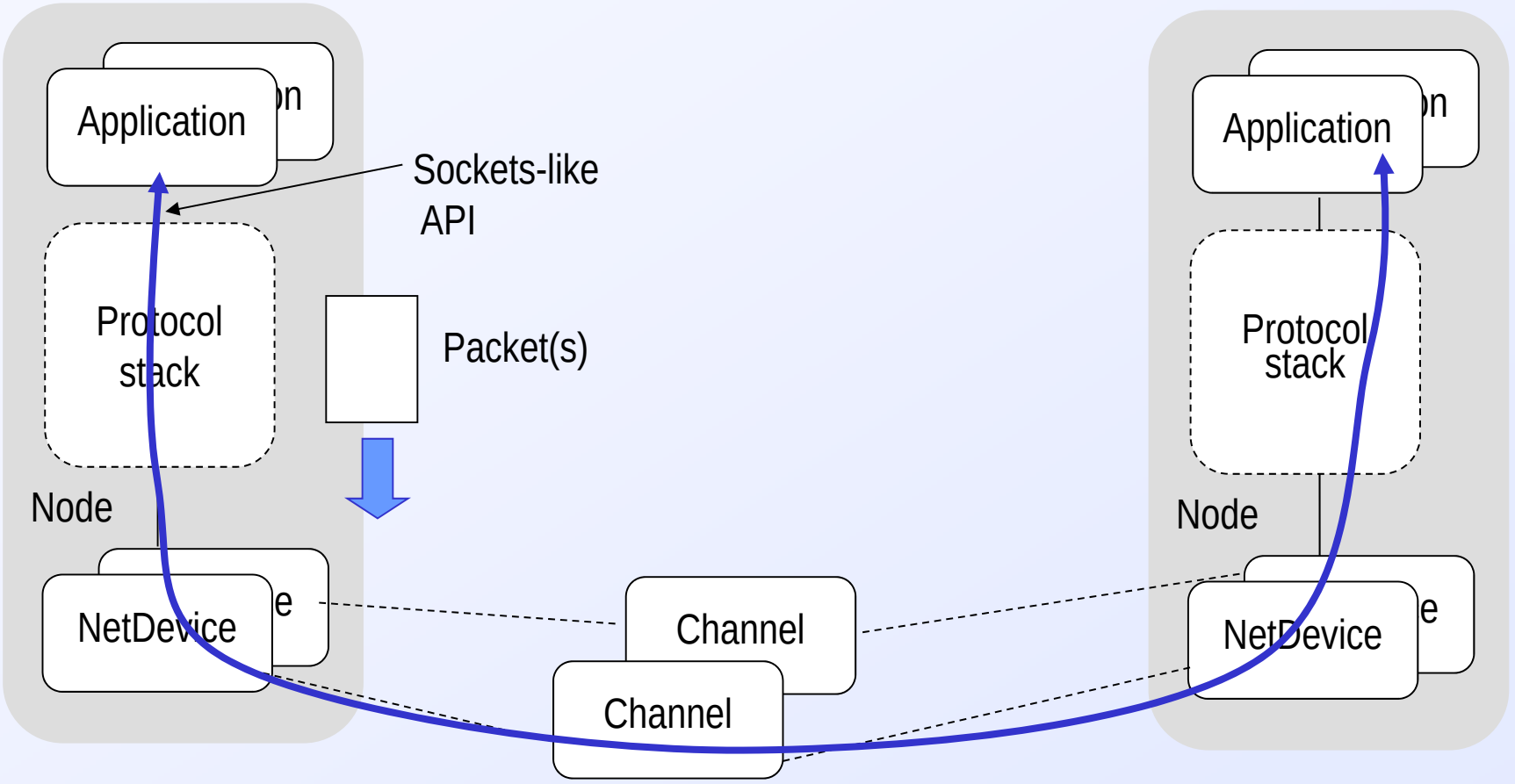
Chapter 4 Conceptual Overview

Source material:

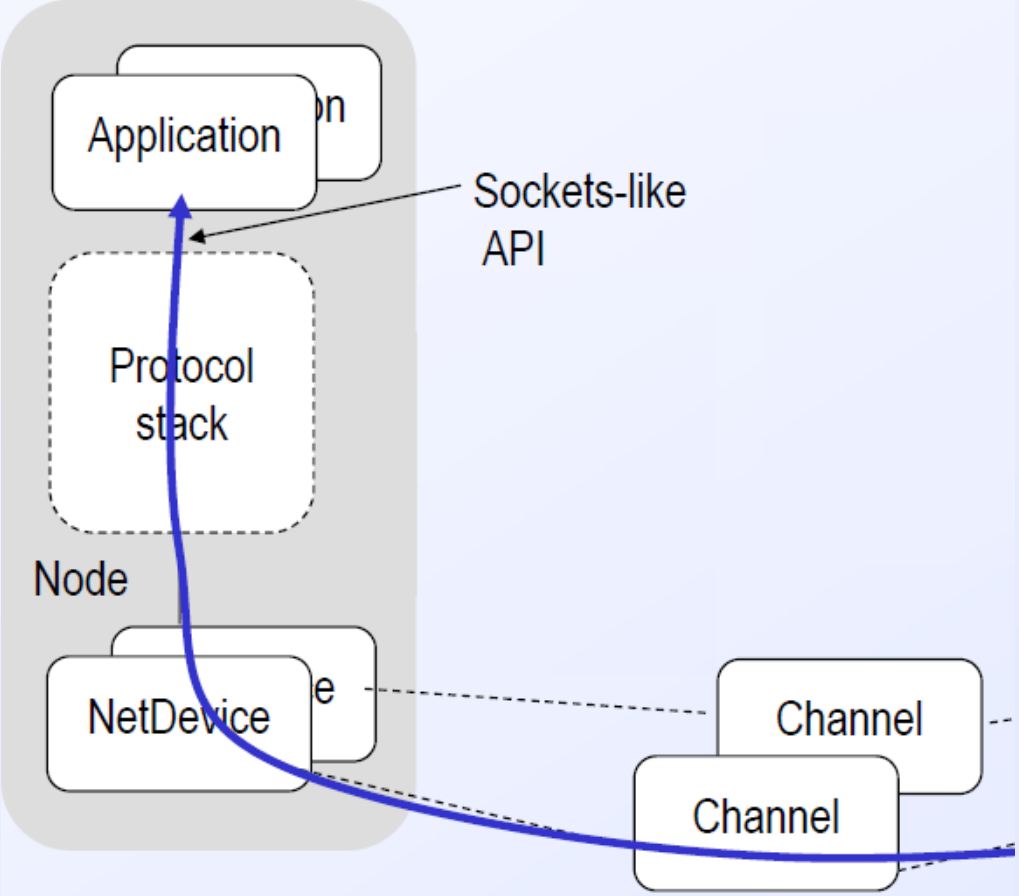
- https://www.nsnam.org/docs/release/3.28/tutorial/html/conceptual-overview.html
- https://www.nsnam.org/tutorials/ns-3-tutorial-Walid-Younes.pdf

The basic ns-3 architecture



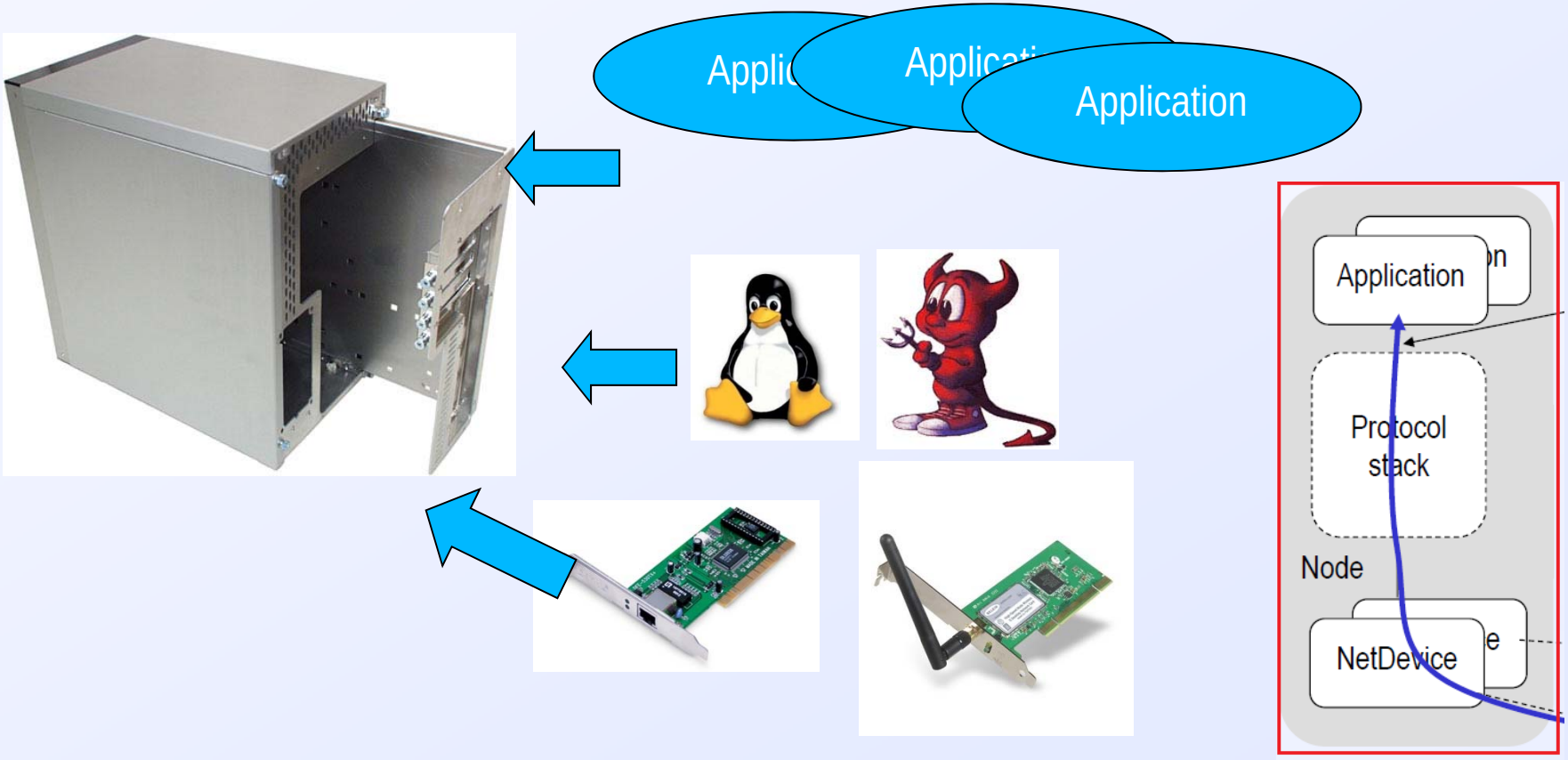
Components

- Basic Components
  - Nodes
  - Net Device
  - Channels
  - Application
  - Protocol Stack



