TCP Variant Simulation using NS3

Name: Karim Md Monjurul

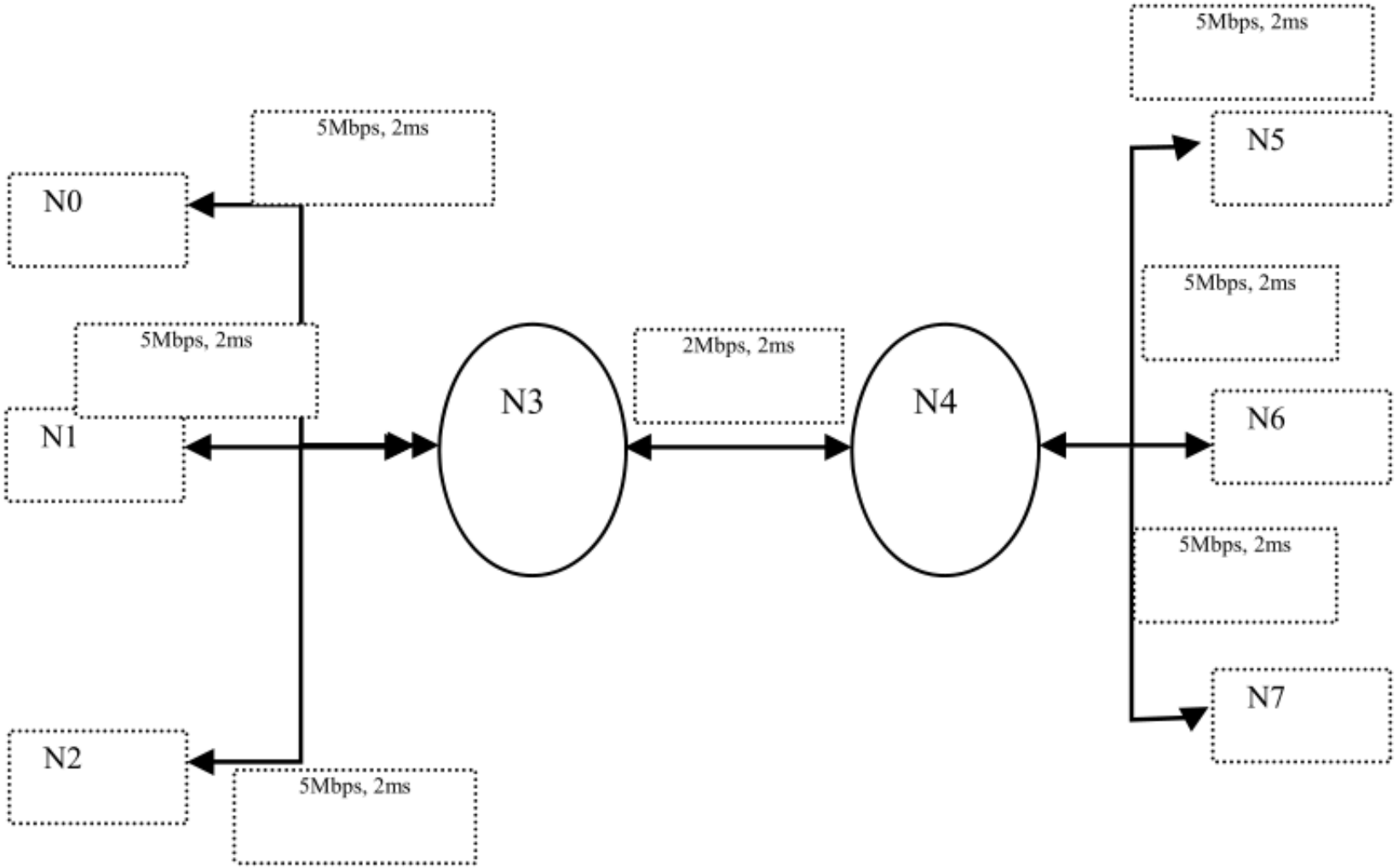
ID: 3820160025

Supervisor: Prof. Zhu Lie Huang

What we have used

- VMWare Workstation Player 12.5
- Ubuntu 16.04 LTS x64
- NS3 version 3.26
- NetAnim 3.107
- Gnuplot

Project Topology



Source Code Demonstration Main Function and Nodes Creation and defining TCP Congestion Algorithm Type

```
int main ()
{

  std::string lat = "2ms";
  std::string rate = "5Mbps";   // P2P link
  std::string rate1="2Mbps";   // for Node 3 to 4
  bool enableFlowMonitor = false;

  CommandLine cmd;
  cmd.AddValue ("latency", "P2P link Latency in miliseconds", lat);
  cmd.AddValue ("rate", "P2P data rate in bps", rate);
  cmd.AddValue ("EnableMonitor", "Enable Flow Monitor", enableFlowMonitor);

  //cmd.Parse (argc, argv);
 //Sets the default congestion control algorithm
  Config::SetDefault("ns3::TcpL4Protocol::SocketType", StringValue("ns3::TcpNewReno"));
```

```
//****************** Nodes Creation required by the topology as Shown above

  NS_LOG_INFO ("Create nodes.");
  NodeContainer c; // ALL Nodes
  c.Create(8);

  NodeContainer n0n3 = NodeContainer (c.Get (0), c.Get (3));
  NodeContainer n1n3 = NodeContainer (c.Get (1), c.Get (3));
  NodeContainer n2n3 = NodeContainer (c.Get (2), c.Get (3));
  NodeContainer n3n4 = NodeContainer (c.Get (3), c.Get (4));
  NodeContainer n5n4 = NodeContainer (c.Get (5), c.Get (4));
