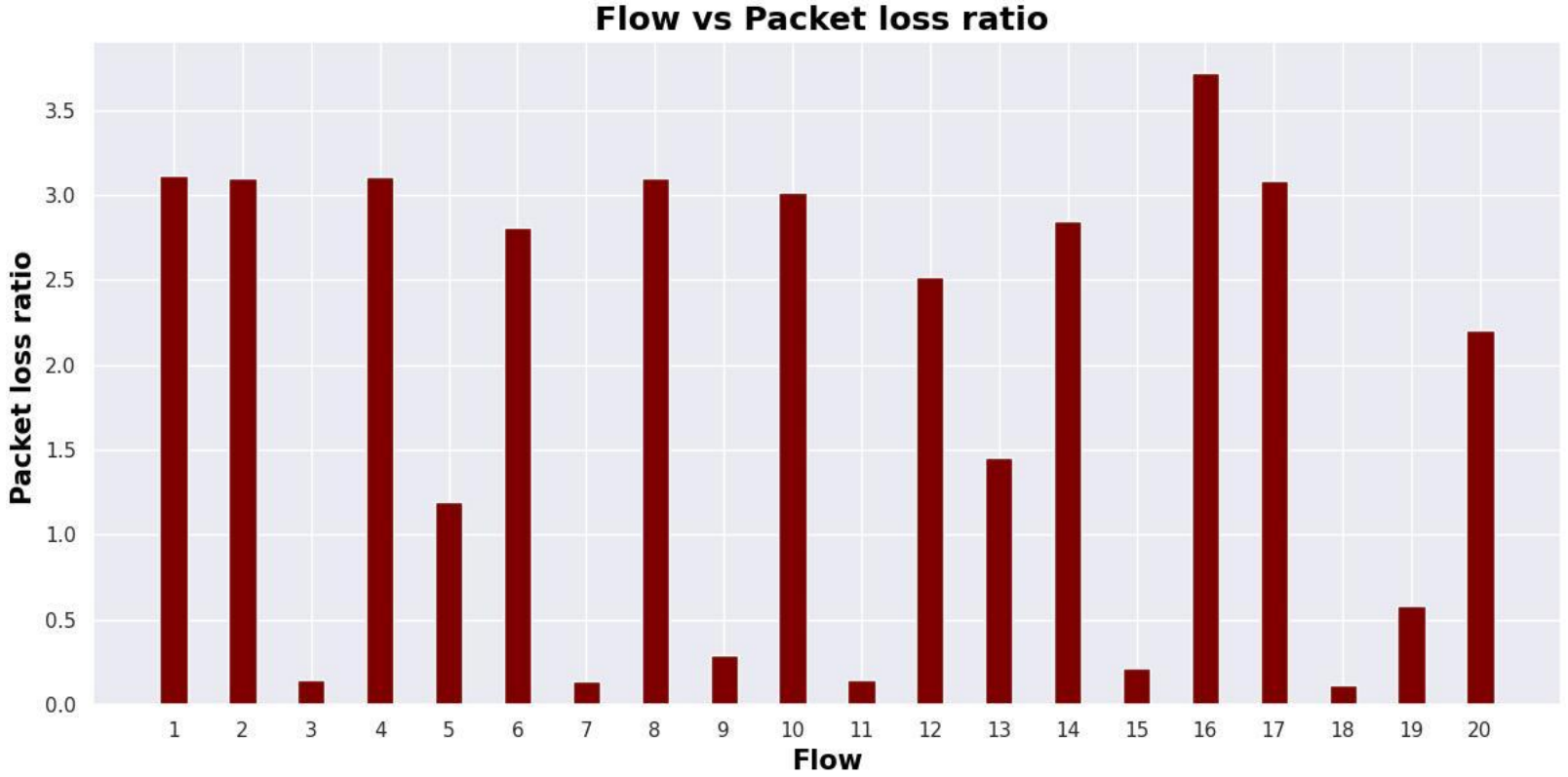




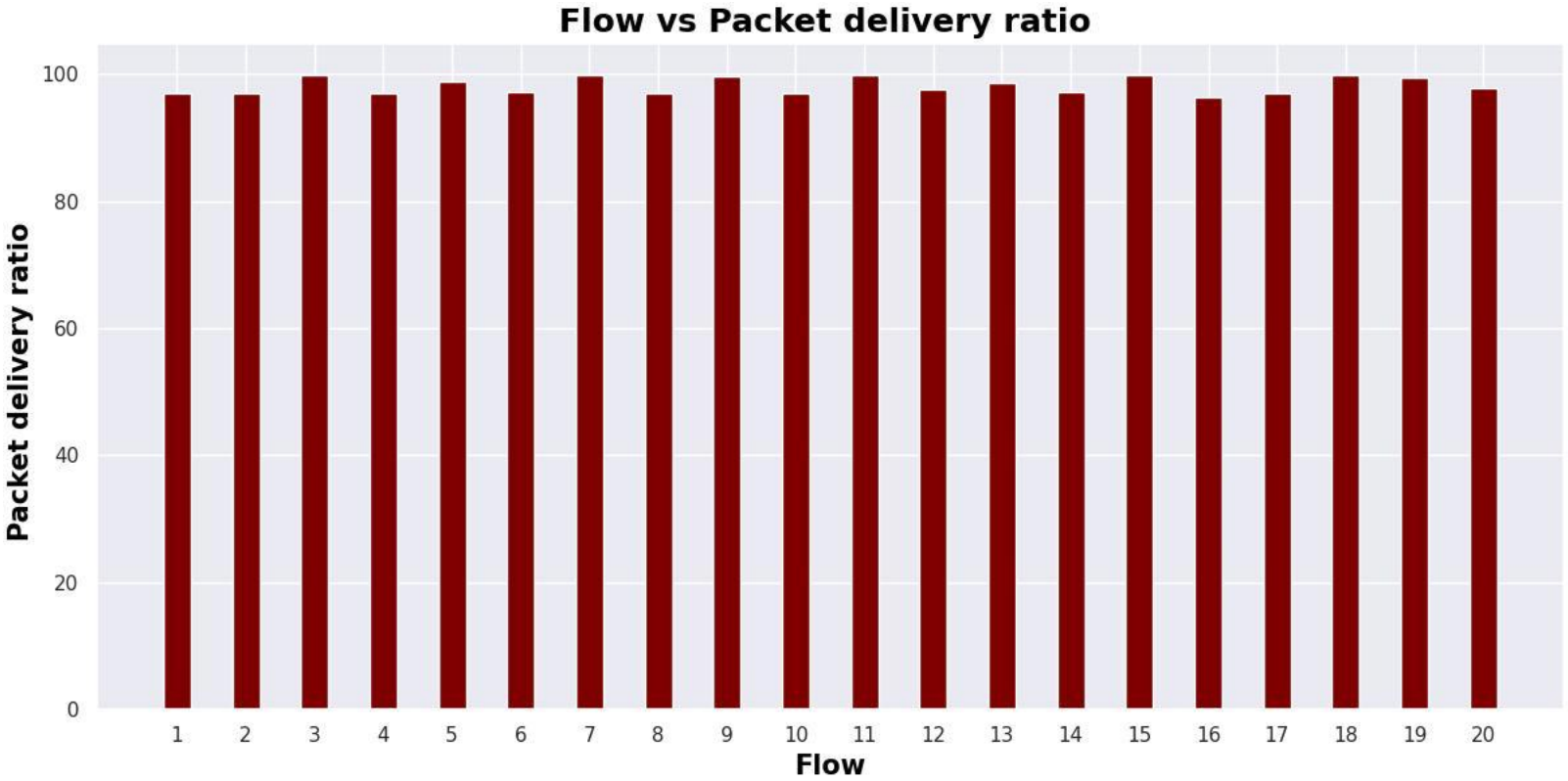
Flow - Network Throughput Plot



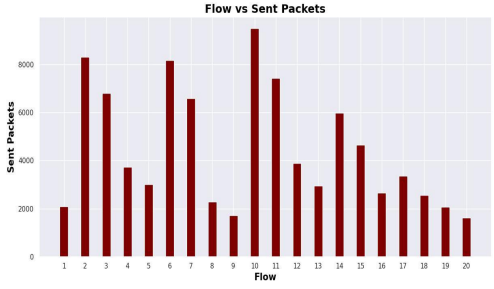
Flow - Packet Loss Ratio Plot

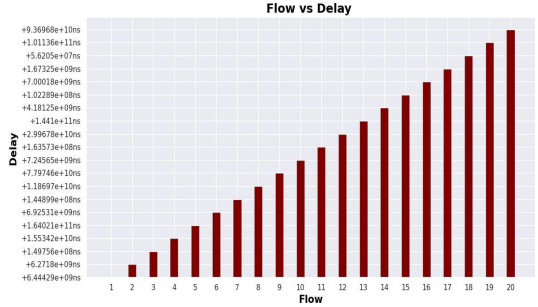


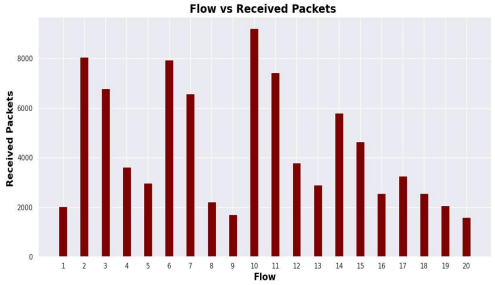
Flow - Packet Delivery Ratio Plot

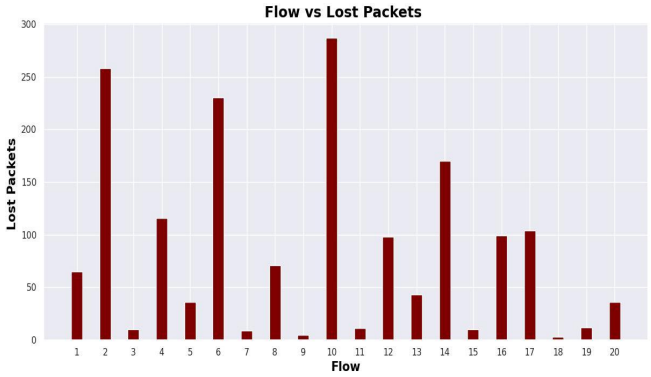


