NS3 and TCP Congestion Control

TAs: Akanksha and Sumit

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

TCP Communication

There are three important abstract base classes:

- class TcpSocket: This is defined in src/internet/model/tcp-socket.{cc,h}. This class exists for hosting TcpSocket attributes that can be reused across different implementations. For instance, the attribute InitialCwnd can be used for any of the implementations that derive from class TcpSocket.
- class TcpSocketFactory: This is used by the layer-4 protocol instance to create TCP sockets of the right type.
- class TcpCongestionOps: This supports different variants of congestion control– a key topic of simulation-based TCP research.

TCP Communication for Receiver

Bind() - Bind the socket to an address, or to a general endpoint.

Bind6() - Same as Bind(), but for IPv6.

BindToNetDevice() - Bind the socket to the specified NetDevice, creating a general endpoint.

Listen() - Listen on the endpoint for an incoming connection. When an incoming request for connection is detected (i.e. the other peer invoked Connect()) the application will be signaled with the callback NotifyConnectionRequest (set in SetAcceptCallback() beforehand). If the connection is accepted (the default behavior, when the associated callback is a null one) the Socket will fork itself, i.e. a new socket is created to handle the incoming data/connection, in the state SYN_RCVD.

TCP Communication for Receiver

ShutdownSend()

Signal a termination of send, or in other words prevents data from being added to the buffer. After this call, if buffer is already empty, the socket will send a FIN, otherwise FIN will go when buffer empties.

Recv()

Grab data from the TCP socket. TCP is a stream socket, and it is allowed to concatenate multiple packets into bigger ones.

RecvFrom()

Same as Recv, but with the source address as parameter.

TCP Communication for sender

Connect() - Set the remote endpoint, and try to connect to it. The TCP then will be in the SYN_SENT state. If a SYN-ACK is received, the TCP will setup the congestion control, and connection is established.

GetTxAvailable() - Return the amount of data that can be stored in the TCP Tx buffer.

Send()/SendTo() - Send the data into the TCP Tx buffer.

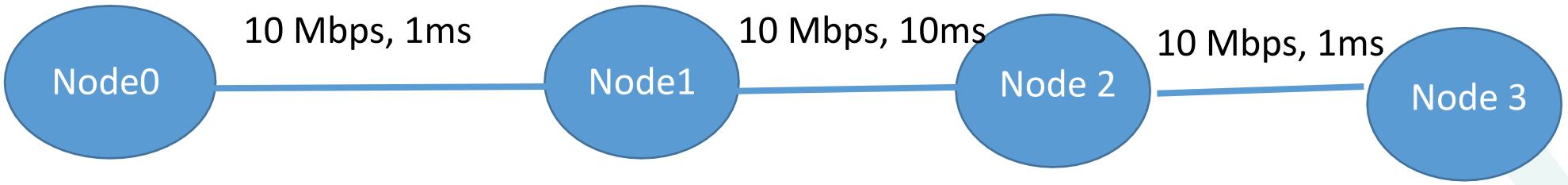
Close() - Terminate the local side of the connection, by sending a FIN (after all data in the tx buffer has been transmitted). This does not prevent the socket in receiving data, and employing retransmit mechanism if losses are detected.

Network Topology

n0 ---------- n1 ---------- n2 ---------- n3

10 Mbps 1 Mbps 10 Mbps

1 ms 10 ms 1 ms



```
uint32_t stream = 1;
std::string socketFactory = "ns3::TcpSocketFactory";
std::string tcpTypeId = "ns3::TcpLinuxReno";
std::string qdiscTypeId = "ns3::FifoQueueDisc";
bool isSack = true;
uint32_t delAckCount = 1;
std::string recovery = "ns3::TcpClassicRecovery";
```

```
// Create nodes
NodeContainer leftNodes;
NodeContainer rightNodes;
NodeContainer routers;
routers.Create(2);
leftNodes.Create(1);
rightNodes.Create(1);

std::vector<NetDeviceContainer> leftToRouter;
std::vector<NetDeviceContainer> routerToRight;

// Create the point-to-point link helpers and connect two router nodes
PointToPointHelper pointToPointRouter;
pointToPointRouter.SetDeviceAttribute("DataRate", StringValue("1Mbps"));
pointToPointRouter.SetChannelAttribute("Delay", StringValue("10ms"));
NetDeviceContainer r1r2ND = pointToPointRouter.Install(routers.Get(0), rou
```

```
// Function to trace change in cwnd at n0
static void
CwndChange(uint32_t oldCwnd, uint32_t newCwnd)
{
    fPlotCwnd << Simulator::Now().GetSeconds() << " " << newCwnd / segmentSize << std::endl;
}
```

```
// Trace Function for cwnd
void
TraceCwnd(uint32_t node, uint32_t cwndWindow, Callback<void, uint32_t, uint32_t> CwndTrace)
{
    Config::ConnectWithoutContext("/NodeList/" + std::to_string(node) +
                                  "/$ns3::TcpL4Protocol/SocketList/" +
                                  std::to_string(cwndWindow) + "/CongestionWindow",
                                  CwndTrace);
}
```

```
// Function to install sink application
void
InstallPacketSink(Ptr<Node> node, uint16_t port, std::string socketFactory)
{
    PacketSinkHelper sink(socketFactory, InetSocketAddress(Ipv4Address::GetAny(), port));
    ApplicationContainer sinkApps = sink.Install(node);
    sinkApps.Start(Seconds(10.0));
    sinkApps.Stop(stopTime);
}
```

