1.TCP Tahoe:

Code:

```
#include "tutorial-app.h"
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include <fstream>

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.
//
============================================================================
===
//
/**
 * Congestion window change callback
 *
 * \param oldCwnd Old congestion window.
 * \param newCwnd New congestion window.
 */
static void
CwndChange(uint32_t oldCwnd, uint32_t newCwnd)
{
   NS_LOG_UNCOND(Simulator::Now().GetSeconds() << "\t" << newCwnd);
}
/**
 * Rx drop callback
 *
 * \param p The dropped packet.
 */
static void
RxDrop(Ptr<const Packet> p)
{
   NS_LOG_UNCOND("RxDrop at " << Simulator::Now().GetSeconds());
}
int
main(int argc, char* argv[])
{
   CommandLine cmd(__FILE__);
   cmd.Parse(argc, argv);

   // In the following three lines, TCP NewReno is used as the congestion
   // control algorithm, the initial congestion window of a TCP connection is
   // set to 1 packet, and the classic fast recovery algorithm is used. Note
   // that this configuration is used only to demonstrate how TCP parameters
   // can be configured in ns-3. Otherwise, it is recommended to use the default
   // settings of TCP in ns-3.
   Config::SetDefault("ns3::TcpL4Protocol::SocketType", StringValue("ns3::TcpNewReno"));
   Config::SetDefault("ns3::TcpSocket::InitialCwnd", UintegerValue(1));
   Config::SetDefault("ns3::TcpL4Protocol::RecoveryType",
                      TypeIdValue(TypeId::LookupByName("ns3::TcpClassicRecovery")));
```

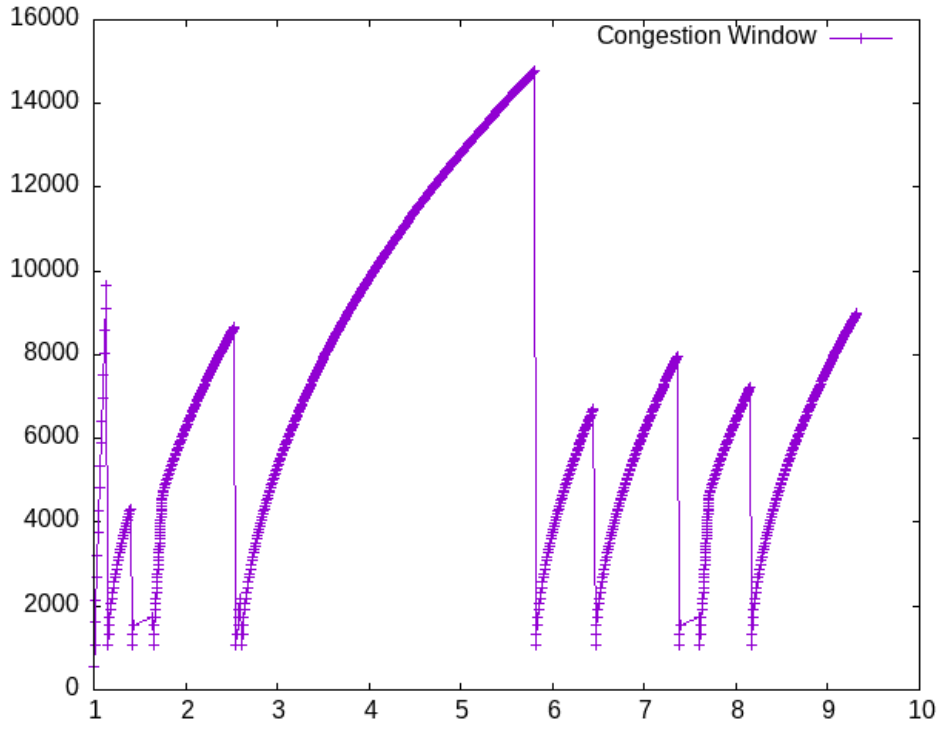
```
  NodeContainer nodes;
  nodes.Create(2);

  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<TutorialApp> app = CreateObject<TutorialApp>();
  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:

./ns3 run examples/tutorial/fifth.cc > fifth.dat 2>&1

```
8.1985  1608
nitt@OptiPlex:~/ns-allinone-3.42/ns-3.42$ gnuplot

        G N U P L O T
        Version 5.4 patchlevel 2    last modified 2021-06-01

        Copyright (C) 1986-1993, 1998, 2004, 2007-2021
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'unknown'
gnuplot> set terminal png size 640,480