  NodeContainer n6n4 = NodeContainer (c.Get (6), c.Get (4));
  NodeContainer n7n4 = NodeContainer (c.Get (7), c.Get (4));
```

Source Code Demonstration Stack Creation and IP address Assign

```
//************************ Install Internet Stack*******************

  InternetStackHelper internet;
  internet.Install (c);

//**************** channels Creation without IP addressing*********

  NS_LOG_INFO ("Create channels.");
  PointToPointHelper p2p,p2p_for3_4;
  p2p.SetDeviceAttribute ("DataRate", StringValue (rate));
  p2p.SetChannelAttribute ("Delay", StringValue (lat));
  NetDeviceContainer d0d3 = p2p.Install (n0n3);
  NetDeviceContainer d1d3 = p2p.Install (n1n3);
  NetDeviceContainer d2d3 = p2p.Install (n2n3);
  NetDeviceContainer d5d4 = p2p.Install (n5n4);
  NetDeviceContainer d6d4 = p2p.Install (n6n4);
  NetDeviceContainer d7d4 = p2p.Install (n7n4);

  p2p_for3_4.SetDeviceAttribute ("DataRate", StringValue (rate1));
  p2p_for3_4.SetChannelAttribute ("Delay", StringValue (lat));
  NetDeviceContainer d3d4 = p2p_for3_4.Install (n3n4);
```

```
//**********************IP addresses Setup************************
  NS_LOG_INFO ("Assign IP Addresses.");
  Ipv4AddressHelper ipv4;
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i0i3 = ipv4.Assign (d0d3);

  ipv4.SetBase ("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer i1i3 = ipv4.Assign (d1d3);

  ipv4.SetBase ("10.1.3.0", "255.255.255.0");
  Ipv4InterfaceContainer i2i3 = ipv4.Assign (d2d3);

