

Session 4. Low-Level Tracing 과 TCP/UDP/IP

황선욱

Multimedia & Wireless Networking Laboratory, SNU
swhwang@mwnl.snu.ac.kr

Contents

- Low-Level Tracing
- Transport Layer Model
- TCP Congestion Window Tracing Example

Low-Level Tracing

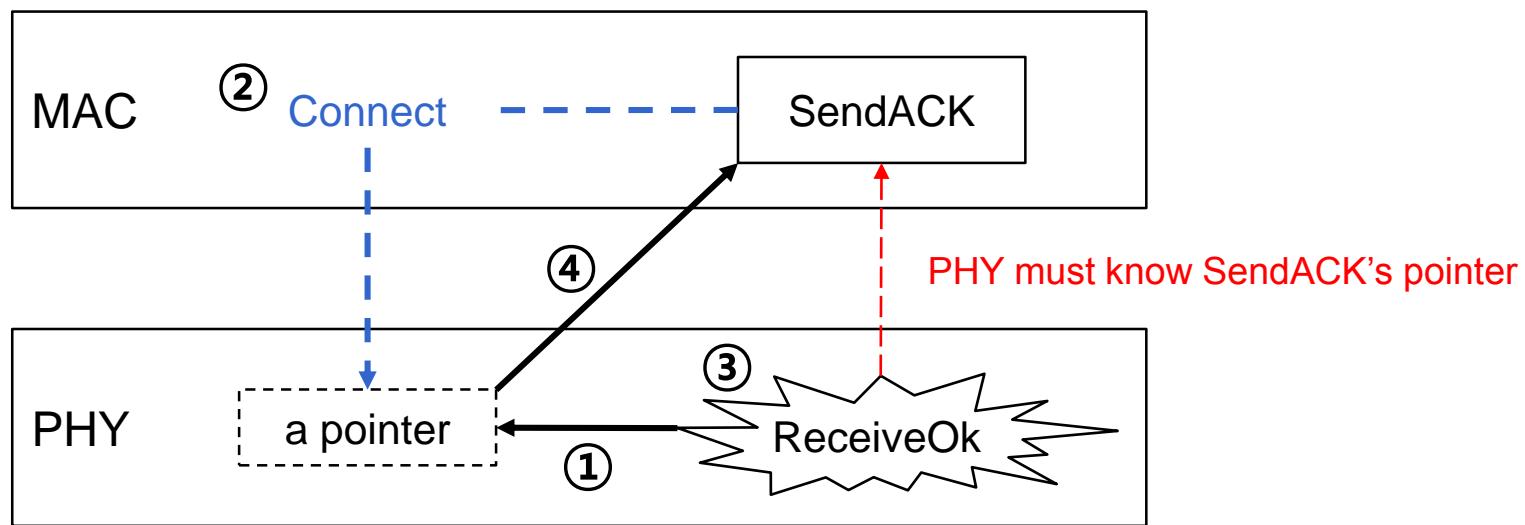
Tracing Overview

- Concept: Independent tracing sources and tracing sinks along with a uniform mechanism for connecting sources to sinks
- **Trace sources**
 - Entities that can signal events and provide access to interesting data
 - Ex) Indicate when a packet is received
Indicate when an interesting state change happens
- **Trace sinks**
 - Entities that consume trace information
- Simulator provides a set of pre-configured trace sources
- Trace source is a kind of point-to-multipoint information link
 - One trace source can be connected by several trace sinks