Node basics

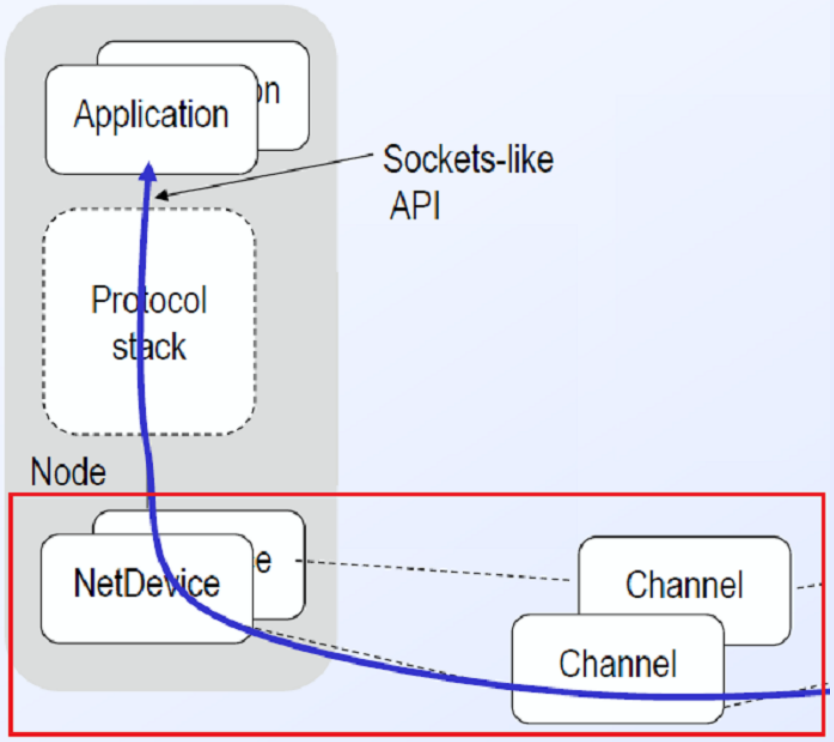
A Node is a shell of a computer to which applications, stacks, and NICs are added



NetDevices and Channels

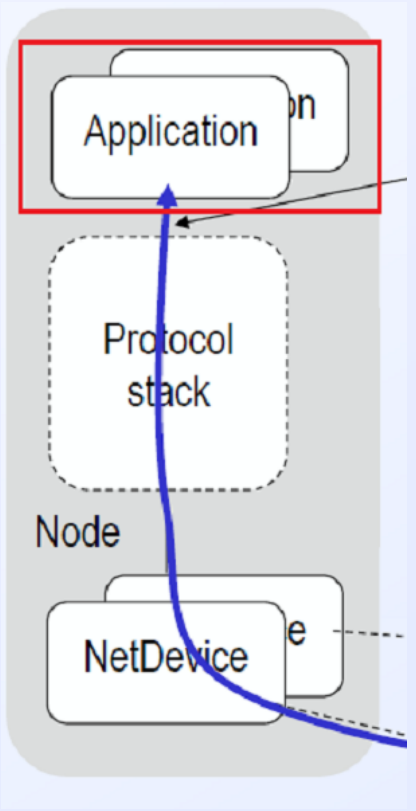
- NetDevices are strongly bound to Channels of a matching type
- Net Devices examples
  - Ethernet NIC
  - Wifi Net Device
- Channel examples
  - CSMA Channel
  - Wifi Channel
- **Point-to-point** – PPP link
- **CSMA** – Ethernet Link
- **Wifi** – 802.11 link
- and many more

Nodes are architected for multiple interfaces



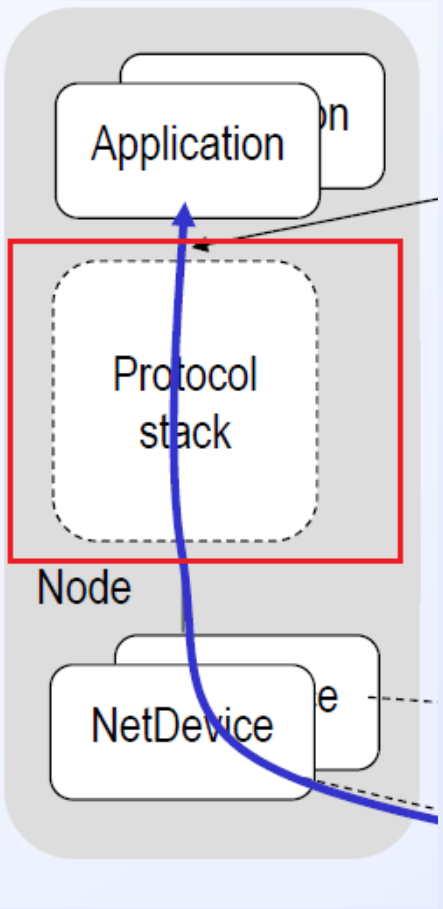
Application (Traffic Generator)

- Bulk-Send – Send data as fast as possible
  - `BulkSendApplication`
- On-Off – On off pattern
  - `OnOffApplication`
- Packet Sink - receives packets
  - `PacketSink`
- Udp-Server – Receive UDP packets
  - `UdpServer, UdpServerHelper`
- UDP-Client – UDP packet with seq no and time stamp
  - `UdpClient, UdpClientHelper`
- **UDP-echo-client, UDP-echo-server**
  - `UdpEchoClient,UdpEchoServer`



Internet Stack

- Internet Stack
  - Provides IPv4 and some IPv6 models currently
- No non-IP stacks ns-3 existed until 802.15.4 was introduced in ns-3.20
  - but no dependency on IP in the devices, Node object, Packet object, etc.(partly due to the object aggregation system)



Other basic models in ns-3

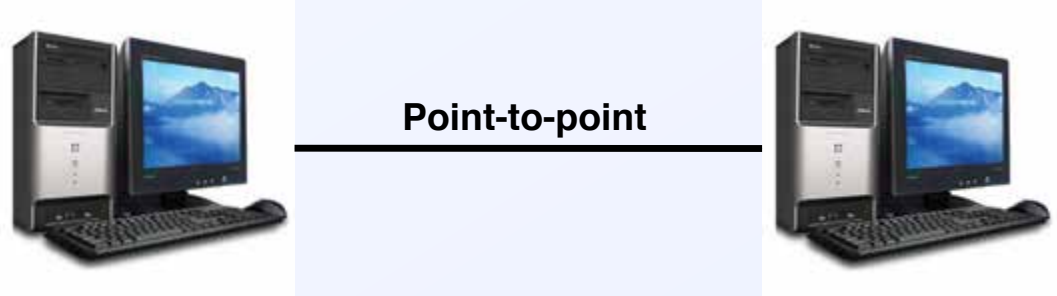
- Error models and queues
- Mobility models
- Packet routing
  - OLSR, AODV, DSR, DSDV, Static, Nix-Vector, Global (link state)

Structure of a generic ns-3 script

- Boilerplate: important for documentation
- Module includes: include header files
- ns-3 namespace: global declaration
- Logging: optional
- Main function: declare main function
- Topology helpers: objects to combine distinct operations
- Applications: on/off, UdpEchoClient/Server
- Tracing: `.tr` and/or `.pcap` files
- Simulator: start/end simulation, cleanup

Let's write a first network simulation

- A word about programming languages
  - I'll use C/C++, but also possible to use python
- Let's simulate a simple dumbbell topology:



- You will see a lot of container and helper classes
  - Containers are a convenient way of creating, managing, accessing groups of similar objects
  - Helpers make life easier by providing higher-level commands to execute common pieces of code

Containers and Helpers

- The general idea
  - Sets of objects are stored in Containers
  - Operations encoded in Helper object, apply to Container
- Provide high-level wrappers
- Easy way to build network models with repeating patterns of components, configurations
- With helpers, model description is
  - High level
  - More readable

Examples

- Containers
  - `NodeContainer`
  - `NetDeviceContainer`
  - `Ipv4AddressContainer`
- Helper classes
  - `InternetStackHelper`
  - `WifiHelper`
  - `MobilityHelper`
  - `OlsrHelper`
- Many models provides a helper class
  - See files in `src/*/helper` for source code

We will see examples in the various tutorial scripts

The C++ program that does the work: examples/tutorial/first.cc

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Omitted boilerplate legalese at the top of the code to save space, but read it!

No need to squint: we're going to look at this in detail!