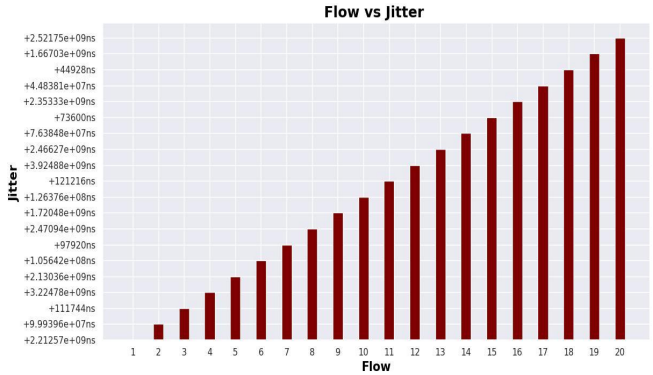
Flow - Other Parameters











Adding Modified Congestion Control Algorithm TCP-Swift

Steps :

1.
```
$cd ns-3.35/src/internet/model/
$cp tcp-dctcp.cc tcp-swift.cc
$cp tcp-dctcp.h tcp-swift.h
```

2. Replaced all “dctcp” with “swift”

3. Changed src/internet/wscript :

```
def build(bld):
    obj.source = [
        added 'model/tcp-swift.cc',
    headers.source = [
        added 'model/tcp-swift.h',
```

4. Changed my test script

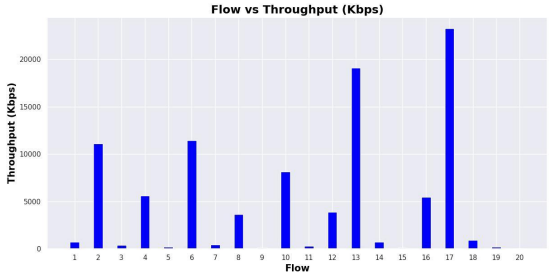
```
std::string tcpTypeId = "TcpSwift";
Config::SetDefault ("ns3::TcpL4Protocol::SocketType", StringValue ("ns3::" + tcpTypeId));
```