Callback and Low-Level Tracing

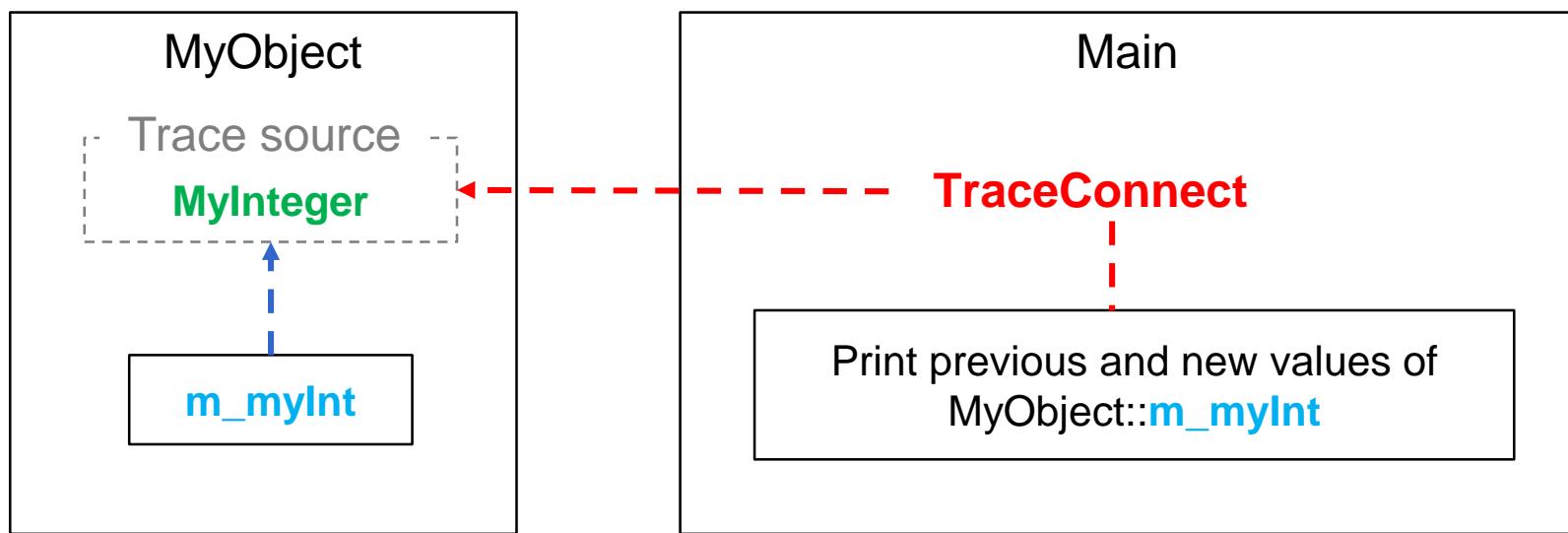
Callback

- Allow a piece of code to call a function without inter-module dependency
- Use a pointer-to-function
- Decouple the calling function from the called class completely



Callback and Low-Level Tracing

- Relation between tracing system and callback
 - A trace source is a variable that holds a list of callbacks
 - A trace sink is a function used as the target of a callback
 - When a trace sink wants to know information given by a trace source, it adds its own function to the callback list



Low-level Tracing Example (1/2)

```
class MyObject : public Object
{
public:
    static TypeId GetTypeId (void)
    {
        static TypeId tid = TypeId ("MyObject")
            .SetParent<Object> ()
            .SetGroupName ("Tutorial")
            .AddConstructor<MyObject> ()
            .AddTraceSource ("MyInteger",
                "An integer value to trace.",
                MakeTraceSourceAccessor (&MyObject::m_myInt));
        return tid;
    }
    MyObject () {}
    TracedValue<int32_t> m_myInt;
};
```

scratch/s4_ex1.cc

Provides the “hooks” used for connecting the trace source to the outside the config system

Provides the infrastructure that overloads the operators and drives callback process

Low-level Tracing Example (2/2)

```
/* Trace sink:  
 * this function will be called whenever  
 * the overloaded operators of the TracedValue is excuted */  
  
void  
IntTrace (int32_t oldValue, int32_t newValue)  
{  
    std::cout << "Traced " << oldValue << " to " << newValue << std::endl;  
}
```

```
int  
main (int argc, char *argv[])  
{  
    Ptr<MyObject> myObject = CreateObject<MyObject> ();  
    myObject->TraceConnectWithoutContext ("MyInteger",  
                                         MakeCallback (&IntTrace));  
    myObject->m_myInt = 1234;  
}
```

Connects a trace sink to a trace source

Operator "=" invoke the Callback

Config Subsystem Tracing

- Config path (the context)
 - Path of predefined trace source
 - Represents a chain of Object pointers
 - Ex) / NodeList/7/\$ns3::MobilityModel/CourseChange
- Config subsystem is used to allow selecting a trace source in the config path
 - `void ns3::Config::Connect(std::string path,
const CallbackBase &cb)`

How to Find and Connect Trace Sources

- I. How to find out what trace sources are available
- II. How to find reference usage of a trace source
- III. How to connect the trace source

1. Available Trace Sources

- ns-3 Doxygen (<https://www.nsnam.org/doxygen/>)

The screenshot shows the ns-3 Doxygen documentation interface. The top navigation bar includes the ns-3 logo, the text "A Discrete-Event Network Simulator ns-3-dev @ f6c49ef(+)", and links for HOME, TUTORIALS, DOCS, and DEVELOP. The left sidebar has a tree view under "ns-3" with nodes like ns-3 Documentation, All Attributes, All GlobalValues, All LogComponents, All TraceSources (which is selected), Todo List, Deprecated List, and Bug List. It also has sections for Modules, Namespaces, Classes, and Files. The main content area is titled "ns3::TcpSocketBase" and lists several attributes:

- RTO: Retransmission timeout
- RTT: Last RTT sample
- NextTxSequence: Next sequence number to send (SND.NXT)
- HighestSequence: Highest sequence number ever sent in socket's life time
- State: TCP state
- CongState: TCP Congestion machine state
- EcnState: Trace ECN state change of socket
- AdvWND: Advertised Window Size
- RWND: Remote side's flow control window
- BytesInFlight: Socket estimation of bytes in flight
- HighestRxSequence: Highest sequence number received from peer
- HighestRxAck: Highest ack received from peer
- CongestionWindow: The TCP connection's congestion window
- CongestionWindowInflated: The TCP connection's congestion window inflates as in older RFC
- SlowStartThreshold: TCP slow start threshold (bytes)

2. Finding Trace Source Usage

- Find existing implementation within example codes
- Using find & grep command
 - To find existing codes which contain “CongestionWindow” and “Connect”
 - `find . -name '*.cc' | xargs grep CongestionWindow | grep Connect`
 - Output results:

```
skim11@sim7-OptiPlex-9020:~/lecture/ns-allinone-3.26/ns-3.26$ find . -name '*.cc' | xargs grep CongestionWindow | grep Connect
./examples/tcp/tcp-variants-comparison.cc: Config::ConnectWithoutContext ("/ NodeList/1/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow");
./examples/tcp/tcp-large-transfer.cc: Config::ConnectWithoutContext ("/ NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow",
./examples/tutorial/seventh.cc: ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeBoundCallback (&CwndChange, stream)
./examples/tutorial/sixth.cc: ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeBoundCallback (&CwndChange, stream));
./examples/tutorial/fifth.cc: ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));
./src/internet/model/tcp-socket-base.cc: ok = m_tcb->TraceConnectWithoutContext ("CongestionWindow",
./src/internet/model/tcp-socket-base.cc: ok = m_tcb->TraceConnectWithoutContext ("CongestionWindow",
./src/internet/test/tcp-general-test.cc: m_senderSocket->TraceConnectWithoutContext ("CongestionWindow",
./src/traffic-control/examples/codel-vs-pfifo-asymmetric.cc: Config::ConnectWithoutContext ("/ NodeList/0/$ns3::TcpL4Protocol/Soc
./src/traffic-control/examples/codel-vs-pfifo-basic-test.cc: Config::ConnectWithoutContext ("/ NodeList/1/$ns3::TcpL4Protocol/Soc
./src/test/ns3tcp/ns3tcp-cwnd-test-suite.cc: ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&Ns3TcpCwnd)
./src/test/ns3tcp/ns3tcp-cwnd-test-suite.cc: ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&Ns3TcpCwnd)
./scratch/s5_ex2.cc: //onOffAppSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));
./scratch/s5_ex2.cc: //ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));
./scratch/s5_ex2.cc: //ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));
./scratch/s5_ex2.cc: Config::ConnectWithoutContext ("/ NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow", MakeCallback (
```

3. Connecting Trace Source

- ConnectWithoutContext
 - void ConnectWithoutContext (std::string **path**, const CallbackBase & **cb**)
 - **path** a config path to match trace sources
 - **cb** the callback to connect to the matching trace sources
 - To find all trace sources which match the input path and will connect the input callback to them
- Connect
 - void Connect (std::string **path**, std::string **context**, const CallbackBase & **cb**)
 - **path** a config path to match trace sources
 - **cb** the callback to connect to the matching trace sources
 - To find all trace sources which match the input path and will connect the input callback to them in such a way that the callback will receive an extra context string upon trace event notification

Tracing Example: Trace Value (1/2)

- Trace source
 - **m_sent** in UdpEchoClient: Counter for sent packets

UdpEchoClient.h

```
#include "ns3/traced-value.h"  
...  
  
uint32_t m_dataSize;  
uint8_t *m_data;  
  
TracedValue<uint32_t> m_sent;  
Ptr<Socket> m_socket;  
Address m_peerAddress;
```

UdpEchoClient.cc

```
.AddTraceSource ("SentPacket", "The number of transmitted packets",  
                MakeTraceSourceAccessor (&UdpEchoClient::m_sent))
```

Tracing Example: Trace Value (2/2)