Some standard overhead

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"
```

```
using namespace ns-3;
```

Logging

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
```

```
LogComponentEnable ("UdpEchoClientApplication",
  LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication",
  LOG_LEVEL_INFO);
```

Now traffic activity will be printed to the console

Focus first on the lines that create the network topology

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

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

Node 0

Node 1

These nodes don't do anything yet

Physical and Life Sciences

Next, define the point-to-point channel

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

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

point-to-point channel

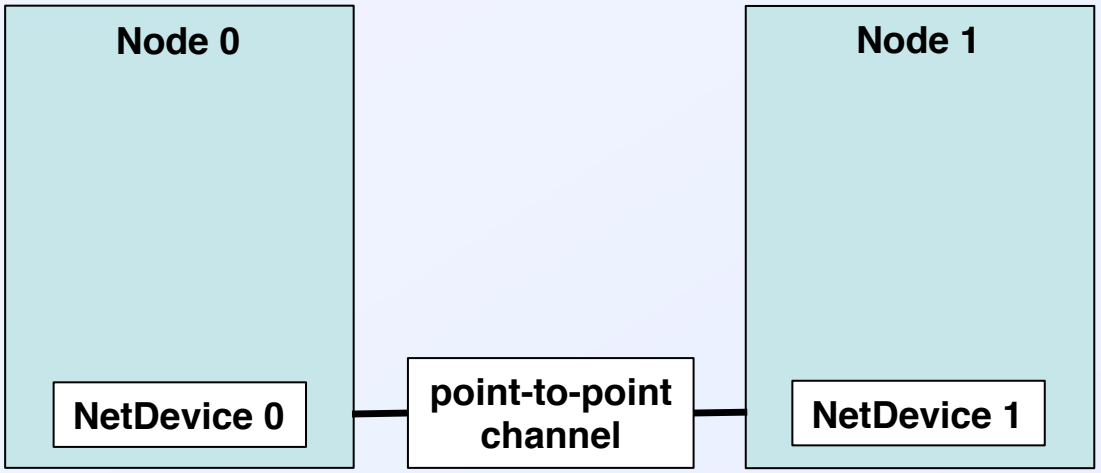
We've created the channel, but it's not connected to anything yet

Physical and Life Sciences

Next, define the point-to-point channel

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

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

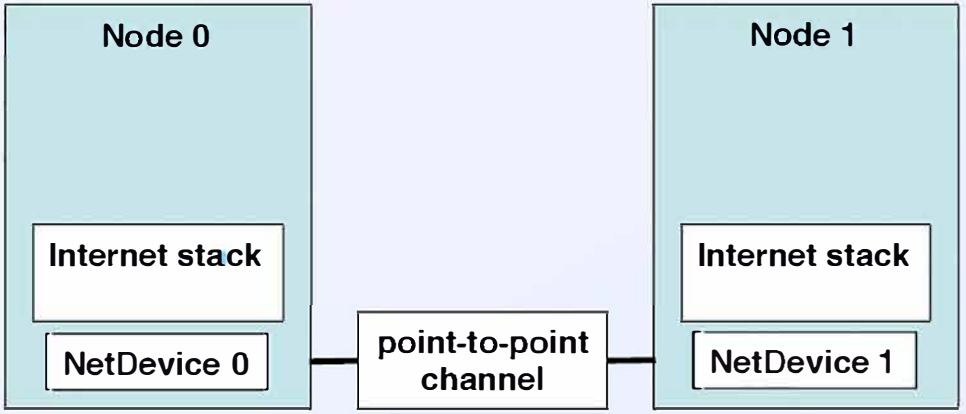


We've installed NICs on the nodes, and linked them via the p2p channel

Now we install Internet stack

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
InternetStackHelper stack;
stack.Install(nodes);
```



We have installed Internet stack on both nodes.

Let's define IP addresses

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

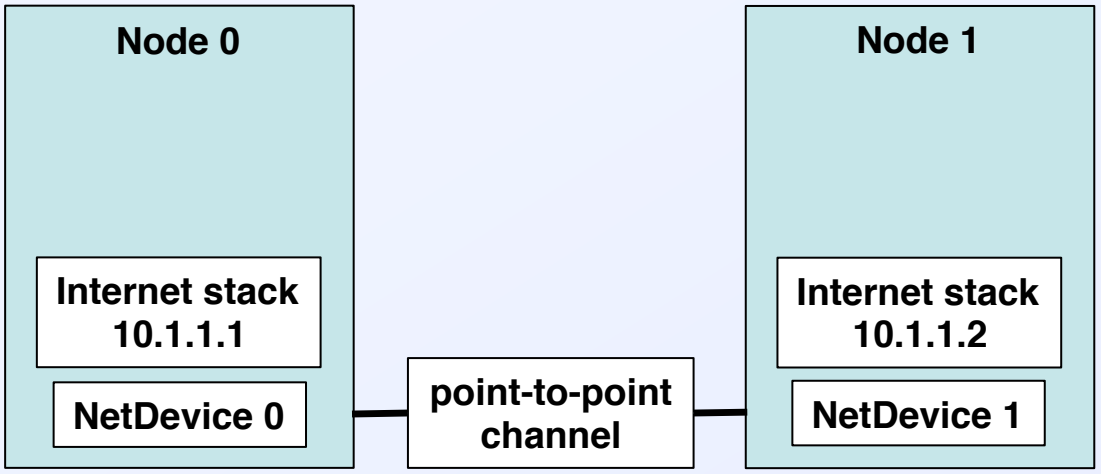
```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
```

- Mask = 255.255.255.0
- 1st address = 10.1.1.1
- 2nd address = 10.1.1.2
- …

Now we assign the IP addresses

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
Ipv4InterfaceContainer interfaces =
  address.Assign (devices);
```



All we need now are applications to generate some traffic

Set up the server application

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

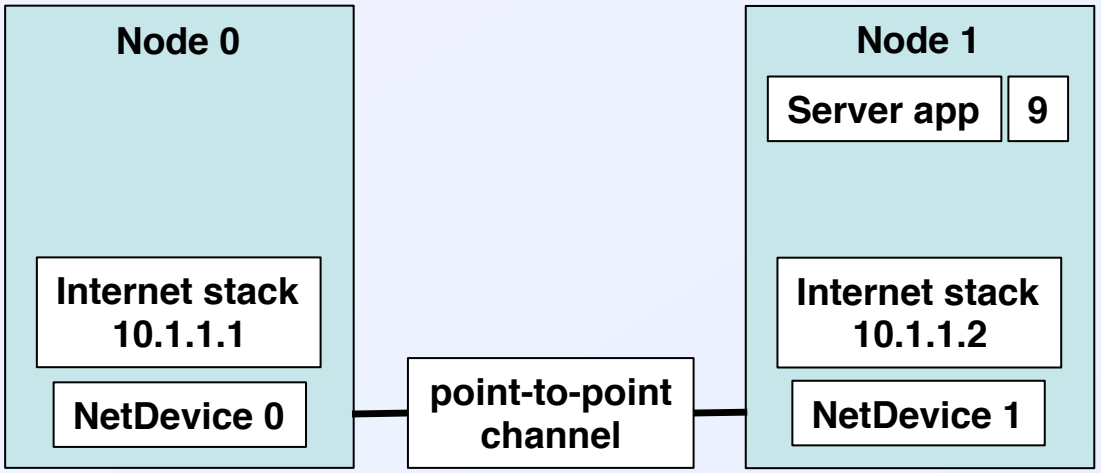
```
UdpEchoServerHelper echoServer (9);
```

- Sets up User Datagram Protocol echo server
- Uses port 9
- Echo server just echoes back any packet it receives
  - used, e.g., for testing or troubleshooting

Install the server application

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
ApplicationContainer serverApps =
  echoServer.Install (nodes.Get (1));
```

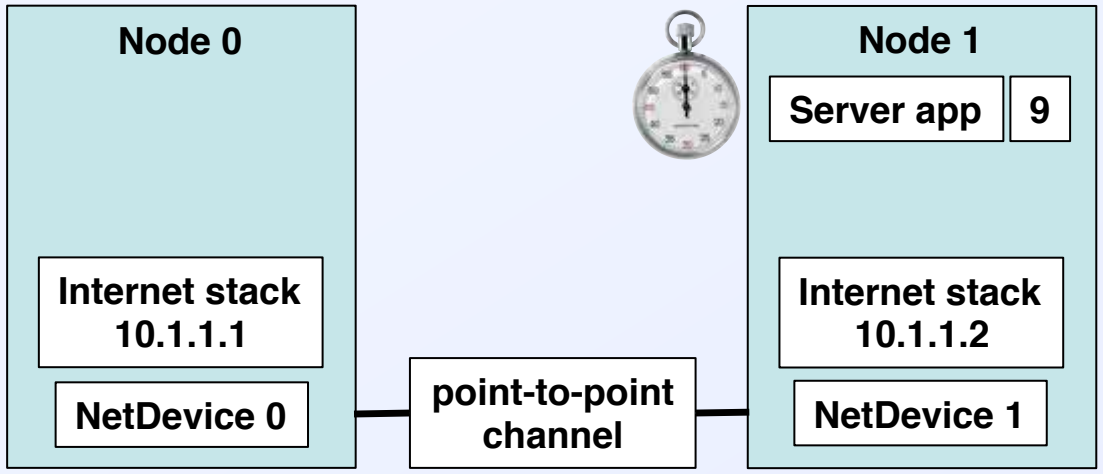


Schedule the server activity

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

- Echo server enabled at t = 1s
- Echo server disabled at t = 10s



Now the server knows when to listen: who's going to talk to it?

Define the client application

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
UdpEchoClientHelper echoClient
  (interfaces.GetAddress (1), 9);
```

- Remote address = server's IP = 10.1.1.2
- Remote port = 9

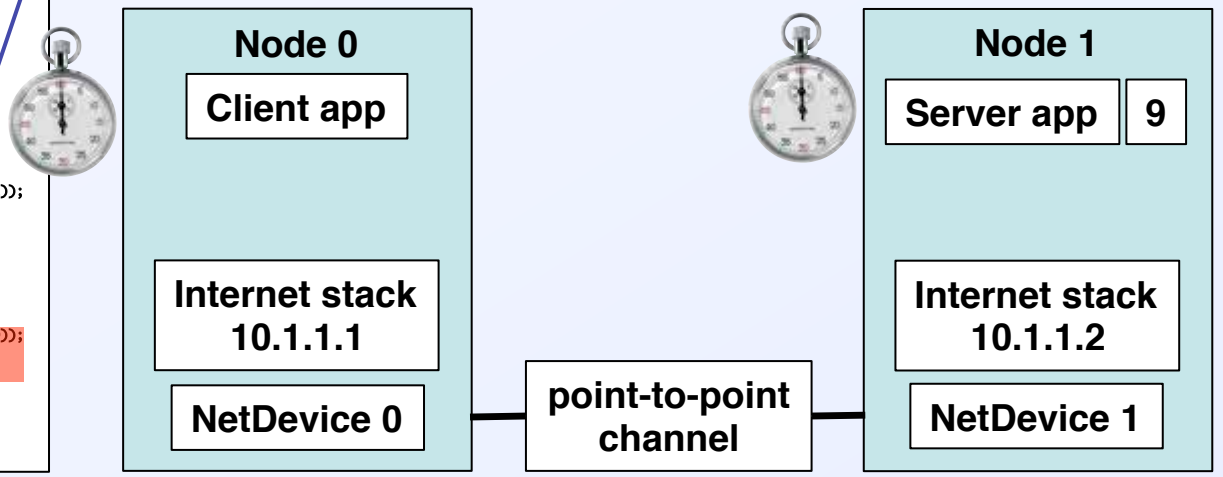
```
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

- MaxPackets = max number of packets to send
- Interval = time between sent packets
- PacketSize = size (1024 bytes in this case)

Install the client application

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
ApplicationContainer clientApps =
  echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```



Client: at t = 2s send a packet to the server

Finally: run the simulation & clean up

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
Simulator::Run
();
```

- **Run the simulation**

```
Simulator::Destroy ();
return 0;
```

- **Clean up**

What about output?