  ipv4.SetBase ("10.1.4.0", "255.255.255.0");
  Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);

  ipv4.SetBase ("10.1.5.0", "255.255.255.0");
  Ipv4InterfaceContainer i5i4 = ipv4.Assign (d5d4);

  ipv4.SetBase ("10.1.6.0", "255.255.255.0");
  Ipv4InterfaceContainer i6i4 = ipv4.Assign (d6d4);

  ipv4.SetBase ("10.1.7.0", "255.255.255.0");
  Ipv4InterfaceContainer i7i4 = ipv4.Assign (d7d4);
```

Source Code Demonstration TCP Node N0 to N5

```
//************** TCP connection from N0 to N5*********************************************
uint16_t sinkPort1 = 8080;
Address sinkAddress1 (InetSocketAddress (i5i4.GetAddress (0), sinkPort1)); // interface of n5
PacketSinkHelper packetSinkHelper1 ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort1));
ApplicationContainer sinkApps1 = packetSinkHelper1.Install (c.Get (5)); //n5 as sink
sinkApps1.Start (Seconds (2.));
// sinkApps.Stop (Seconds (25.));

Ptr<Socket> ns3TcpSocket1 = Socket::CreateSocket (c.Get (0), TcpSocketFactory::GetTypeId ()); //source at n0

//********************** Congestion window**********************
ns3TcpSocket1->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));

//**********************TCP application at N0*********************************
Ptr<MyApp> app1 = CreateObject<MyApp> ();
app1->Setup (ns3TcpSocket1, sinkAddress1, 1040, 100000, DataRate ("1Mbps"));
c.Get (0)->AddApplication (app1);
app1->SetStartTime (Seconds (2.));