- Trace sink

```
static void  
Counter (uint32_t prev, uint32_t now)  
{  
    NS_LOG_UNCOND ("prev: " << prev << " now: " << now << " packets at "  
                  << Simulator::Now ().GetSeconds ());  
}
```

- Trace connector

```
clientApps.Get (0) ->TraceConnectWithoutContext ("SentPacket",  
                                                MakeCallback (&Counter));
```

Tracing Example: Trace Function (1/3)

- Trace source
 - **Tx** in UdpEchoClient: Callback for tracing the packet Tx events

UdpEchoClient.h

```
#include "ns3/traced-callback.h"  
...  
class UdpEchoClient : public Application  
{  
    ...  
    TracedCallback<Ptr<const Packet> > m_txTrace;
```

UdpEchoClient.cc

```
.AddTraceSource ("Tx", "A new packet is created and is sent",  
                MakeTraceSourceAccessor (&UdpEchoClient::m_txTrace),  
                "ns3::Packet::TracedCallback")  
...  
void UdpEchoClient::Send (void) {  
    ...  
    m_txTrace (p);  
    m_socket->Send (p);  
    ...
```

Tracing Example: Trace Function (2/3)

- Trace sinks

```
/* Trace sink without context */
static void
Transmit (Ptr<const Packet> p)
{
    NS_LOG_UNCOND ("Tx at " << Simulator::Now ().GetSeconds ());
}

/* Trace sink with context */
static void
Transmit2 (std::string context, Ptr<const Packet> p)
{
    NS_LOG_UNCOND (context);
    NS_LOG_UNCOND ("Tx at " << Simulator::Now ().GetSeconds ());
}
```

Tracing Example: Trace Function (3/3)

- Trace connector

```
/* Trace connect using pointer */
clientApps.Get(0)->TraceConnectWithoutContext ("Tx",
                                                MakeCallback (&Transmit));
// nodes.Get(0)->GetApplication(0)->TraceConnectWithoutContext ("Tx",
                                                MakeCallback (&Transmit));
clientApps.Get(0)->TraceConnect ("Tx", "Now Tx", MakeCallback (&Transmit2));
```

```
/* Trace connect using config subsystem */
std::ostringstream oss;
oss << "/NodeList/0/ApplicationList/0/$ns3::UdpEchoClient/Tx";
// oss << "/NodeList/*ApplicationList/*/$ns3::UdpEchoClient/Tx";
// oss << "/NodeList/"<<nodes.Get(0)->GetId()
    << "/ApplicationList/*/$ns3::UdpEchoClient/Tx";
Config::ConnectWithoutContext (oss.str(), MakeCallback(&Transmit));
// Config::Connect (oss.str(), MakeCallback(&Transmit2));
```

Transport Layer Model in ns-3

TCP Models in ns-3

- ns-3 was written to support multiple TCP implementations
- Two important abstract base classes
 - Class [TcpSocket](#)
 - Base class of all TcpSockets
 - Only hosts TcpSocket attributes that can be reused across different implementations.
 - Class [TcpSocketFactory](#)
 - Used by applications to create TCP sockets
 - Holds global default variables used to initialize newly created sockets

TcpSocket Class Attributes

- SndBufSize: TcpSocket maximum transmit buffer size (bytes)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint32_t](#) / Initial value: 131072
- RcvBufSize: TcpSocket maximum receive buffer size (bytes)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint32_t](#) / Initial value: 131072
- SegmentSize: TCP maximum segment size (bytes)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint32_t](#) / Initial value: 536
- InitialSlowStartThreshold: TCP slow start threshold (bytes)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint32_t](#) / Initial value: 65535
- InitialCwnd: TCP initial congestion window size (segments)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint32_t](#) / Initial value: 1