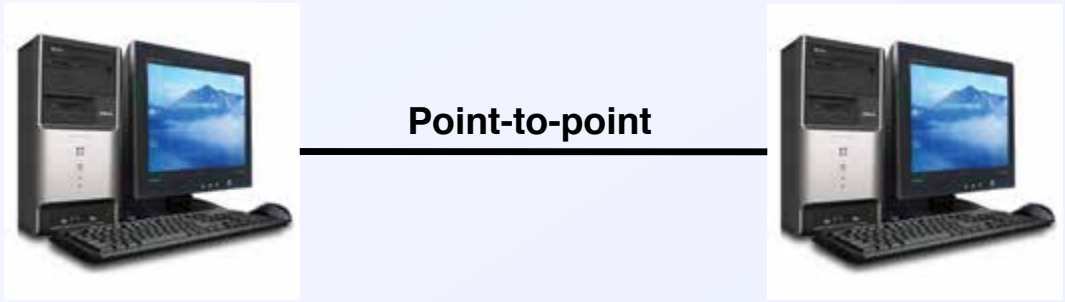
Building and running the script

- Copy `examples/tutorial/first.cc` into the `scratch` directory
  - `$ cp examples/tutorial/first.cc scratch/myfirst.cc`
- Build your script using WAF
  - `$ ./waf`
- Run it out of the scratch directory
  - `$ ./waf --run scratch/myfirst`
- Output:

```
Waf: Entering directory `/mypath/ns-3-dev/build'
Waf: Leaving directory `/mypath/ns-3-dev/build'
'build' finished successfully (5.283s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Closer look at the output from the script

```
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```



- Pretty terse output: can we get more?
  - Who did the sending & receiving?
  - Event times?
  - More…
- What if we want more than 2 nodes?

Now that you've had your first taste of ns-3: what else is there?

- Getting info out of your script
  - For debugging: logging
  - For actual work: tracing (we'll spend plenty of time on this)
- Building topologies
- Models, attributes and reality
- Where to go for more info
  - Official tutorial: https://www.nsnam.org/docs/release/3.28/tutorial/html/
  - Doxygen: https://www.nsnam.org/doxygen/index.html
  - Forum: https://groups.google.com/forum/?fromgroups#!forum/ns-3-users
  - Mathieu Lacage video (philosophy and design of ns-3): http://inl.info.ucl.ac.be/tutorials/tfiss09-lacage

Chapter 5: Tweaking

Source material:

- **Online tutorial:** https://www.nsnam.org/docs/release/3.28/tutorial/html/tweaking.html
- **M. Lacage (2009):** http://www.nsnam.org/tutorials/ns-3-tutorial-tunis-apr09.pdf

Logging

- Not the same thing as tracing
- Used primarily for
  - Quick messages
  - Debugging, not meant for extracting results!
- Macros of the form "NS_LOG*"
- Define logging component for your script
  - NS_LOG_COMPONENT_DEFINE(...)
- Enable logging for the script
  - LogCompoentEnable(...)
- Disable logging for the script
  - LogComponentDisable(…)

Different logging levels for different levels of verbosity

- Seven Logging levels
  - `LOG_ERROR(…)`: serious error messages only
  - `LOG_WARN(…)`: warning messages
  - `LOG_DEBUG(…)`: rare ad-hoc debug messages
  - `LOG_INFO(…)`: informational messages (e.g., banners)
  - `LOG_FUNCTION(…)`: function tracing
  - `LOG_LOGIC(…)`: control flow tracing within functions
  - `LOG_ALL(…)`: log everything
- LOG_LEVEL_TYPE =LOG_TYPE + all the levels above it
  - `LOG_LEVEL_INFO=LOG_INFO+LOG_DEBUG+LOG_WARN+LOG_ERROR`
- Displaying log messages: two ways
  - In your script: `LogComponentEnable(…)`
  - Using the `NS_LOG` environment variable

Using the NS_LOG environment variable

- With NS_LOG you can
  - Turn logging on/off for specific components
  - Set logging level
- In general you do something like this:
  - `export NS_LOG="name_of_logging_component=logging_flag"`
    - Or `setenv NS_LOG …` if you're using csh
  - The logging_flag can be
    - A logging level (e.g., `level_all`)
    - A combination of flags OR-ed together to print more info
- To set logging levels for several components use ":" between components

Let's look at some examples

Let's go back to `first.cc`

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
```

```
LogComponentEnable ("UdpEchoClientApplication",
  LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication",
  LOG_LEVEL_INFO);
```

```
$ ./waf –run scratch/myfirst
```

```
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

How do we get more info out of the client app without recompiling?

Getting debugging info from the client application

```
$ setenv NS_LOG “UdpEchoClientApplication=level_all”
$ ./waf –run scratch/myfirst
```

```
UdpEchoClientApplication:UdpEchoClient()
UdpEchoClientApplication:SetDataSize(1024)
UdpEchoClientApplication:StartApplication()
UdpEchoClientApplication:ScheduleTransmit()
UdpEchoClientApplication:Send()
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
UdpEchoClientApplication:HandleRead(0x683d40, 0x6845c0)
Received 1024 bytes from 10.1.1.2
UdpEchoClientApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
```

Now we can see all the function calls made by the application
Next: who sent and received what?

Adding a prefix to logged messages

Use "|" to combine flags

Use ":" to separate multiple components

```
$ setenv NS_LOG "UdpEchoClientApplication=level_all|prefix_func:
  UdpEchoServerApplication=level_all|prefix_func"
$ ./waf -run scratch/myfirst
```

Don't use a newline here!

```
UdpEchoServerApplication:UdpEchoServer()
UdpEchoClientApplication:UdpEchoClient()
UdpEchoClientApplication:SetDataSize(1024)
UdpEchoServerApplication:StartApplication()
UdpEchoClientApplication:StartApplication()
UdpEchoClientApplication:ScheduleTransmit()
UdpEchoClientApplication:Send()
UdpEchoClientApplication:Send(): Sent 1024 bytes to 10.1.1.2
UdpEchoServerApplication:HandleRead(): Received 1024 bytes from 10.1.1.1
UdpEchoServerApplication:HandleRead(): Echoing packet
UdpEchoClientApplication:HandleRead(0x683d50, 0x6845d0)
UdpEchoClientApplication:HandleRead(): Received 1024 bytes from 10.1.1.2
UdpEchoClientApplication:StopApplication()
UdpEchoServerApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoServerApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
UdpEchoServerApplication:~UdpEchoServer()
```

Note these lines

Who said what when? Extracting timing information

```
$ setenv NS_LOG "UdpEchoClientApplication=level_all|prefix_func|prefix_time:
   UdpEchoServerApplication=level_all|prefix_func|prefix_time"
$ ./waf -run scratch/myfirst
```

```
0s UdpEchoServerApplication:UdpEchoServer()
0s UdpEchoClientApplication:UdpEchoClient()
0s UdpEchoClientApplication:SetDataSize(1024)
1s UdpEchoServerApplication:StartApplication()
2s UdpEchoClientApplication:StartApplication()
2s UdpEchoClientApplication:ScheduleTransmit()
2s UdpEchoClientApplication:Send()
2s UdpEchoClientApplication:Send(): Sent 1024 bytes to 10.1.1.2
2.00369s UdpEchoServerApplication:HandleRead(): Received 1024 bytes from 10.1.1.1
2.00369s UdpEchoServerApplication:HandleRead(): Echoing packet
2.00737s UdpEchoClientApplication:HandleRead(0x6842a0, 0x684b20)
2.00737s UdpEchoClientApplication:HandleRead(): Received 1024 bytes from 10.1.1.2
10s UdpEchoClientApplication:StopApplication()
10s UdpEchoServerApplication:StopApplication()
UdpEchoClientApplication:DoDispose()
UdpEchoServerApplication:DoDispose()
UdpEchoClientApplication:~UdpEchoClient()
UdpEchoServerApplication:~UdpEchoServer()
```

Note the extra 3.69 ms: due to channel delay & device data rate

Adding your own log messages to the script

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

Add the line:
`NS_LOG_INFO("Creating Topology");`

```
$ setenv NS_LOG "FirstScriptExample=info"
$ ./waf -run scratch/myfirst
```

```
Creating Topology
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Note this line

Physical and Life Sciences

Using Command Line Arguments

- **Adding command line argument “nPackets”**

```
…
CommandLine cmd;
cmd.AddValue("nPackets", "Number of packets to echo", nPackets);
cmd.Parse (argc, argv);
...
```

- **Use the variable “nPackets”**

```
echoClient.SetAttribute ("MaxPackets", UintegerValue (nPackets));
```

- **Run the script with user argument**

```
$ ./waf --run "scratch/myfirst --nPackets=2"
```

- **Find out more on arguments**