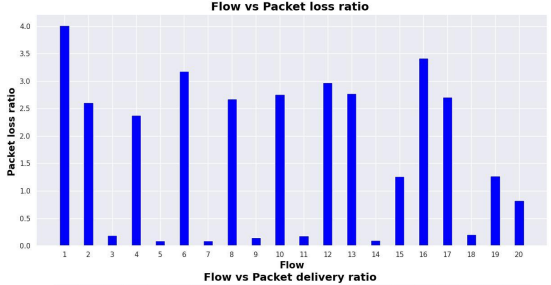
Tweaking The Code of DCTCP

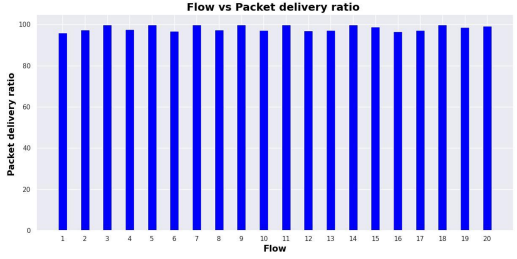
- Changed estimation gain from 1/16 to 1/8

```
TypeId TcpSwift::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::TcpSwift")
    .SetParent<TcpLinuxReno> ()
    .AddConstructor<TcpSwift> ()
    .SetGroupName ("Internet")
    .AddAttribute ("SwiftShiftG",
                   "Parameter G for updating swift_alpha",
                   DoubleValue (0.125),
                   MakeDoubleAccessor (&TcpSwift::m_g),
                   MakeDoubleChecker<double> (0, 1))
```