// app->SetStopTime (Seconds (25.));
```

Source Code Demonstration TCP Node N1 to N6

```
//*************** TCP connection from N1 to N6**********************
uint16_t sinkPort2 = 8081;
Address sinkAddress2 (InetSocketAddress (i6i4.GetAddress (0), sinkPort2)); // interface of n6
PacketSinkHelper packetSinkHelper2 ("ns3::TcpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort2));
ApplicationContainer sinkApps2 = packetSinkHelper2.Install (c.Get (6)); //n6 as sink
sinkApps2.Start (Seconds (5.));

Ptr<Socket> ns3TcpSocket2 = Socket::CreateSocket (c.Get (1), TcpSocketFactory::GetTypeId ()); //source at n1

//************* Congestion window for N1 to N6
ns3TcpSocket2->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));

// Create TCP application at N1
Ptr<MyApp> app2 = CreateObject<MyApp> ();
app2->Setup (ns3TcpSocket2, sinkAddress2, 1040, 100000, DataRate ("1Mbps"));
c.Get (1)->AddApplication (app2);
app2->SetStartTime (Seconds (5.));
//  app2->SetStopTime (Seconds (25.));
```

Source Code Demonstration UDP Node N2 to N7

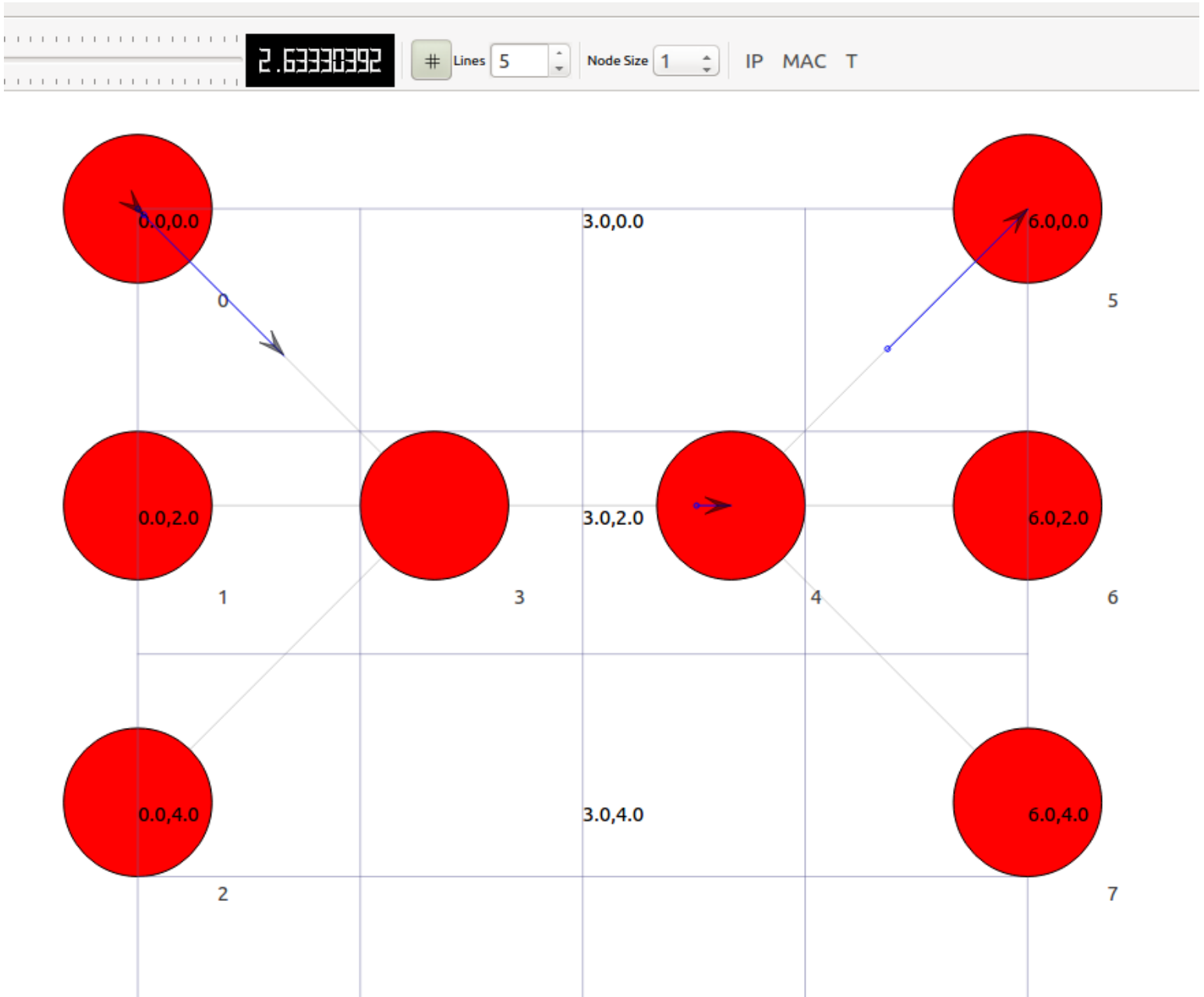
```
// **********************UDP connection from N2 to N7 ****************************
uint16_t sinkPort3 = 6;
Address sinkAddress3 (InetSocketAddress (i7i4.GetAddress (0), sinkPort3)); // interface of n7
PacketSinkHelper packetSinkHelper3 ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), sinkPort3));
ApplicationContainer sinkApps3 = packetSinkHelper3.Install (c.Get (7)); //n7 as sink