```
$ ./waf --run "scratch/myfirst --PrintHelp"
```

A first (quick) look at tracing

- Tracing is a structured form of simulation output
- Built on the principle of trace sources and sinks
  - Trace source: signals sim event and provides access to the data
  - Trace sink: consumes the trace information
- For now: high-level tracing
  - Use pre-defined trace sinks in helpers
- Two types of tracing are available
  - ASCII tracing: produces readable files
  - PCAP tracing: produces files for traffic analyzers (tcpdump, wireshark)

Let's look at some examples

ASCII tracing in `myfirst.cc`

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

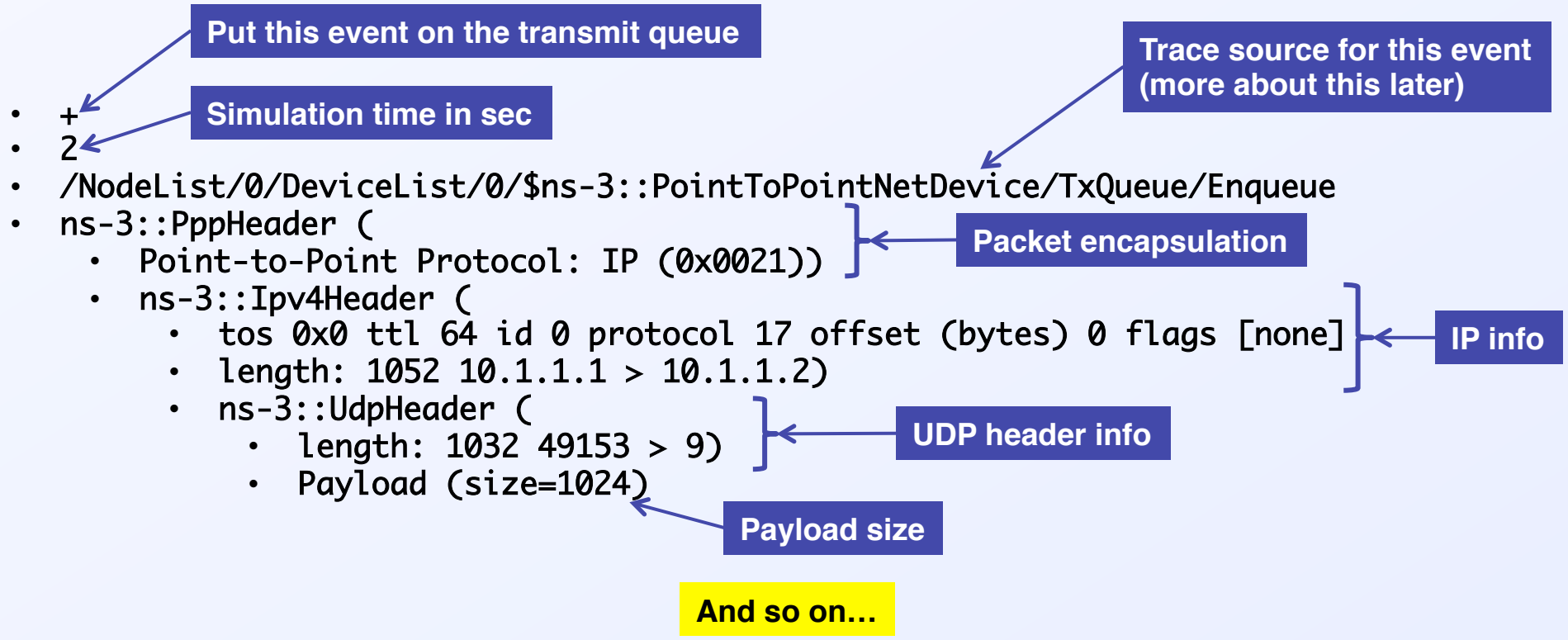
```
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll( ascii.CreateFileStre
  am("myfirst.tr"));
```

```
$ ./waf –run scratch/myfirst
```

- **Creates `myfirst.tr` in top level directory**
- **Each line = one trace event**

The `myfirst.tr` file: breakdown of the 1st line

```
+2 /NodeList/0/DeviceList/0/$ns-3::PointToPointNetDevice/TxQueue/Enqueue
ns-3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns-3::Ipv4Header
(tos 0x0 ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length:
1052 10.1.1.1 > 10.1.1.2) ns-3::UdpHeader (length: 1032 49153 > 9) Payload
(size=1024)
```



PCAP tracing in `myfirst.cc`

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  NodeContainer nodes;
  nodes.Create (2);

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

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

  InternetStackHelper stack;
  stack.Install (nodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");

  Ipv4InterfaceContainer interfaces = address.Assign (devices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

```
pointToPoint.EnablePcapAll("myfirst");
```

```
$ ./waf -run scratch/myfirst
```

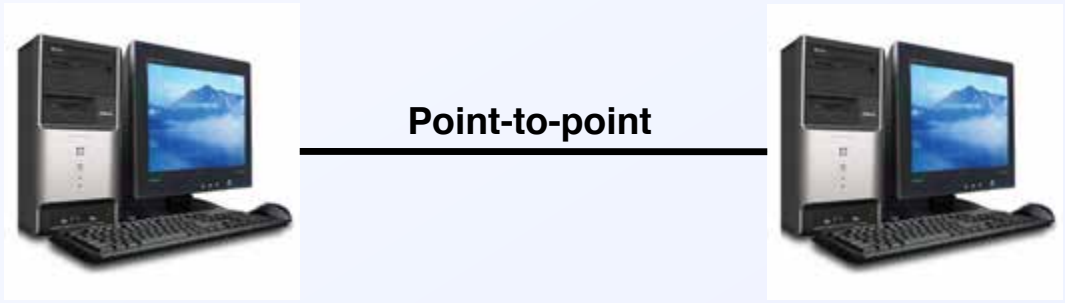
Creates pcap files in top level directory

- `myfirst-0-0.pcap`
- `myfirst-1-0.pcap`

node **device**

We'll see later how to give more descriptive names…

Contents of the PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r myfirst-0-0.pcap
reading from file myfirst-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.007372 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

```
$ /usr/sbin/tcpdump -nn -tt -r myfirst-1-0.pcap
reading from file myfirst-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.1.2.9: UDP, length 1024
2.003686 IP 10.1.1.2.9 > 10.1.1.1.49153: UDP, length 1024
```

Summary

- Using the Logging Module
- Using Command Line Arguments
- Using the Tracing System

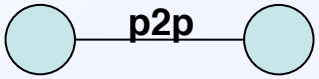
Chapter 6: Building topologies

Source material:

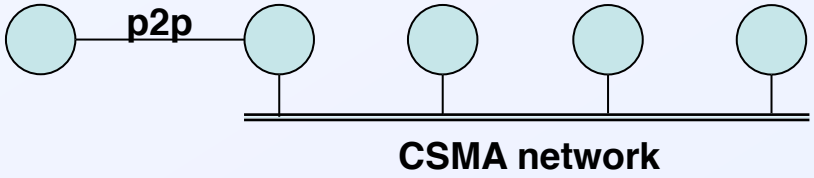
- **Online tutorial:** https://www.nsnam.org/docs/release/3.28/tutorial/html/building-topologies.html
- **G. Riley (2008):** http://www.wns2.org/docs/wns_tutorial-handout.pdf

Building topology: the goal

Expand from this:



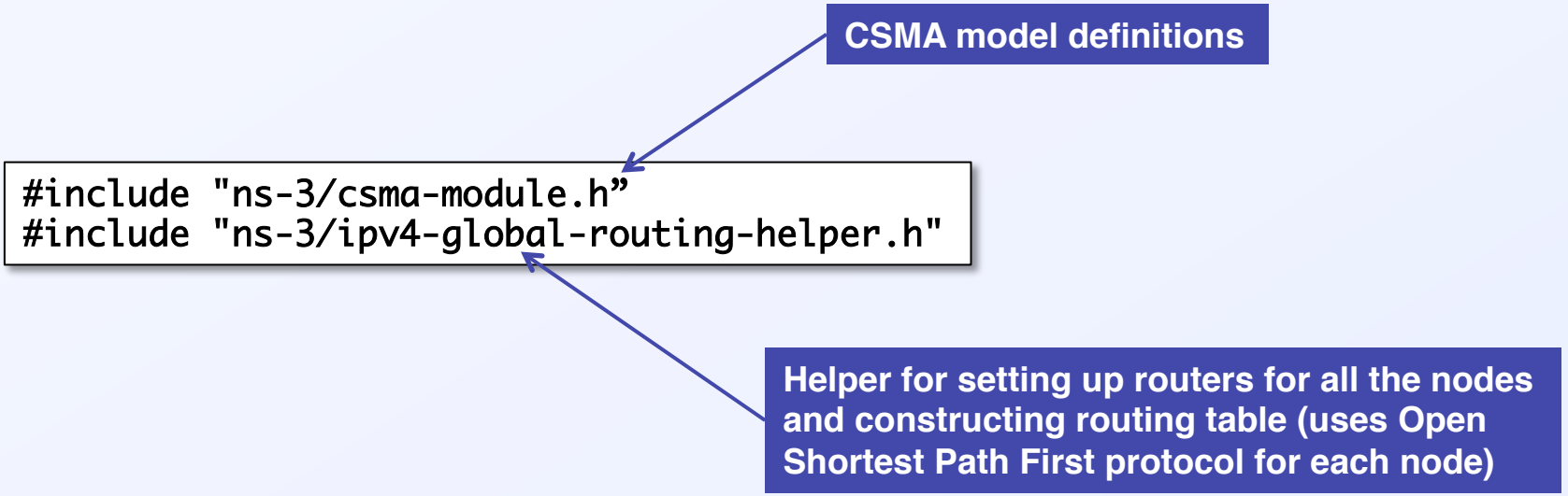
To this:



Anatomy of the second.cc script (orange⇒ not in first.cc)

```
#include "ns-3/core-module.h"
#include "ns-3/network-module.h"
#include "ns-3/csma-module.h"
#include "ns-3/internet-module.h"
#include "ns-3/point-to-point-module.h"
#include "ns-3/applications-module.h"
#include "ns-3/ipv4-global-routing-helper.h"

using namespace ns-3;