Terminal type is now 'png'
Options are 'nocrop enhanced size 640,480 font "arial,12.0" '
gnuplot> set output "tahoe-cwnd.png"
gnuplot> plot "fifth.dat" using 1:2 title 'Congestion Window' with linespoints
gnuplot> exit
```



2.TCP Reno:
Code:

```
#include "tutorial-app.h"

#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/stats-module.h"

#include <fstream>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("SeventhScriptExample");

//
// ============================================================================
===
//
//         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.
//
// NOTE: If this example gets modified, do not forget to update the .png figure
// in src/stats/docs/seventh-packet-byte-count.png
//
================================================================================
//

/**
 * Congestion window change callback
 *
 * \param stream The output stream file.
 * \param oldCwnd Old congestion window.
 * \param newCwnd New congestion window.
 */
static void
CwndChange(Ptr<OutputStreamWrapper> stream, uint32_t oldCwnd, uint32_t newCwnd)
{
   NS_LOG_UNCOND(Simulator::Now().GetSeconds() << "\t" << newCwnd);
   *stream->GetStream() << Simulator::Now().GetSeconds() << "\t" << oldCwnd << "\t" <<
newCwnd
                        << std::endl;
}

/**
 * Rx drop callback
 *
 * \param file The output PCAP file.
 * \param p The dropped packet.
 */
static void
RxDrop(Ptr<PcapFileWrapper> file, Ptr<const Packet> p)
{
   NS_LOG_UNCOND("RxDrop at " << Simulator::Now().GetSeconds());
   file->Write(Simulator::Now(), p);
}
```

```
int
main(int argc, char* argv[])
{
   bool useV6 = false;

   CommandLine cmd(__FILE__);
   cmd.AddValue("useIpv6", "Use Ipv6", useV6);
   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);

   uint16_t sinkPort = 8080;
   Address sinkAddress;
   Address anyAddress;
   std::string probeType;
   std::string tracePath;
   if (!useV6)
   {
      Ipv4AddressHelper address;
      address.SetBase("10.1.1.0", "255.255.255.0");
      Ipv4InterfaceContainer interfaces = address.Assign(devices);
      sinkAddress = InetSocketAddress(interfaces.GetAddress(1), sinkPort);
      anyAddress = InetSocketAddress(Ipv4Address::GetAny(), sinkPort);
      probeType = "ns3::Ipv4PacketProbe";
      tracePath = "/NodeList/*/$ns3::Ipv4L3Protocol/Tx";
   }
   else
   {
      Ipv6AddressHelper address;
```

```
    address.SetBase("2001:0000:f00d:cafe::", Ipv6Prefix(64));
    Ipv6InterfaceContainer interfaces = address.Assign(devices);
    sinkAddress = Inet6SocketAddress(interfaces.GetAddress(1, 1), sinkPort);
    anyAddress = Inet6SocketAddress(Ipv6Address::GetAny(), sinkPort);
    probeType = "ns3::Ipv6PacketProbe";
    tracePath = "/NodeList/*/$ns3::Ipv6L3Protocol/Tx";
  }

  PacketSinkHelper packetSinkHelper("ns3::TcpSocketFactory", anyAddress);
  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());

  Ptr<TutorialApp> app = CreateObject<TutorialApp>();
  app->Setup(ns3TcpSocket, sinkAddress, 1040, 1000, DataRate("1Mbps"));
  nodes.Get(0)->AddApplication(app);
  app->SetStartTime(Seconds(1.));
  app->SetStopTime(Seconds(20.));

  AsciiTraceHelper asciiTraceHelper;
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream("seventh.cwnd");
  ns3TcpSocket->TraceConnectWithoutContext("CongestionWindow",
                                           MakeBoundCallback(&CwndChange, stream));

  PcapHelper pcapHelper;
  Ptr<PcapFileWrapper> file =
    pcapHelper.CreateFile("seventh.pcap", std::ios::out, PcapHelper::DLT_PPP);
  devices.Get(1)->TraceConnectWithoutContext("PhyRxDrop", MakeBoundCallback(&RxDrop,
file));

  // Use GnuplotHelper to plot the packet byte count over time
  GnuplotHelper plotHelper;