sinkApps3.Start (Seconds (10.));
sinkApps3.Stop (Seconds (17.));

Ptr<Socket> ns3UdpSocket = Socket::CreateSocket (c.Get (2), UdpSocketFactory::GetTypeId ()); //source at n2

// Create UDP application at N2
Ptr<MyApp> app3 = CreateObject<MyApp> ();
app3->Setup (ns3UdpSocket, sinkAddress3, 1040, 100000, DataRate ("1Mbps"));
c.Get (2)->AddApplication (app3);
app3->SetStartTime (Seconds (10.));
app3->SetStopTime (Seconds (17.));
```

Source Code Demonstration Final Part

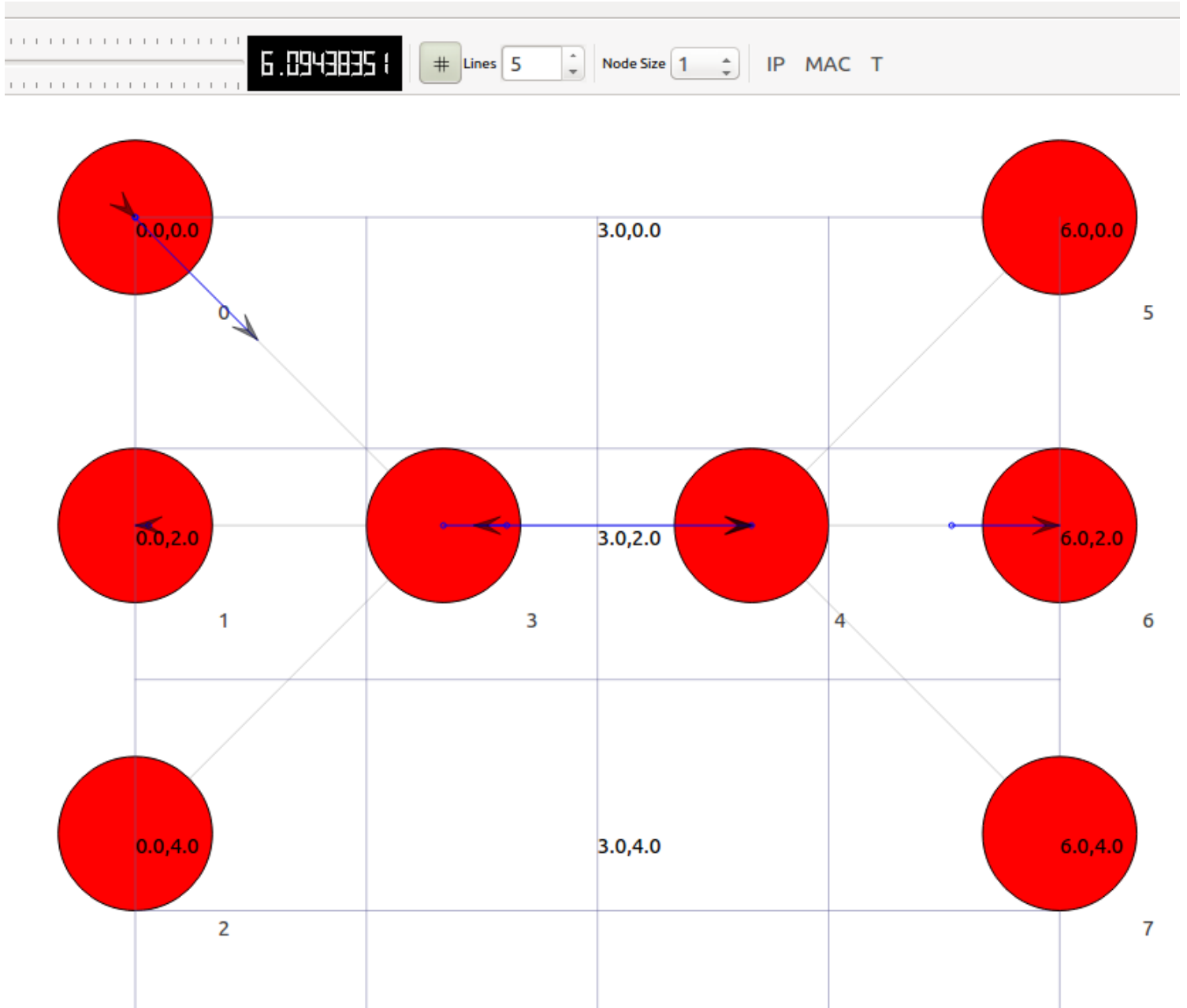
```
// Flow Monitor
Ptr<FlowMonitor> flowmon;
if (enableFlowMonitor)
 {
  FlowMonitorHelper flowmonHelper;
  flowmon = flowmonHelper.InstallAll ();
 }

//
// Now, do the actual simulation.
//
NS_LOG_INFO ("Run Simulation.");
 Simulator::Stop (Seconds(25.0));

  //Enabling Pcap Tracing
  //p2p.EnablePcapAll("scratch/Assignment");
```

```
AnimationInterface anim("Assignment.xml");
anim.SetConstantPosition(c.Get(0),0.0,0.0);
anim.SetConstantPosition(c.Get(1),0.0,2.0);
anim.SetConstantPosition(c.Get(2),0.0,4.0);
anim.SetConstantPosition(c.Get(3),2.0,2.0);
anim.SetConstantPosition(c.Get(4),4.0,2.0);
anim.SetConstantPosition(c.Get(5),6.0,0.0);
anim.SetConstantPosition(c.Get(6),6.0,2.0);
anim.SetConstantPosition(c.Get(7),6.0,4.0);

Simulator::Run ();
if (enableFlowMonitor)
 {
 flowmon->CheckForLostPackets ();
 flowmon->SerializeToXmlFile("Assignment.flowmon", true, true);
 }
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```