... More in ns-3 Doxygen

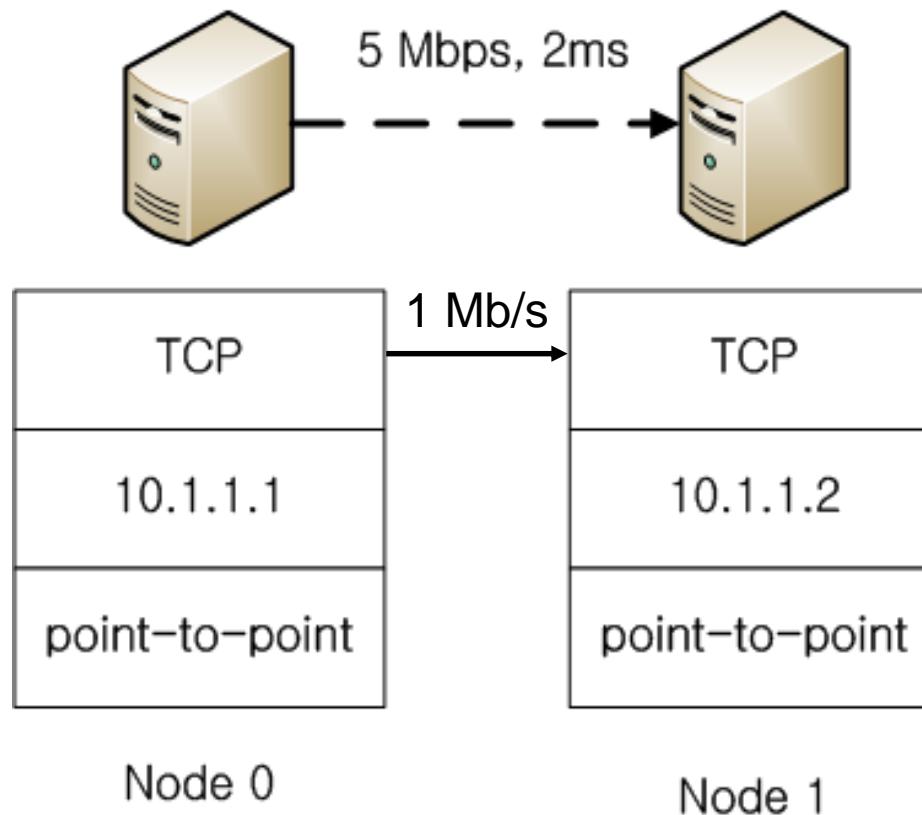
UDP Models in NS-3

- Two important abstract base classes
 - Class [UdpSocket](#)
 - Base class of all UdpSockets
 - Class [UdpSocketFactory](#)
 - Used by applications to create UDP sockets.
- UdpSocket Class Attributes
 - RcvBufSize: [UdpSocket](#) maximum receive buffer size (bytes)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint32_t](#) / Initial value: 131072
 - IpTtl: socket-specific TTL for unicast IP packets (if non-zero)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint8_t](#) / Initial value: 0
 - IpMulticastTtl: socket-specific TTL for multicast IP packets (if non-zero)
 - Set with class: [ns3::UIntegerValue](#)
 - Underlying type: [uint8_t](#) / Initial value: 0

... More in ns-3 Doxygen

TCP Congestion Window Tracing Example

Topology



CongestionWindow Trace Source

- Which class contains CongestionWindow as a trace source?

The screenshot shows the ns-3 documentation website. The top navigation bar includes the ns-3 logo, the text "A Discrete-Event Network Simulator ns-3-dev @ f6c49ef(+)", and links for HOME, TUTORIALS, DOCS, and DEVELOP. Below the navigation bar is a menu with options: Main Page, Related Pages, Modules, Namespaces, Classes, and Files. The "All TraceSources" option under the Modules menu is highlighted. The main content area displays the "ns3::TcpSocketBase" class page. On the left, there is a sidebar with links for ns-3 Documentation, All Attributes, All GlobalValues, All LogComponents, All TraceSources (which is selected), Todo List, Deprecated List, Bug List, and links for Modules, Namespaces, Classes, and Files. The right side lists various attributes of the ns3::TcpSocketBase class, including RTO, RTT, NextTxSequence, HighestSequence, State, CongState, EcnState, AdvWND, RWND, BytesInFlight, HighestRxSequence, HighestRxAck, CongestionWindow (which is highlighted in blue), CongestionWindowInflated, and SlowStartThreshold.

- RTO: Retransmission timeout
- RTT: Last RTT sample
- NextTxSequence: Next sequence number to send (SND.NXT)
- HighestSequence: Highest sequence number ever sent in socket's life time
- State: TCP state
- CongState: TCP Congestion machine state
- EcnState: Trace ECN state change of socket
- AdvWND: Advertised Window Size
- RWND: Remote side's flow control window
- BytesInFlight: Socket estimation of bytes in flight
- HighestRxSequence: Highest sequence number received from peer
- HighestRxAck: Highest ack received from peer
- CongestionWindow: The TCP connection's congestion window
- CongestionWindowInflated: The TCP connection's congestion window inflates as in older RFC
- SlowStartThreshold: TCP slow start threshold (bytes)

What Script to Use?