NS_LOG_COMPONENT_DEFINE ("SecondScriptExample");

int main (int argc, char *argv[])
{
  bool verbose = true;
  uint32_t nCsma = 3;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

  cmd.Parse (argc,argv);

  if (verbose)
    {
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
    }

  nCsma = nCsma == 0 ? 1 : nCsma;

  NodeContainer p2pNodes;
  p2pNodes.Create (2);

  NodeContainer csmaNodes;
  csmaNodes.Add (p2pNodes.Get (1));
  csmaNodes.Create (nCsma);

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

  NetDeviceContainer p2pDevices;
  p2pDevices = pointToPoint.Install (p2pNodes);

  CsmaHelper csma;
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

  NetDeviceContainer csmaDevices;
  csmaDevices = csma.Install (csmaNodes);

  InternetStackHelper stack;
  stack.Install (p2pNodes.Get (0));
  stack.Install (csmaNodes);

  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer p2pInterfaces;
  p2pInterfaces = address.Assign (p2pDevices);

  address.SetBase ("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer csmaInterfaces;
  csmaInterfaces = address.Assign (csmaDevices);

  UdpEchoServerHelper echoServer (9);

  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get
(nCsma));
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));

  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  pointToPoint.EnablePcapAll ("second");
  csma.EnablePcap ("second", csmaDevices.Get (1), true);

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

What's new? A couple of additional include files

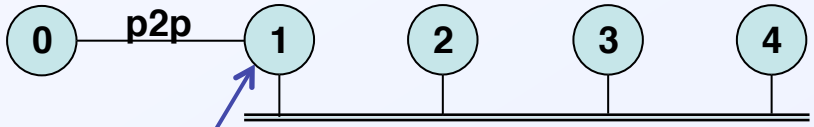


What's new? Command line parser

```
bool verbose = true;
uint32_t nCsma = 3;

CommandLine cmd;
cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);

cmd.Parse (argc,argv);

nCsma = nCsma == 0 ? 1 : nCsma;
```

- Default value nCsma = 3
- CommandLine class can be used to parse command-line arguments
- User can register new arguments with AddValue(name,help,value)
  - name = name of user-supplied argument
  - help = some help text
  - value = reference to the variable where the value parsed will be stored
- The Parse method does exactly what it sounds like it does

What's new? Two devices on a single node



Point-to-point and CSMA devices

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

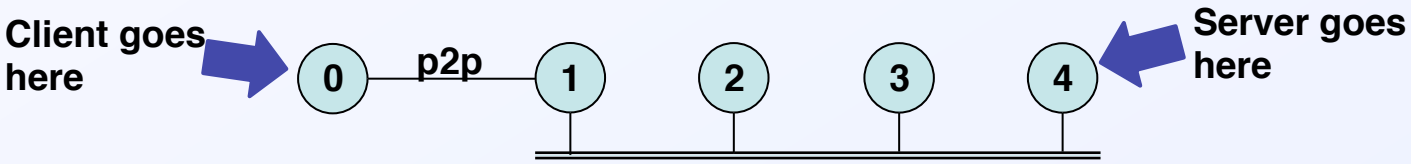
Node 1 will get both types of devices

Creates nCsma=3 additional nodes

```
stack.Install (p2pNodes.Get (0));
stack.Install (csmaNodes);
```

No need to install a stack twice on node 1

What's new? Where do we put the client and server?




```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Adds our dual-purpose node

Creates nCsma=3 additional nodes

- `nCsma` is the index of the last node in `csmaNodes`
- ⇒ The following lines refer to the server all the way to the right

```
ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
```

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
```

- The following line installs the client all the way to the left

```
ApplicationContainer clientApps = echoClient.Install (p2pNodes.Get (0));
```

What's new? Not much else…

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Constructs the routing tables

- The rest looks almost identical to the p2p setup

Set channel attributes and install CSMA devices

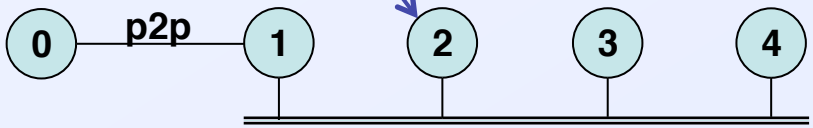
```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

```
address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);
```

Assign IP addresses to CSMA devices

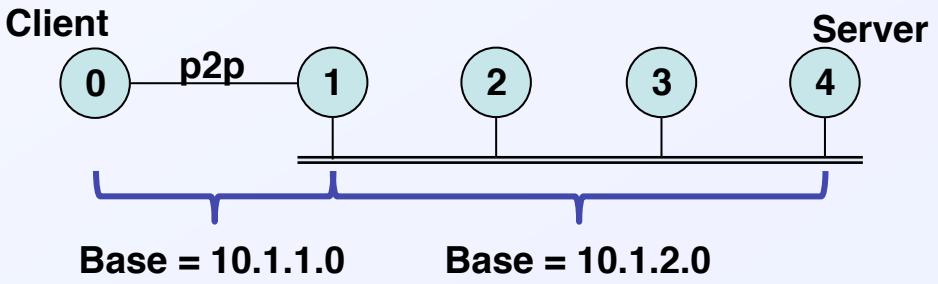
```
csma.EnablePcap ("second", csmaDevices.Get (1), true);
```

File prefix | Device id | Promiscuous mode

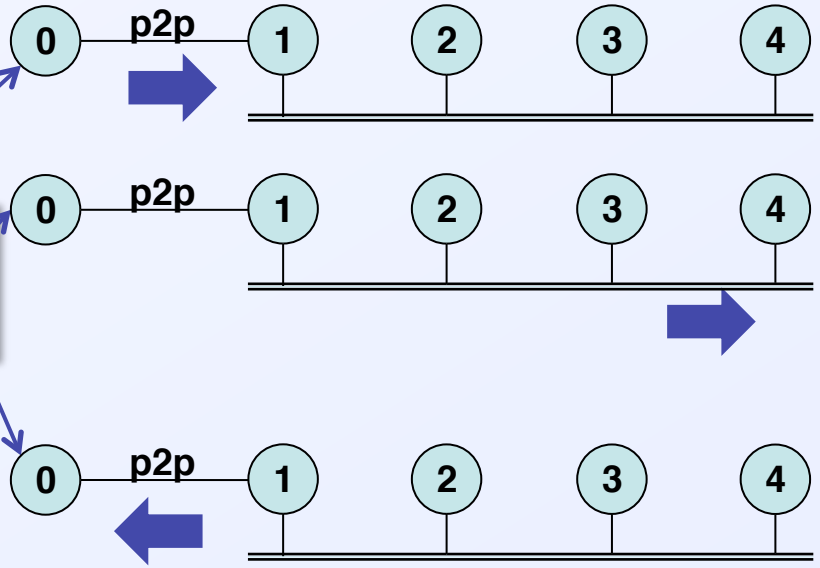


Running mysecond: logging output to console

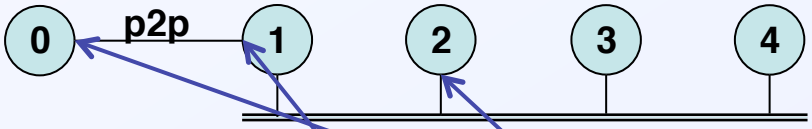
```
$ setenv NS_LOG=“”
$ ./waf –run scratch/mysecond
```




```
Sent 1024 bytes to 10.1.2.4
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.2.4
```



Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

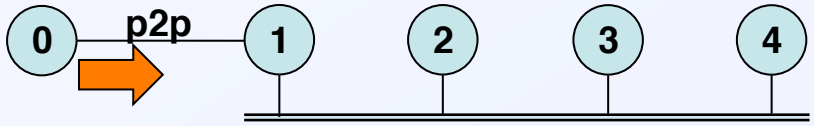
```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

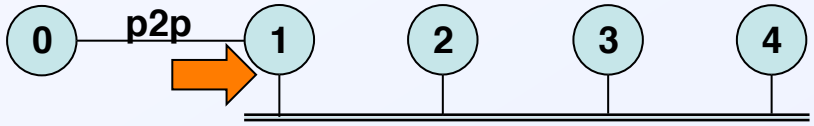
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Packet leaves client, headed for server

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

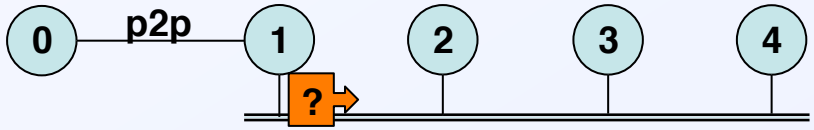
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Packet reaches node 1 on p2p network

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

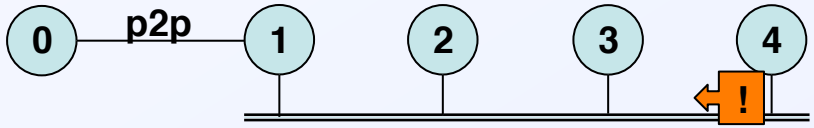
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node 1 broadcasts request on CSMA network

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

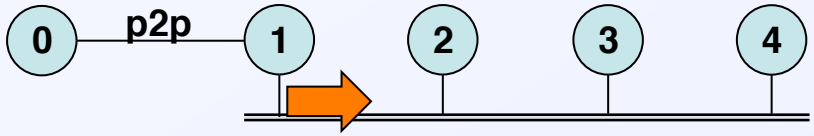
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node 4 replies with MAC address

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

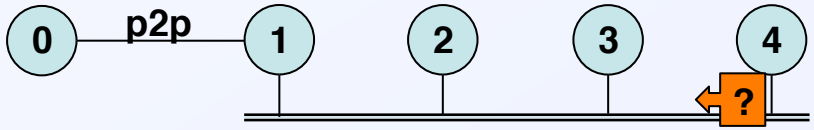
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Packet continues on its way to server

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

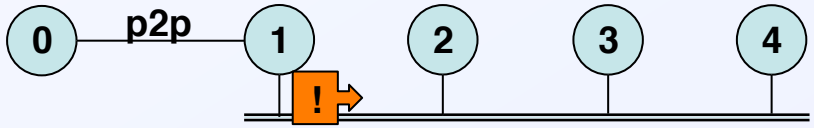
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Server broadcasts request for address of 1st CSMA node