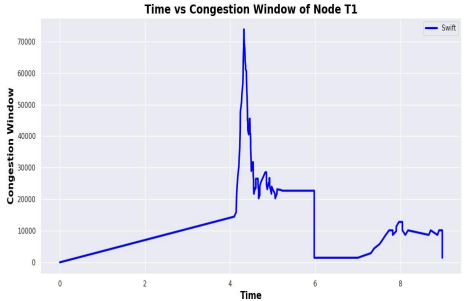
Generated Plots by Swift

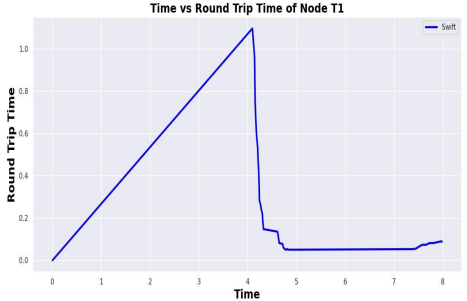


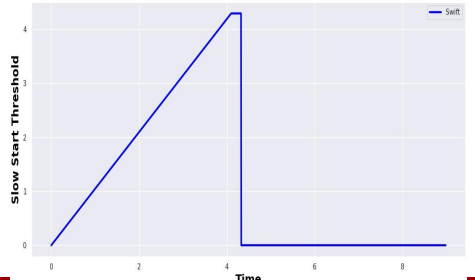




Flow

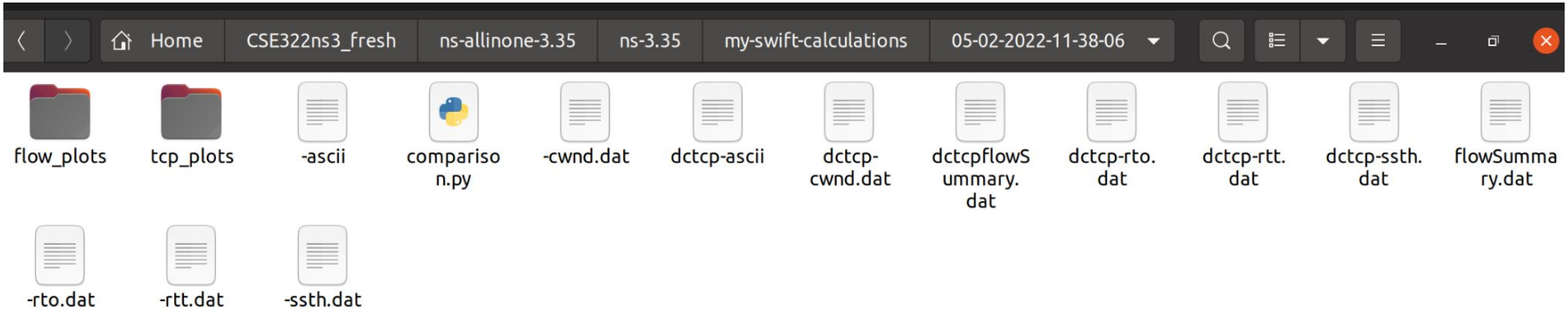




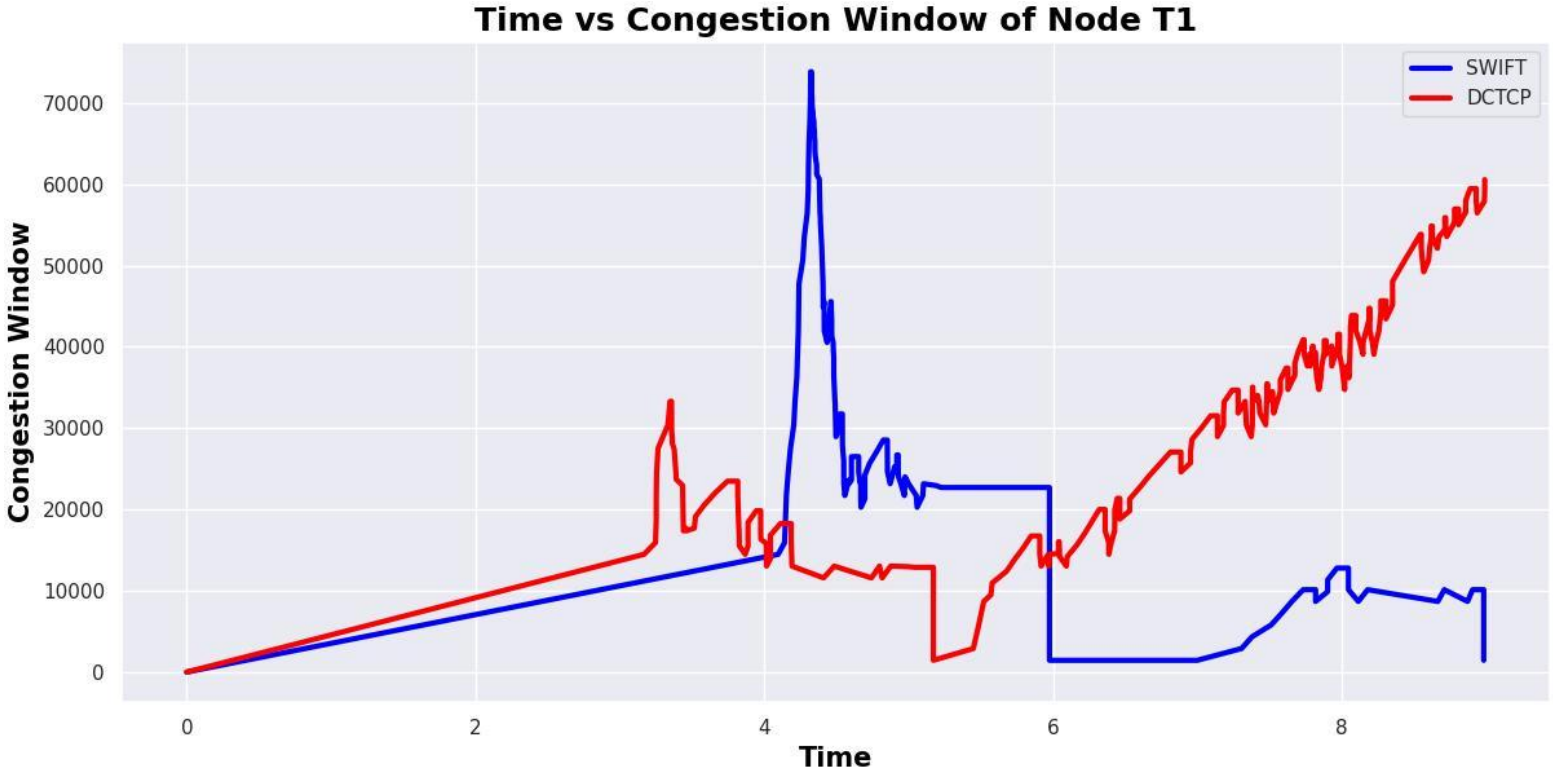


TCP

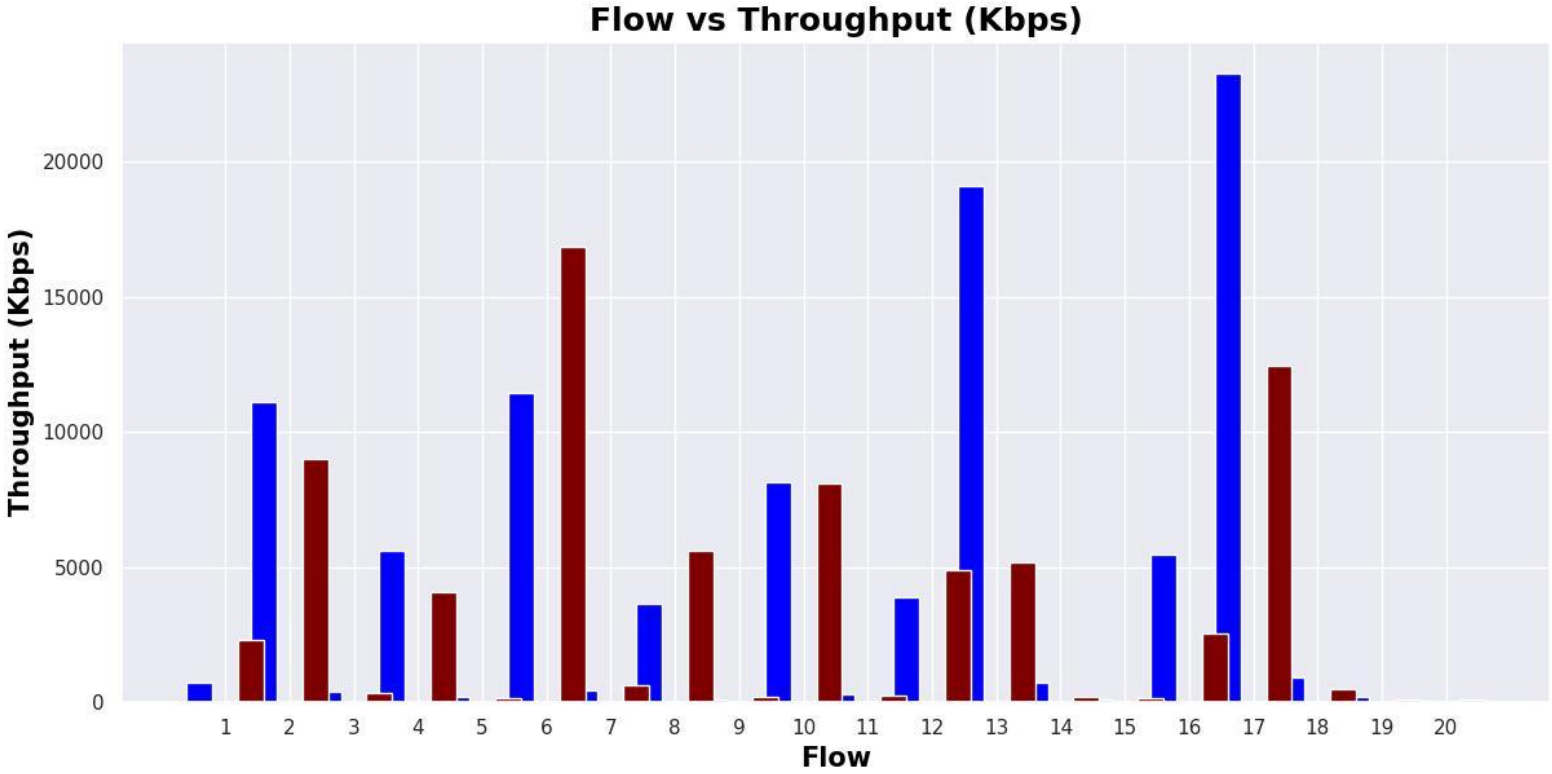
Another parser for comparison plots



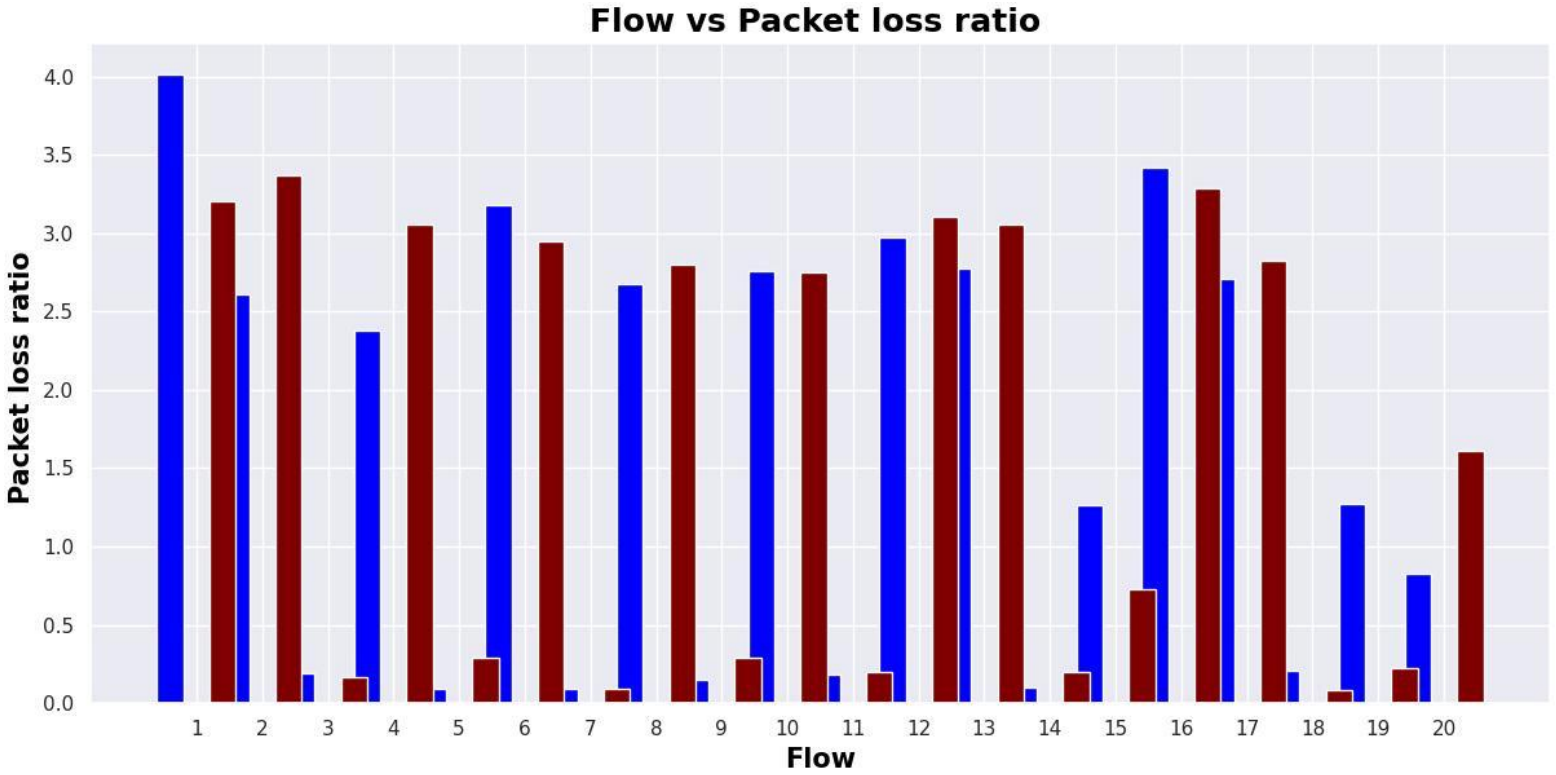
Congestion Window Comparison



Throughput Comparison

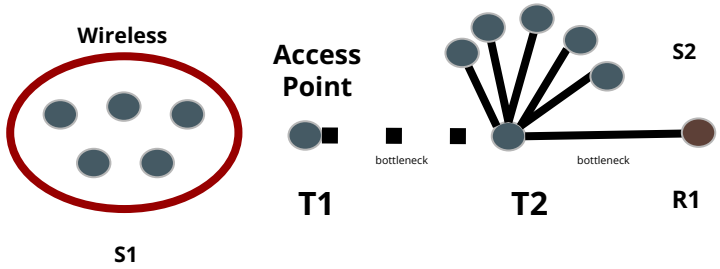


Packet Loss Ratio Comparison



Queries

Queries



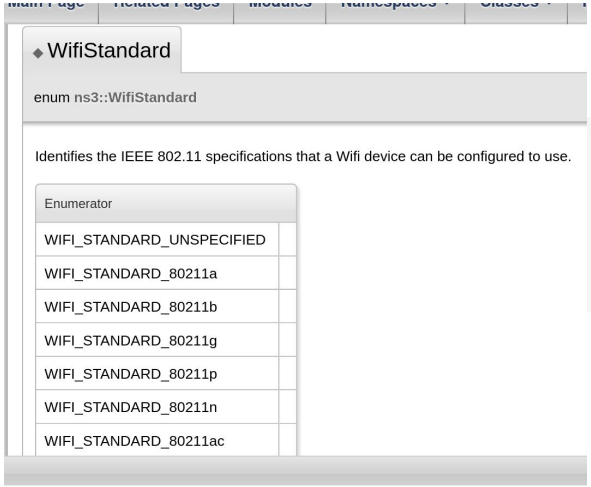
- Cannot set congestion window trace on wifi nodes
- So cannot calculate congestion window of Node T1

```