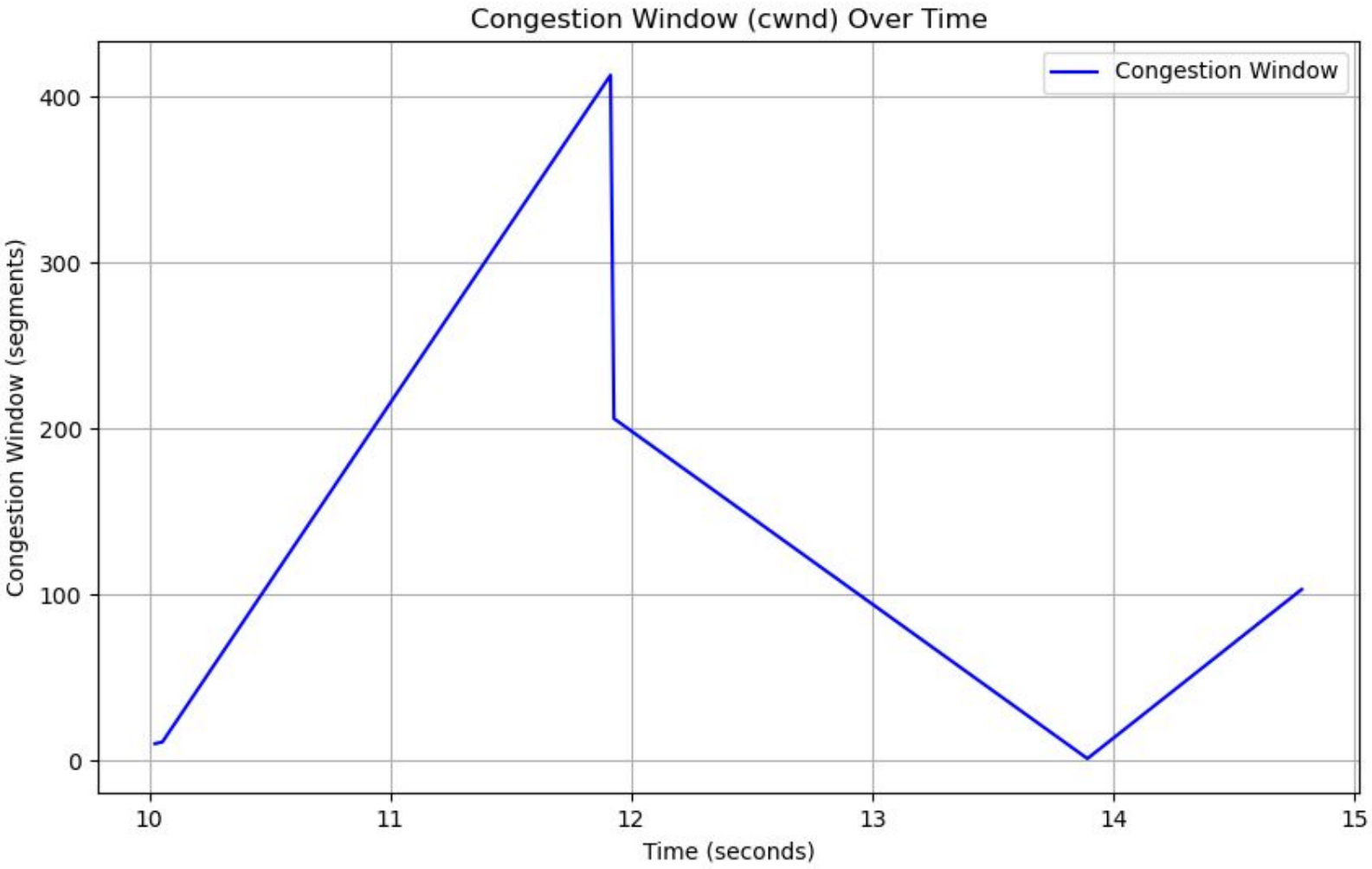
Traces

```
1    10.0251 10
2    10.0552 11
3    10.0598 12
4    10.0645 13
5    10.0691 14
6    10.0737 15
7    10.0783 16
8    10.083 17
9    10.0876 18
10   10.0922 19
11   10.0968 20
12   10.1015 21
13   10.1061 22
14   10.1107 23
15   10.1153 24
16   10.12 25
17   10.1246 26
18   10.1292 27
19   10.1338 28
20   10.1385 29
21   10.1431 30
22   10.1477 31
23   10.1523 32
24   10.157 33
```

```
397    11.8817 406
398    11.8863 407
399    11.891 408
400    11.8956 409
401    11.9002 410
402    11.9048 411
403    11.9094 412
404    11.9141 413
405    11.9282 206
406    13.8924 1
407    14.781 103
```

DEMO

TCP Reno Congestion Window



How would the graph look for TCP Tahoe?

Resources

https://www.nsnam.org/docs/models/html/tcp.html#overview-of-support-for-tcp

https://www.nsnam.org/docs/tutorial/html/tracing.html#background