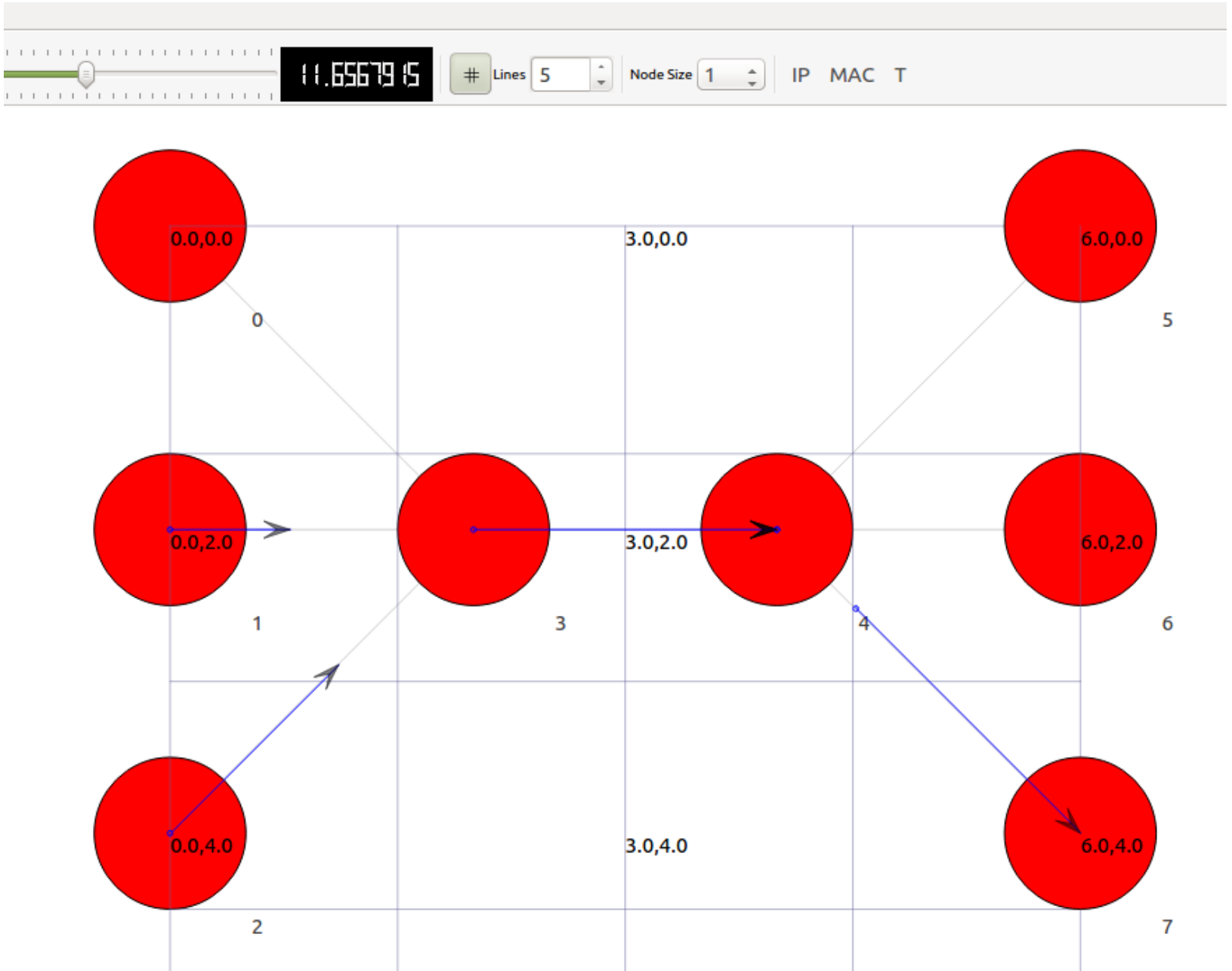
NetAnim Demonstration N0-N5 TCP



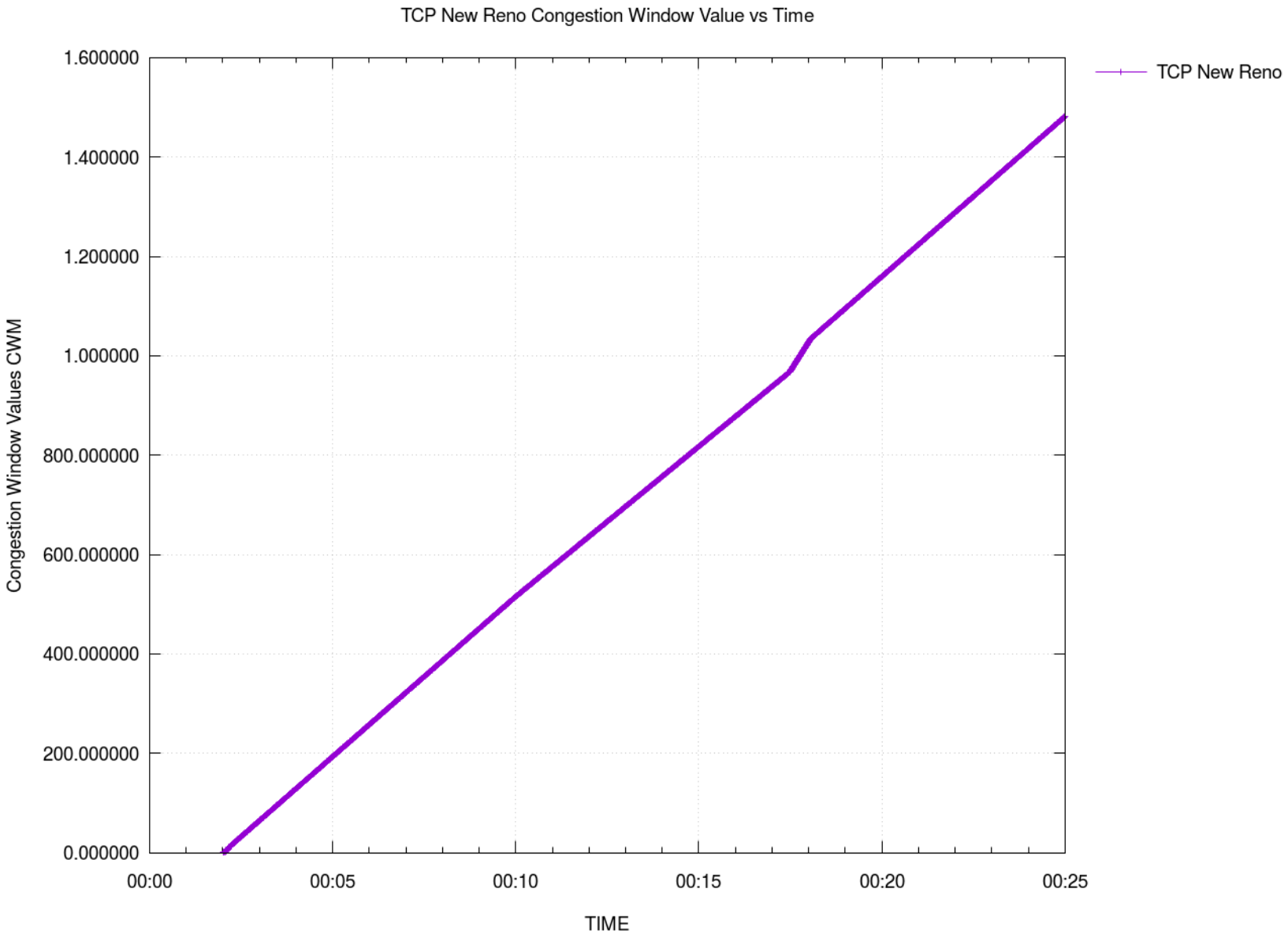
NetAnim Demonstration N1-N6 TCP



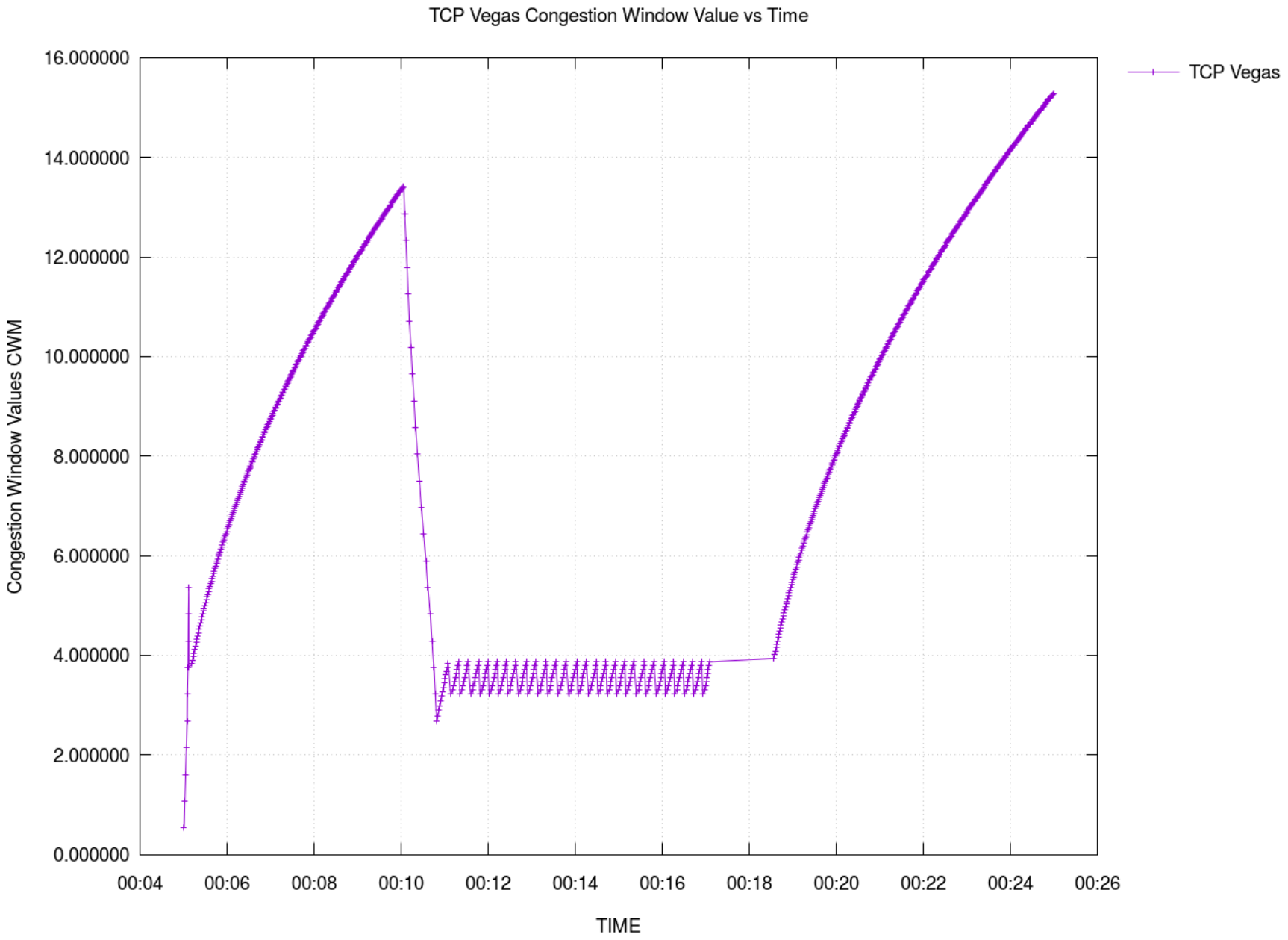
NetAnim Demonstration N2-N7 UDP



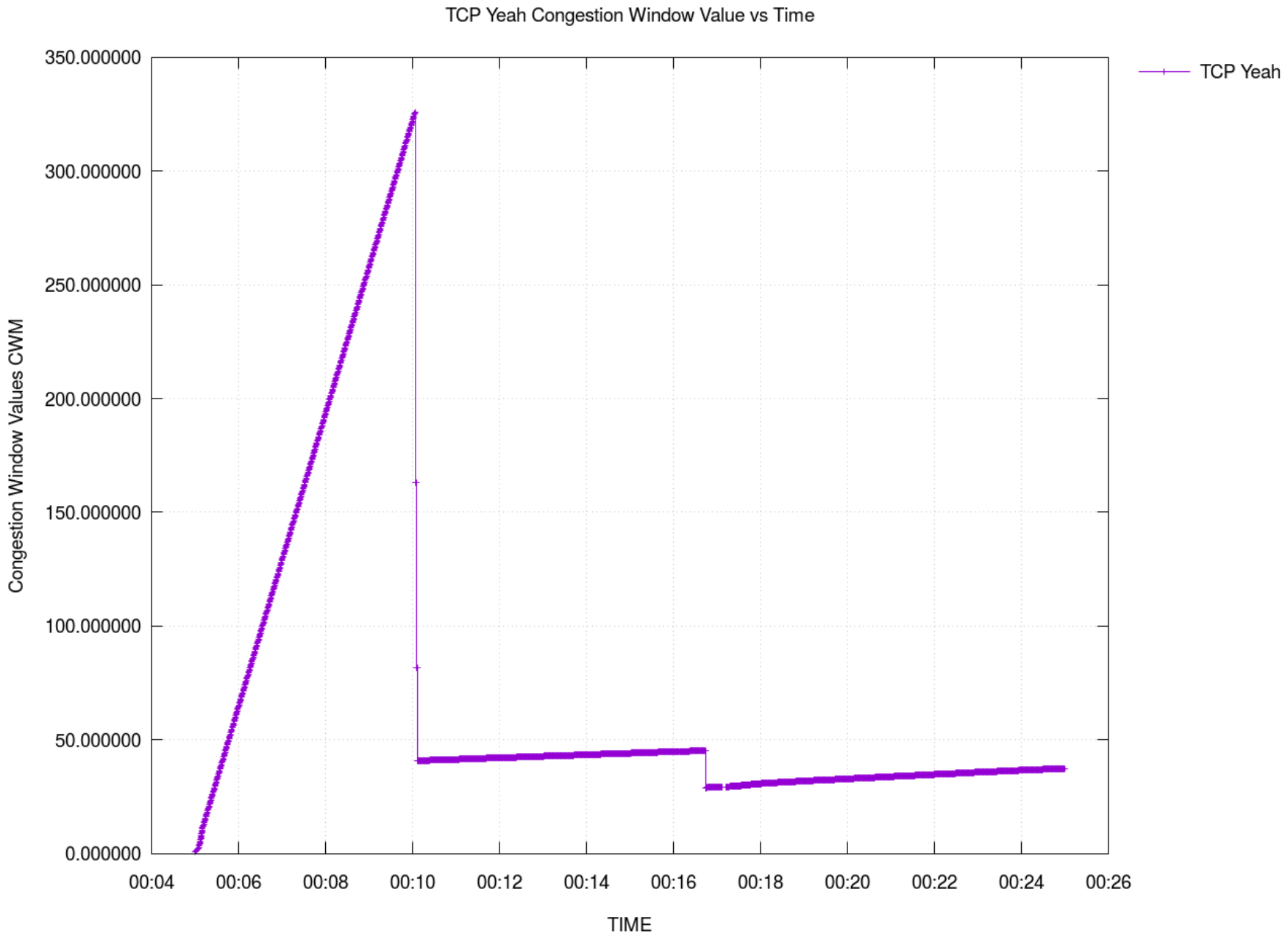
TCP New Reno


TCP New Reno Congestion Window Value vs Time
Congestion Window Values CWM
1.600000
1.400000
1.200000
1.000000
800.000000
600.000000
400.000000
200.000000
0.000000
00:00
00:05
00:10
00:15
00:20
00:25
TIME
TCP New Reno

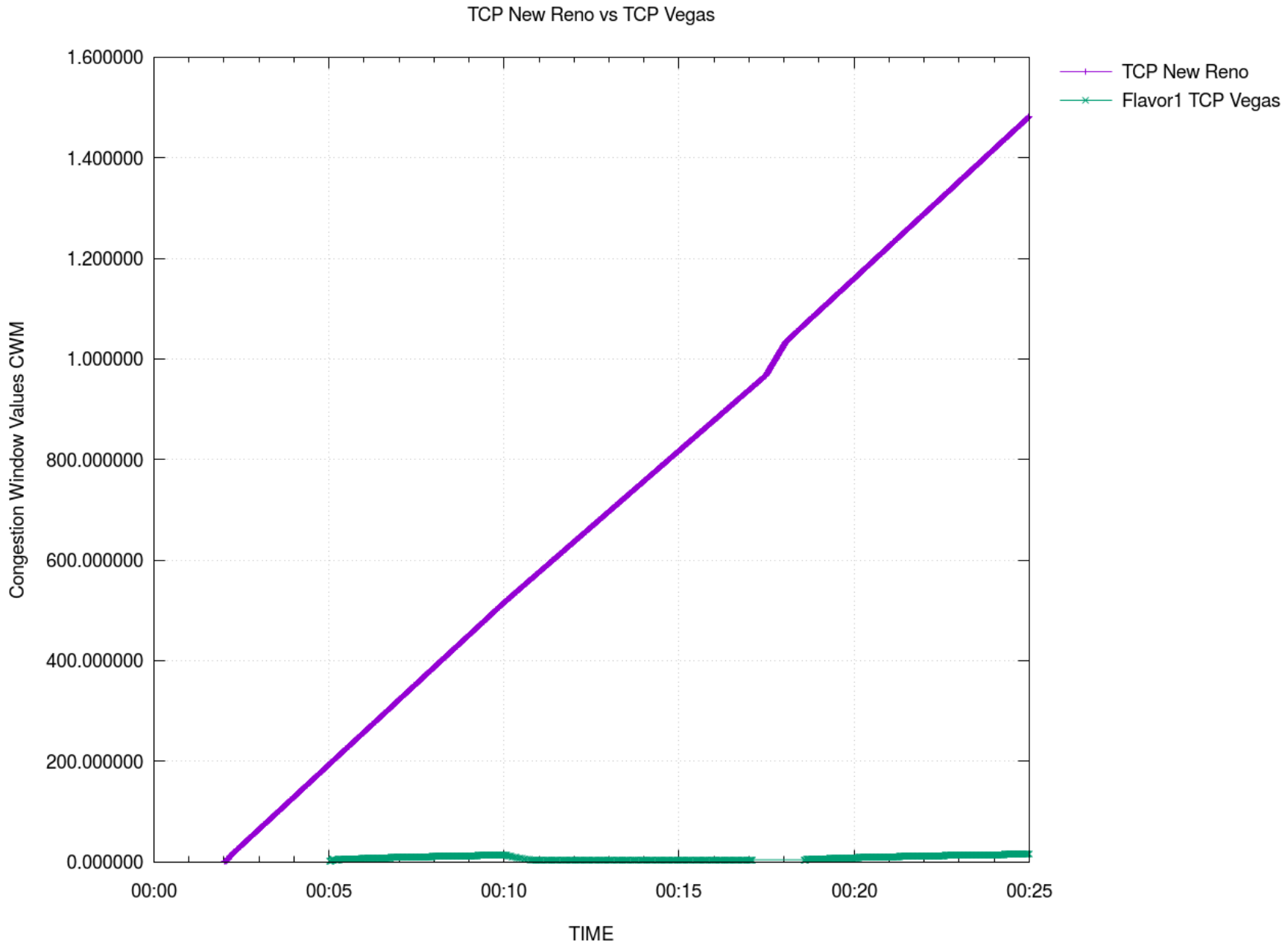
TCP Vegas


TCP Vegas Congestion Window Value vs Time
Congestion Window Values CWM
TIME
TCP Vegas
16.000000
14.000000
12.000000
10.000000
8.000000
6.000000
4.000000
2.000000
0.000000
00:04
00:06
00:08
00:10
00:12
00:14
00:16
00:18
00:20
00:22
00:24
00:26

TCP YeAH


TCP Yeah Congestion Window Value vs Time
Congestion Window Values CWM
TIME
TCP Yeah
350.000000
300.000000
250.000000
200.000000
150.000000
100.000000
50.000000
0.000000
00:04
00:06
00:08
00:10
00:12
00:14
00:16
00:18
00:20
00:22
00:24
00:26

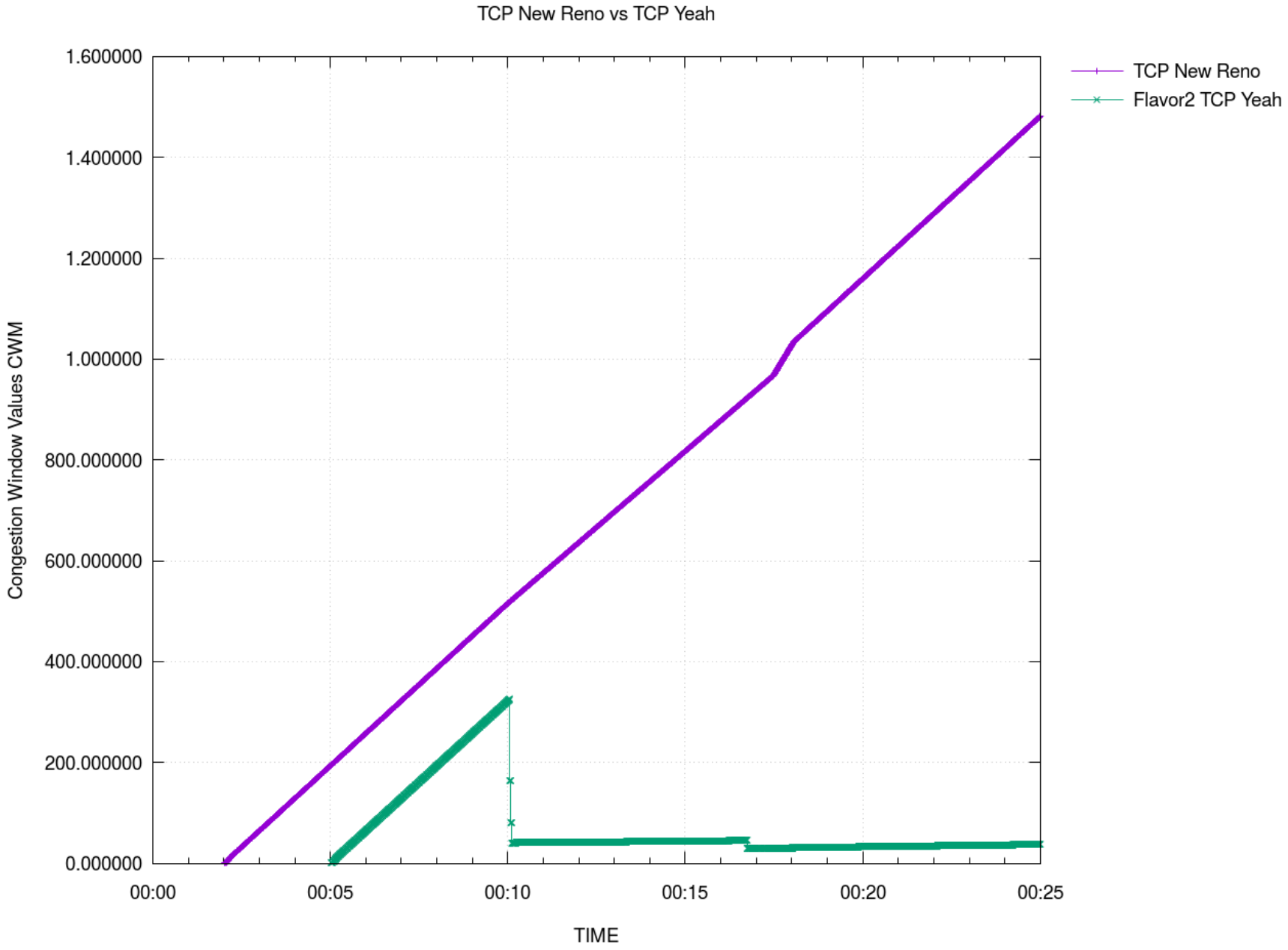
TCP New Reno vs TCP Vegas



TCP New Reno vs TCP Vegas

- NewReno tends to be very steep sloping downwards from left to right having more throughput time than TCP YeAH.
- At congestion window (cwnd) value 1.000000, there is a symptom of slight congestion avoidance at time 17 sec, than again it regained its cwnd value consistently.