- Find the proper example
 - find . -name '*.cc' | xargs grep CongestionWindow | grep Connect

```
skim11@sim7-OptiPlex-9020:~/lecture/ns-allinone-3.26/ns-3.26$ find . -name '*.c  
c' | xargs grep CongestionWindow | grep Connect  
.examples/tcp/tcp-variants-comparison.cc: Config::ConnectWithoutContext ("No  
st/0/CongestionWindow", MakeCallback (&Cwn  
Connecting using Config path  
.examples/tcp/tcp-large-transfer.cc: Config::ConnectWithoutContext ("NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow", MakeCallback (&CwndTrac  
er));  
.examples/tutorial/seventh.cc: ns3TcpSocket->TraceConnectWithoutContext ("Cong  
estionWindow", MakeBoundCallback (&CwndChange, stream));  
TcpSocket->TraceConnectWithoutContext ("Cong  
e CwndChange, stream));
```

Connecting using Pointer

```
.examples/tutorial/fifth.cc: ns3TcpSocket->TraceConnectWithoutContext ("Cong  
estionWindow", MakeCallback (&CwndChange));  
.src/internet/model/tcp-socket-base.cc: ok = m_tcb->TraceConnectWithoutContex  
t ("CongestionWindow",  
.src/internet/model/tcp-socket-base.cc: ok = m_tcb->TraceConnectWithoutContex  
t ("CongestionWindow",  
.src/internet/test/tcp-general-test.cc: m_senderSocket->TraceConnectWithoutCo  
ntext ("CongestionWindow",
```

Key Lines in Example Code

- Trace sinks

```
Static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
    NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}

Static void
RxDrop (Ptr<const Packet> p)
{
    NS_LOG_UNCOND ("RxDrop at" << Simulator::Now ().GetSeconds ());
}
```

- Connect a trace source to the trace sinks

```
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",
                                            MakeCallback (&CwndChange));

devices.Get (1)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback (&RxDrop));
```

Example Code (1/4)

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

scratch/s4_ex2.cc

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("s4_ex2");
```

```
static void
CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
{
    NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
}
```

Trace sinks

```
static void
RxDrop (Ptr<const Packet> p)
{
    NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
}
```

Example Code (2/4)

```
int
main (int argc, char *argv[])
{
    // Create network topology
    NodeContainer nodes;
    nodes.Create (2);

    PointToPointHelper pointToPoint;
    pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
    pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

    NetDeviceContainer devices;
    devices = pointToPoint.Install (nodes);

    // Define an error model
    Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
    em->SetAttribute ("ErrorRate", DoubleValue (0.00001));
    devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
```

Example Code (3/4)

```
// Install Internet stack and assign IP addresses
```

```
InternetStackHelper stack;  
stack.Install (nodes);  
Ipv4AddressHelper address;  
address.SetBase ("10.1.1.0", "255.255.255.252");  
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

```
// Implement TCP sink application */
```

```
uint16_t sinkPort = 8080;  
Address sinkAddress (InetSocketAddress (interfaces.GetAddress (1), sinkPort));  
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", InetSocketAddress  
    (Ipv4Address::GetAny (), sinkPort));  
ApplicationContainer sinkApp = packetSinkHelper.Install (nodes.Get (1));  
sinkApp.Start (Seconds (0.));  
sinkApp.Stop (Seconds (10.));
```

```
devices.Get (1)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback (&RxDrop));
```

Connect RxDrop trace source and sink

Example Code (4/4)

// Implement TCP source application

```
OnOffHelper onoff("ns3::TcpSocketFactory", sinkAddress);
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=1]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute("DataRate", DataRateValue(1000000));
ApplicationContainer sourceApp = onoff.Install(nodes.Get (0));
sourceApp.Start (Seconds (1.));
sourceApp.Stop (Seconds (10.));

Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0), TcpSocketFactory::GetTypeId ());
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback
(&CwndChange));
nodes.Get (0)->GetApplication (0)->GetObject<OnOffApplication> ()->SetSocket (ns3TcpSocket);

Simulator::Stop (Seconds (10));
Simulator::Run ();
Simulator::Destroy ();
return 0;
}
```

Connect RxDrop trace source and sink

Example Result (1/2)