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

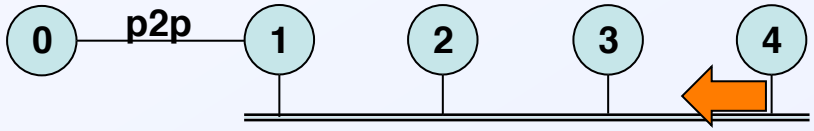
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

1st CSMA node replies with MAC address

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

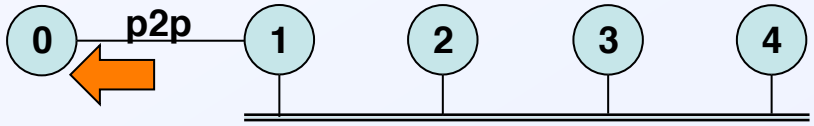
Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Server echoes the packet back

Running mysecond.cc: PCAP files




```
$ /usr/sbin/tcpdump -nn -tt -r second-0-0.pcap
reading from file second-0-0.pcap, link-type PPP (PPP)
2.000000 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.017607 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 0
Device = 0 (p2p)

```
$ /usr/sbin/tcpdump -nn -tt -r second-1-0.pcap
reading from file second-1-0.pcap, link-type PPP (PPP)
2.003686 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Node = 1
Device = 0 (p2p)

Node = 2
Device = 0 (CSMA)

```
$ /usr/sbin/tcpdump -nn -tt -r second-2-0.pcap
reading from file second-2-0.pcap, link-type EN10MB (Ethernet)
2.007698 ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
2.007710 ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
2.007803 IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
2.013815 ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
2.013828 ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
2.013921 IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

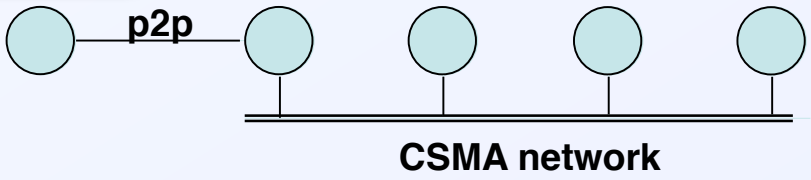
Echoed packet reaches the client

Summary: building PointToPoint and CSMA networks

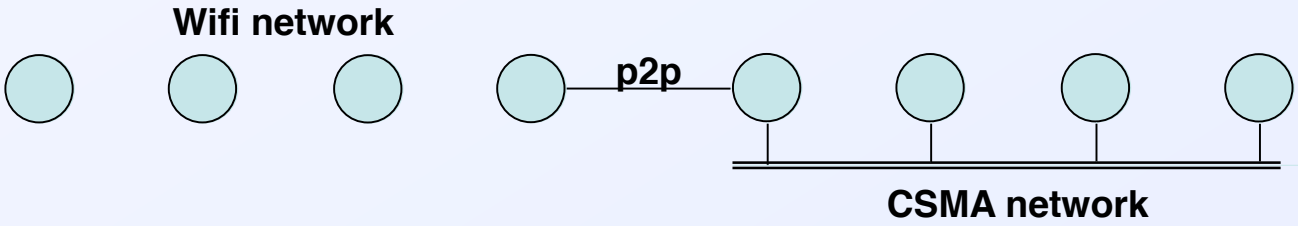
- Create nodes
  - NodeContainer
- Set channel attributes
  - PointToPointHelper
  - pointToPoint.SetDeviceAttribute
  - pointToPoint.SetChannelAttribute
- Install channel on nodes
  - NetDeviceContainer devices;
  - devices = pointToPoint.Install (nodes);
- Assign IP address to device
  - Ipv4InterfaceContainer interfaces = address.Assign (devices);

Building wireless topology: the goal

Expand from this:



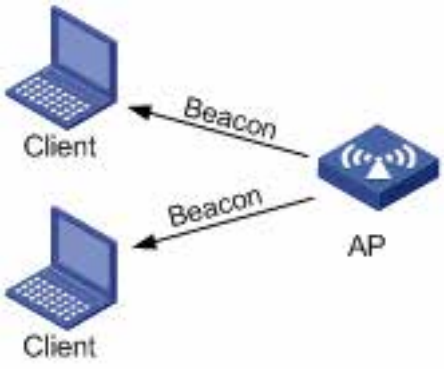
To this:

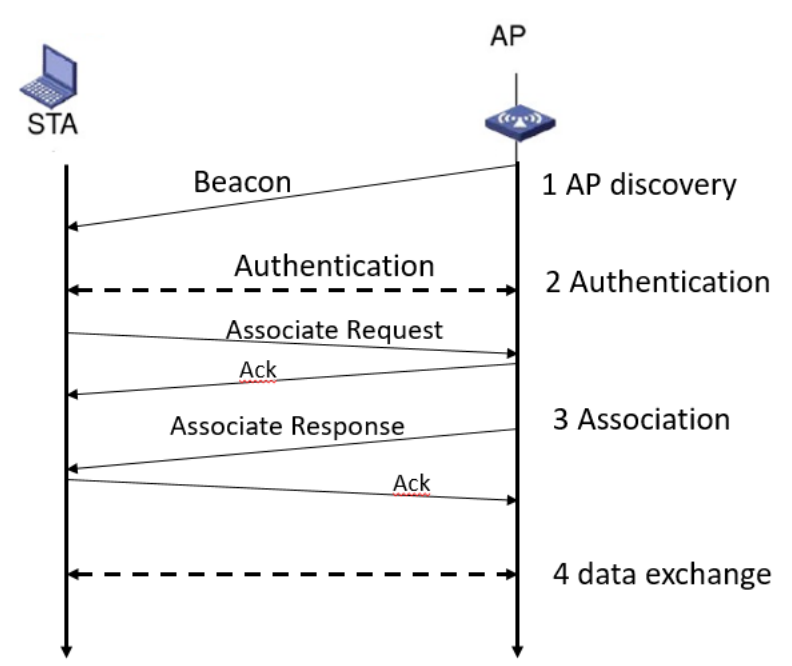


Physical and Life Sciences

Wireless networks 101

- Access point (AP) sends out beacons to advertise WLAN presence by revealing its Service Set ID (SSID = network name)
- WLAN client can
  - Broadcast probe request, wait for AP response (= active probing)
  - Wait for beacon from AP (= passive probing)
- Then (authentication followed by) association





Creating a wireless network with ns-3: `third.cc`

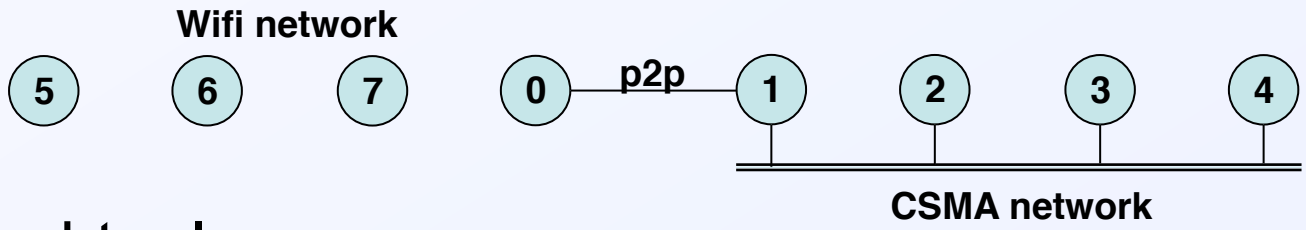
creating wireless network

- Create 2 types of nodes
  - Stations
  - Access Points (AP)
- Create physical layer and channel, and associate
- Create MAC layer characteristics for node type
  - QoS or non-QoS
  - Assign Service Set Identifier (SSID)
  - And other MAC related attributes
- Install phy, and mac to nodes-> devices
  - apDevices = wifi.Install (phy, mac, wifiApNode);
- Set up mobility models
- Configure internet stack, application, routing models

creating P2P network

- Create nodes
  - NodeContainer
- Set channel attributes
  - PointToPointHelper
  - pointToPoint.SetDeviceAttribute
  - pointToPoint.SetChannelAttribute
- Install channel on nodes
  - NetDeviceContainer devices;
  - devices = pointToPoint.Install (nodes);
- Assign IP address to device
  - Ipv4InterfaceContainer interfaces = address.Assign (devices);

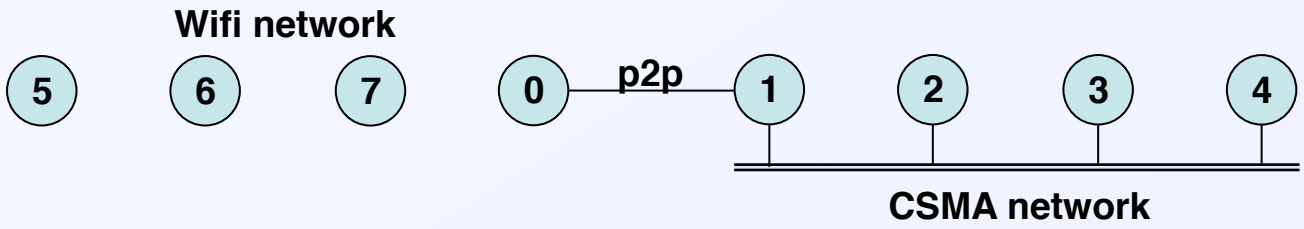
Creating a wireless network with ns-3: `third.cc`



- Network topology
  - Wireless nodes/links
    - 3 STA nodes
    - 1 AP node
    - 802.11 links, non-QoS mode, beaconing enabled
  - Wired nodes
    - 2 connected via p2p link
    - 4 nodes on a CSMA LAN
- Applications: UdpEchoClient/Server
  - Client on a STA node
  - Server on CSMA subnetwork

Let's focus on the wireless-specific parts of the code

Use helpers to create wireless channel/physical/MAC models



Create `nWifi = 3` station nodes

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns-3::AarfWifiManager");

WifiMacHelper mac;
```