- cwnd value of TCP Vegas tends to remain close to 0(zero) from time 05~17 sec, then there is a scenario of data being dropped for about 2 sec.
- This also depicts that there might be a tendency of more packets loss with more packets retransmission and the nodes to be remained busy with fast recovery of data losses.
- TCP Vegas proved to show very poor performance against congestion avoidance.

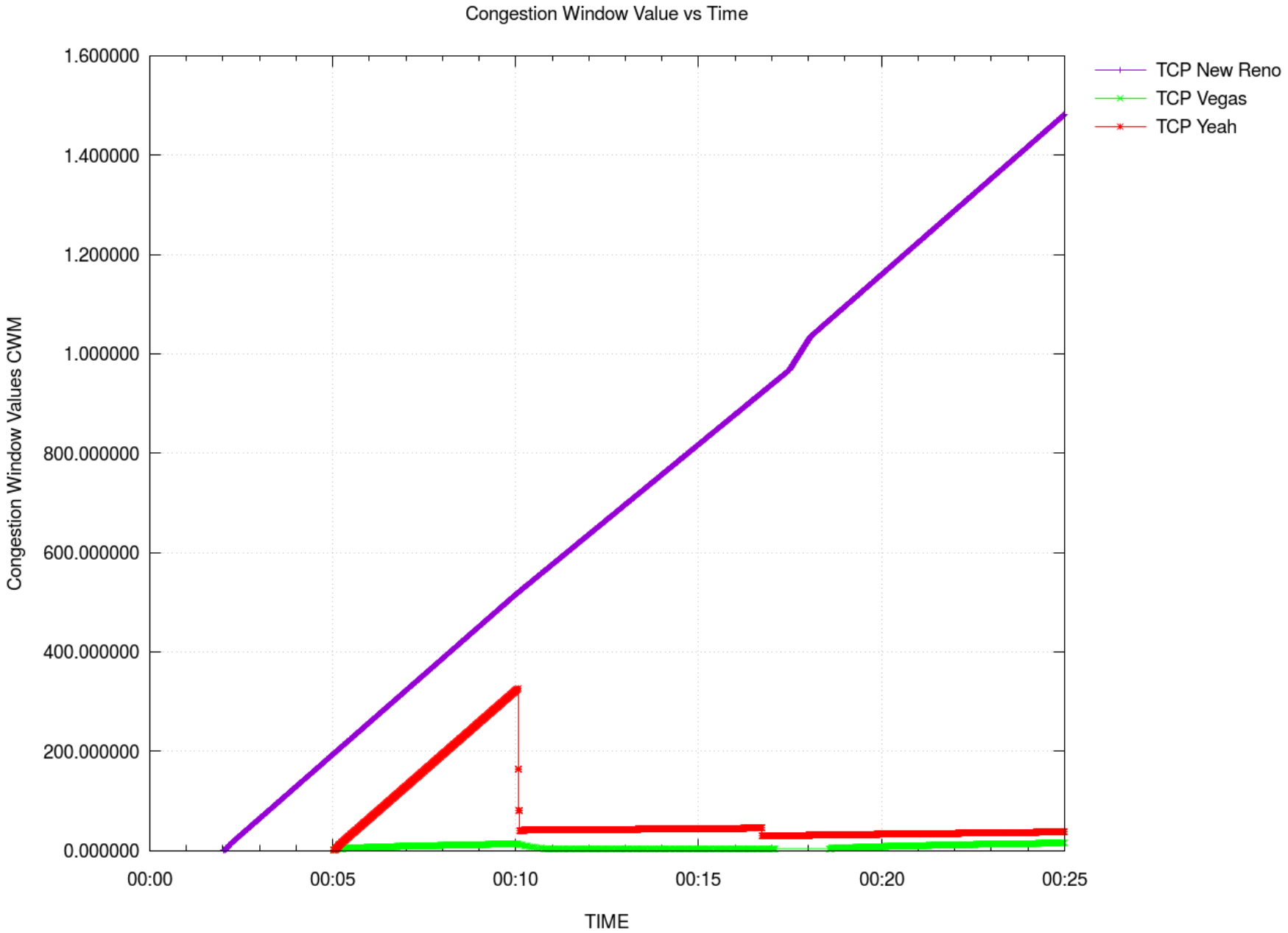
TCP New Reno vs TCP YeAH



TCP New Reno vs TCP YeAH

- TCP NewReno tends to be very steep sloping downwards from right to left having more throughput time than TCP Yeah.
- At cwnd value 1.000000, there is a symptom of slight congestion avoidance at time 17 sec, then again it regained its cwnd value consistently.
- cwnd value of TCP Yeah tends to gain value consistently up to 350.000000 from time 05~10 sec, then it fell drastically to less than 100.000000 at time10 sec.
- The graph tried to regain again but it seemed to stay in the same level depicting that there is more congestion taking longer throughput time and outstanding packets yet to be transferred.
- This also depicts that there might be a tendency of more packets loss with more packets retransmission and the nodes to be remained busy with fast recovery of data losses.

TCP New Reno vs TCP Vegas vs TCP YeAH



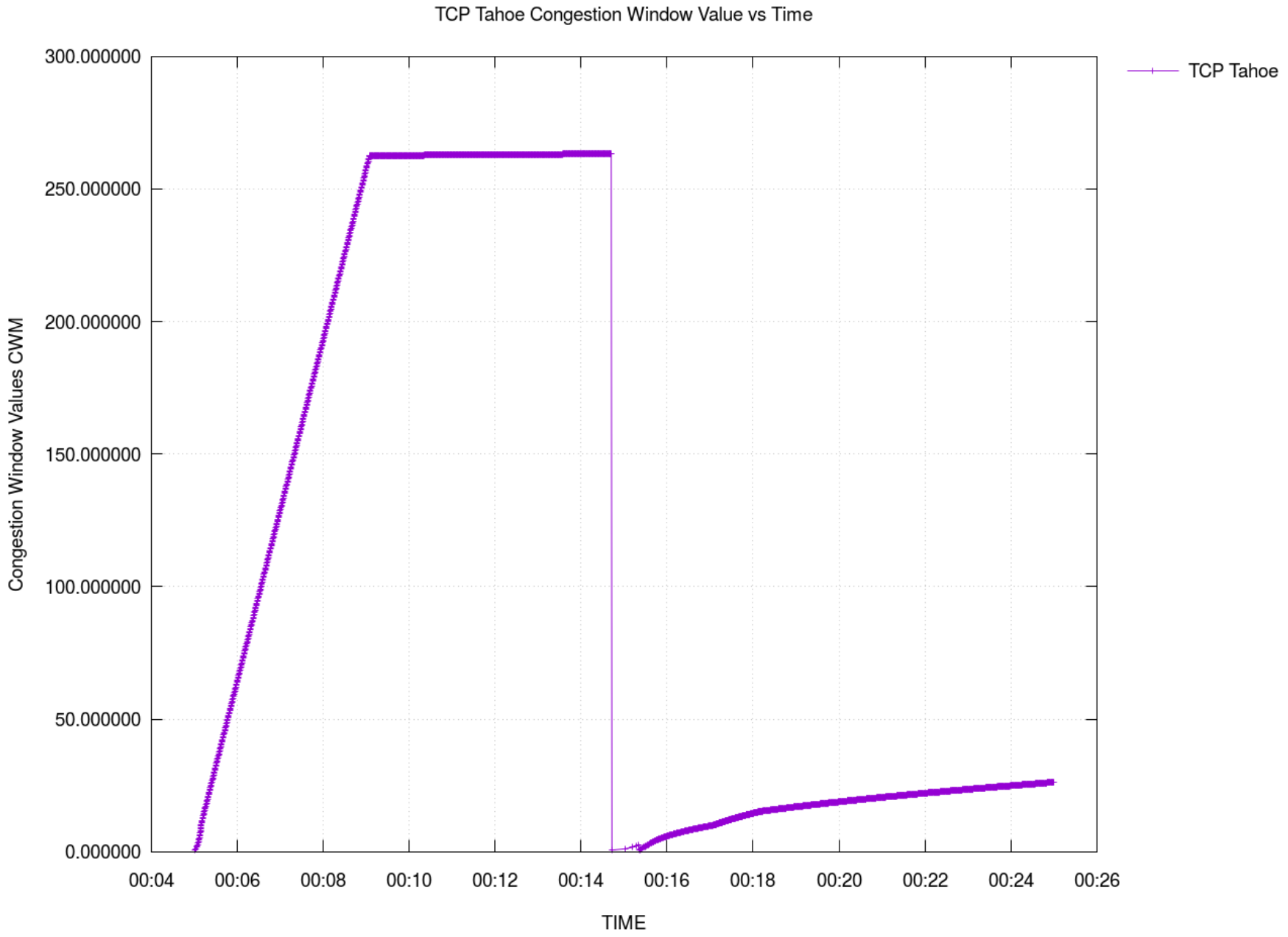
TCP New Reno vs TCP Vegas vs TCP YeAH

- TCP NewReno tends to be very steep sloping downwards from right to left having more throughput time than TCP Yeah.