  // Configure the plot.  The first argument is the file name prefix
  // for the output files generated.  The second, third, and fourth
  // arguments are, respectively, the plot title, x-axis, and y-axis labels
  plotHelper.ConfigurePlot("seventh-packet-byte-count",
                           "Packet Byte Count vs. Time",
                           "Time (Seconds)",
                           "Packet Byte Count");
```

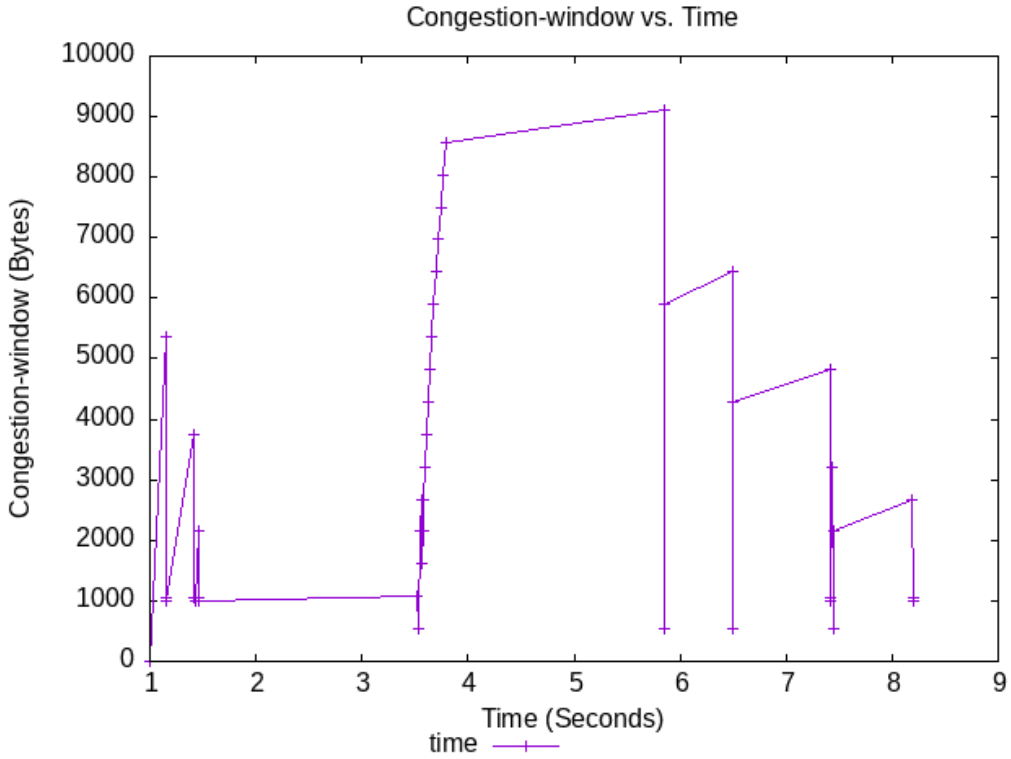
```
    // Specify the probe type, trace source path (in configuration namespace), and
    // probe output trace source ("OutputBytes") to plot.  The fourth argument
    // specifies the name of the data series label on the plot.  The last
    // argument formats the plot by specifying where the key should be placed.
    plotHelper.PlotProbe(probeType,
                         tracePath,
                         "OutputBytes",
                         "Packet Byte Count",
                         GnuplotAggregator::KEY_BELOW);

    // Use FileHelper to write out the packet byte count over time
    FileHelper fileHelper;

    // Configure the file to be written, and the formatting of output data.
    fileHelper.ConfigureFile("seventh-packet-byte-count", FileAggregator::FORMATTED);

    // Set the labels for this formatted output file.
    fileHelper.Set2dFormat("Time (Seconds) = %.3e\tPacket Byte Count = %.0f");

    // Specify the probe type, trace source path (in configuration namespace), and
    // probe output trace source ("OutputBytes") to write.
    fileHelper.WriteProbe(probeType, tracePath, "OutputBytes");

    Simulator::Stop(Seconds(20));
    Simulator::Run();
    Simulator::Destroy();

    return 0;
}
```

seventh.plt

```
set terminal png
set output "seventh-cwnd.png"
set title "Congestion-window vs. Time"
set xlabel "Time (Seconds)"
set ylabel "Congestion-window (Bytes)"

set datafile missing "-nan"
plot "seventh.cwnd" index 0 title "time" with linespoints, "seventh.cwnd" index 1 title "cwnd" with
linespoints
```

Output:

```
./ns3 run examples/tutorial/seventh.cc > seventh.cwnd 2>&1
gnuplot seventh.plt
```


Congestion-window vs. Time
Congestion-window (Bytes)
Time (Seconds)
time
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
1
2
3
4
5
6
7
8
9