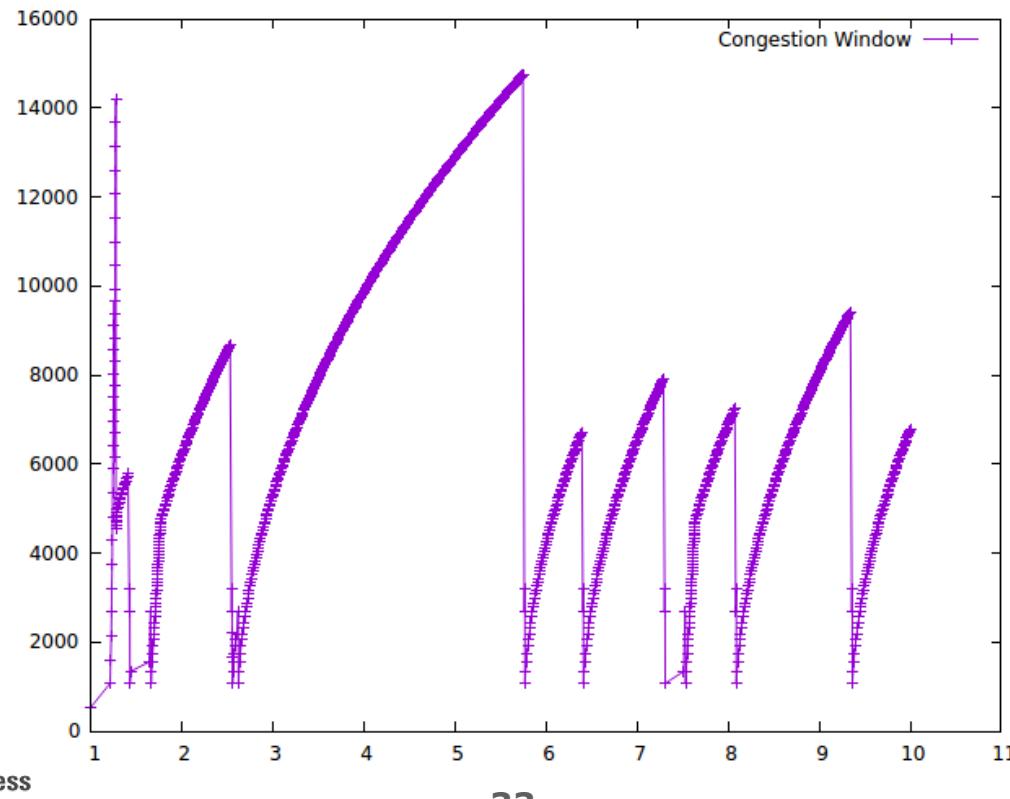
- Run the example code
 - ./waf --run “s4_ex2”

```
1.00419 536
1.21426 1072
1.22023 1608
1.2262 2144
...
1.25262 8040
1.25451 8576
RxDrop at 1.2562
...
```

- Redirect the console output to a file
 - ./waf --run “s4_ex2” >& cwnd.dat

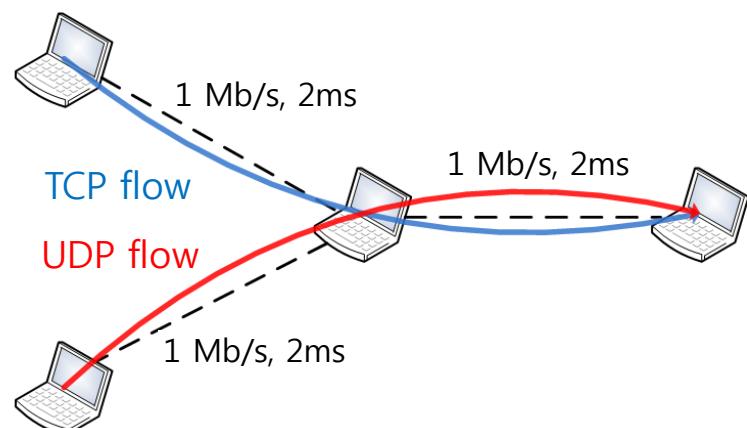
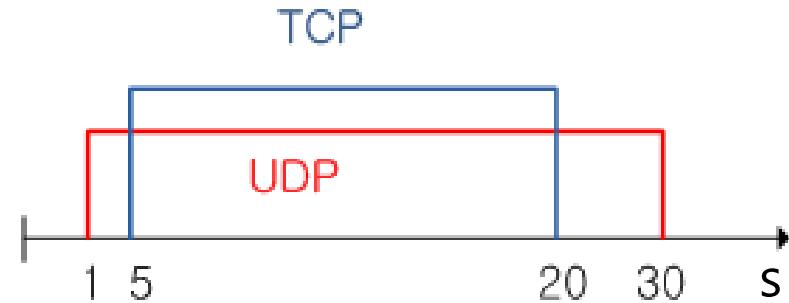
Example Result (2/2)

- Plot using *Gnuplot*
 - gnuplot
 - plot "cwnd.dat" using 1:2 ti 'Congestion Window' with linespoints
 - exit (or just "q")