- At cwnd value 1.000000, there is a symptom of slight congestion avoidance at time 17 sec, than again it regained its cwnd value consistently.
- cwnd value of TCP Vegas tends to remain close to 0(zero) from time 05~17 sec, then there is a scenario of data being dropped for about 2 sec. TCP Vegas proved to show very poor performance against congestion avoidance.
- cwnd value of TCP Yeah tends to gain value consistently up to 350.000000 from time 05~10 sec, then it fell drastically to less than 100.000000 at time10 sec.
- The graph tried to regain again but it seemed to stay in the same level depicting that there is more congestion taking longer throughput time and outstanding packets yet to be transferred.
- There might be a tendency of more packets loss with more packets retransmission and the nodes to be remained busy with fast recovery of data losses.

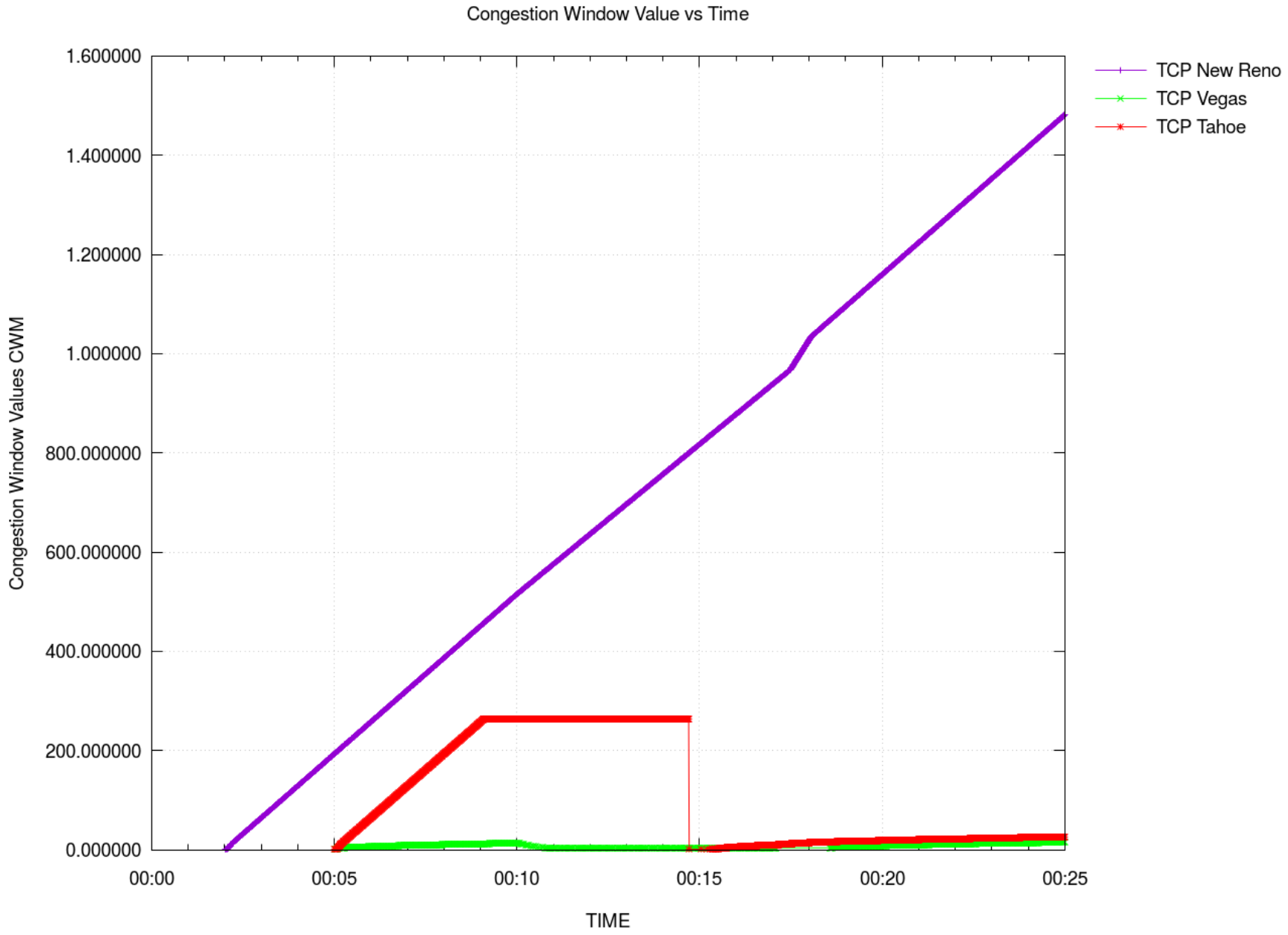
We Found out that

- TCP Vegas does lead to a fair allocation of bandwidth for different delay connections.
- Only TCP YeAH is behaving against long delay connections
- TCP NewReno make some performance improvements to TCP Vegas and YeAH. TCP NewReno achieves higher throughput than Vegas and YeAH for large loss rate.
- TCP Vegas may prove to be better when more than one packet is dropped in one window.TCP Vegas causes much fewer packets retransmissions than TCP NewReno and YeAH.
- TCP NewReno tends to gain its cwnd value aggressively while TCP YeAH tends to be stable and relatively close to 0 (zero).
- when the buffer sizes are small, TCP Vegas performs better than TCP Reno and YeAH, since it does not require much space in switch buffer. As the buffer sizes increase, TCP NewReno and TCP YeAH throughput increase at the cost of a decrease in TCP Vegas throughput.
- It is suggested that a change in Vegas algorithm to make Vegas more aggressive in the competition.
- This may be worthy of further investigation in the future work. However, all the efforts in analysis of queuing algorithms effects lie in the gateway size.
- To conclude,we found TCP NewReno performing the best but the debate for which is better in which aspect is still and open discussion to talk about.

Surprising Factor TCP Tahoe worked on NS 3.24



Surprising Factor



THANK YOU