Config::ConnectWithoutContext (
  "/NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow",
   MakeCallback (&CwndTracer));
```

```
msg="Could not connect callback to /NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow", +0.000010000s -1 file=../src/core/model/config.cc, line=906
terminate called without an active exception
Command ['/home/sadia/CSE322ns3/ns-allinone-3.35/ns-3.35/build/scratch/my-dctcp-3'] terminated with signal SIGIOT. Run it under a debugger to get more information (./waf --run <program> --gdb").
```

Queries

- Wireless low-rate (e.g.,802.15.4)
  - How to set standard?
  - Use LrWpanNetDevice instead of NetDevice?



◆ WifiStandard

enum ns3::WifiStandard

Identifies the IEEE 802.11 specifications that a Wifi device can be configured to use.

Enumerator
WIFI_STANDARD_UNSPECIFIED
WIFI_STANDARD_80211a
WIFI_STANDARD_80211b
WIFI_STANDARD_80211g
WIFI_STANDARD_80211p
WIFI_STANDARD_80211n
WIFI_STANDARD_80211ac

WifiHelper.SetStandard()

```
Ptr<Node> n0 = CreateObject <Node> ();
Ptr<Node> n1 = CreateObject <Node> ();
Ptr<LrWpanNetDevice> dev0 = CreateObject<LrWpanNetDevice> ();
Ptr<LrWpanNetDevice> dev1 = CreateObject<LrWpanNetDevice> ();
```

src/lr-wpan/examples/lr-wpan-error-distance-plot.cc

https://ns-3-users.narkive.com/riLh5fR8/internet-stack-over-lr-wpan

Queries

- Coverage Area

```
// Set the maximum wireless range to 5 meters
// in order to reproduce a hidden nodes scenario,
// i.e. the distance between hidden stations is larger than 5 meters
Config::SetDefault (
  "ns3::RangePropagationLossModel::MaxRange", DoubleValue (5));
```

examples/wireless/wifi-simple-ht-hidden-stations.cc

There are two large families of congestion control protocols that attempt to control queue lengths:

(i) **Delay-based protocols** use increases in RTT measurements as a sign of growing queueing delay, and hence of congestion. These protocols rely heavily on accurate RTT measurement (Swift)

(ii) **Active Queue Management (AQM)** approaches use explicit feedback from congested switches (DCTCP)