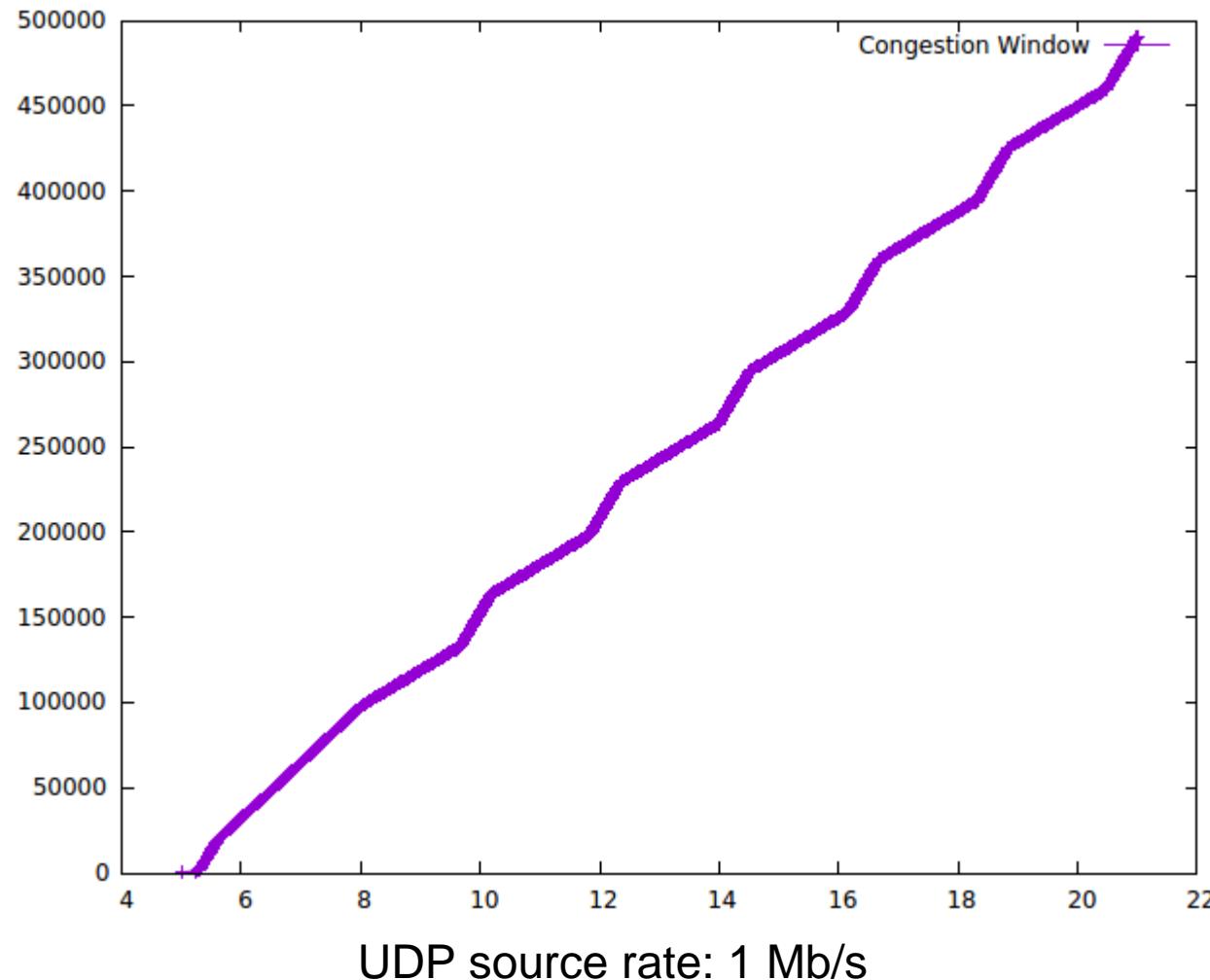


In-class Assignment

- 4 nodes connected by point-to-point links with DataRate= 1 Mb/s & Delay= 2 ms
- TCP traffic (0.5 Mb/s) is always on UDP traffic turns on and off every 1s
- Show the changes of CongestionWindow when UDP traffic source rate is 0.5 and 1, and 2 Mb/s
- ※ Statement initializing routing table
 - ✓ `Ipv4GlobalRoutingHelper::PopulateRoutingTables();`
- ※ The number of packets should be big enough (more than 0.1 million)



In-class Assignment Result



UDP source rate: 1 Mb/s