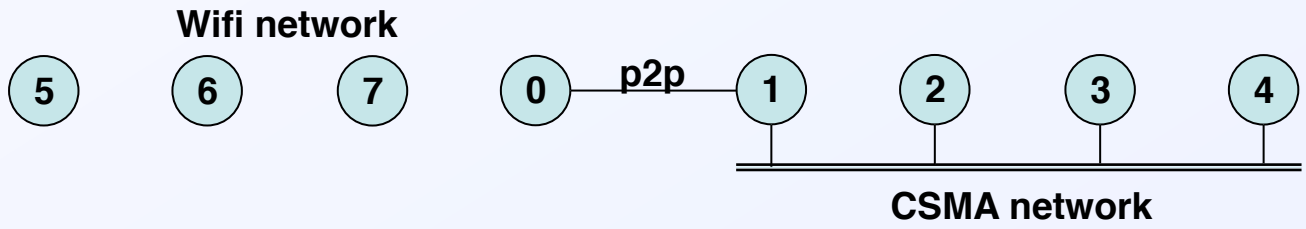
Create 1 AP node

Configure the physical and channel helpers

Connect all physical layer objects via the wireless channel

Configure the MAC layer

Configure MAC types and install Wifi devices



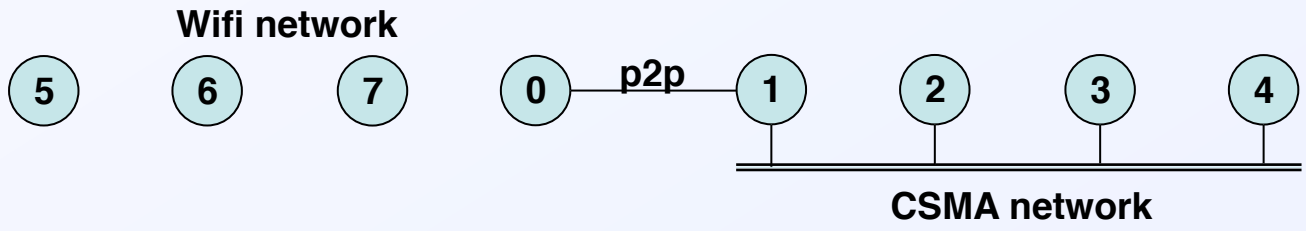

```
WifiMacHelper mac;
```

```
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns-3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;

staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns-3::ApWifiMac",
             "Ssid", SsidValue (ssid))


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

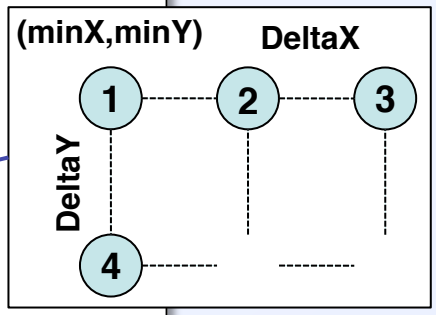
Station nodes:
- 802.11 Service Set Identifier
- No QoS support
- Active probing disabled

Access point node:
- 802.11 Service Set Identifier
- No QoS support

Associate mobility models with all wireless nodes



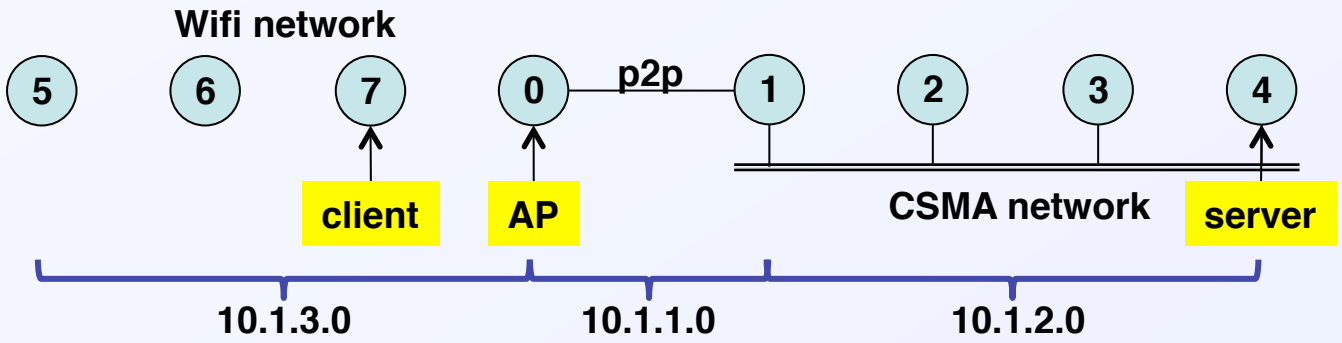

```
MobilityHelper mobility;

 mobility.SetPositionAllocator ("ns-3::GridPositionAllocator",
                                "MinX", DoubleValue (0.0),
                                "MinY", DoubleValue (0.0),
                                "DeltaX", DoubleValue (5.0),
                                "DeltaY", DoubleValue (10.0),
                                "GridWidth", UintegerValue (3),
                                "LayoutType", StringValue ("RowFirst"));

 mobility.SetMobilityModel ("ns-3::RandomWalk2dMobilityModel",
                            "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
 mobility.Install (wifiStaNodes);

 mobility.SetMobilityModel ("ns-3::ConstantPositionMobilityModel");
 mobility.Install (wifiApNode);
```



Brownian-motion model with elastic scattering off boundaries

Fixed access point

Some final points before we run the simulation




```
Simulator::Stop (Seconds (10.0));
```

Needed to prevent AP from scheduling an endless sequence of beacon events

Trace on:
node 0, p2p device
node 1, p2p device

```
pointToPoint.EnablePcapAll ("third");
phy.EnablePcap ("third", apDevices.Get (0));
csma.EnablePcap ("third", csmaDevices.Get (0), true);
```

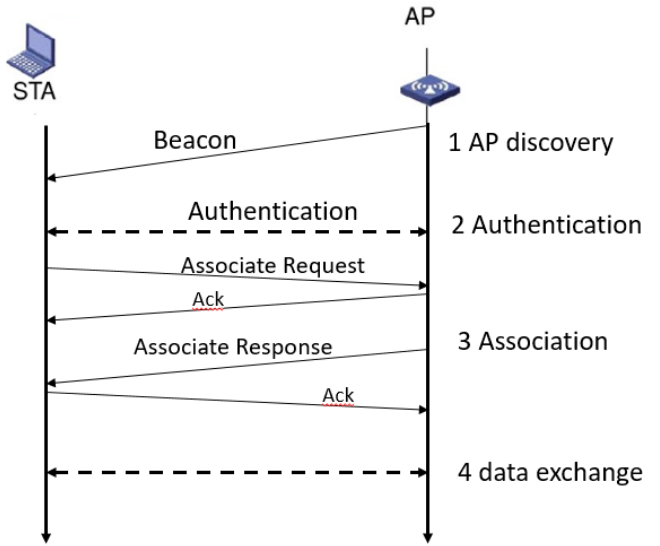
Trace on node 0, Wifi device

Promiscuous trace on node 1, CSMA device

Running `mythird.cc`: PCAP trace of the Wifi network

```
junhuatang@ubuntu:~/workspace/ns-allinone-3.28/ns-3.28$ tcpdump -nn -tt -r myth
ird-0-1.pcap
reading from file mythird-0-1.pcap, link-type IEEE802_11 (802.11)
0.032090 Beacon (ns-3-ssid) [6.0* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS
0.032346 Assoc Request (ns-3-ssid) [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
0.032362 Acknowledgment RA:00:00:00:00:00:09
0.032554 Assoc Request (ns-3-ssid) [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
0.032570 Acknowledgment RA:00:00:00:00:00:08
0.032684 Assoc Response AID(1) :: Successful
0.032828 Acknowledgment RA:00:00:00:00:00:0a
0.032985 Assoc Request (ns-3-ssid) [6.0 9.0 12.0 18.0 24.0 36.0 48.0 54.0 Mbit]
0.033001 Acknowledgment RA:00:00:00:00:00:07
0.033133 Assoc Response AID(2) :: Successful
0.033277 Acknowledgment RA:00:00:00:00:00:0a
0.033392 Assoc Response AID(3) :: Successful
0.033536 Acknowledgment RA:00:00:00:00:00:0a
0.134490 Beacon (ns-3-ssid) [6.0* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS
0.236890 Beacon (ns-3-ssid) [6.0* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS
.....
2.006112 ARP, Request who-has 10.1.3.4 (ff:ff:ff:ff:ff:ff) tell 10.1.3.3, lengt
h 32
2.006128 Acknowledgment RA:00:00:00:00:00:09
2.006233 ARP, Request who-has 10.1.3.4 (ff:ff:ff:ff:ff:ff) tell 10.1.3.3, lengt
h 32
2.006433 ARP, Reply 10.1.3.4 is-at 00:00:00:00:00:0a, length 32
2.006605 Acknowledgment RA:00:00:00:00:00:0a
2.008133 IP 10.1.3.3.49153 > 10.1.2.4.9: UDP, length 1024
2.008149 Acknowledgment RA:00:00:00:00:00:09
2.031740 ARP, Request who-has 10.1.3.3 (ff:ff:ff:ff:ff:ff) tell 10.1.3.4, lengt
h 32
2.032089 ARP, Reply 10.1.3.3 is-at 00:00:00:00:00:09, length 32
2.032105 Acknowledgment RA:00:00:00:00:00:09
2.032237 IP 10.1.2.4.9 > 10.1.3.3.49153: UDP, length 1024
2.033773 Acknowledgment RA:00:00:00:00:00:0a
2.080090 Beacon (ns-3-ssid) [6.0* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS
2.182490 Beacon (ns-3-ssid) [6.0* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS
2.284890 Beacon (ns-3-ssid) [6.0* 9.0 12.0* 18.0 24.0* 36.0 48.0 54.0 Mbit] ESS
```

Association requests establish link between AP and rest of Wifi network



Relevant IP addresses:
10.1.3.3 = client on wifi network
10.1.3.4 = AP on wifi network
10.1.2.4 = server on CSMA network

Beacon interval = 2.5 s
Stop at 10 s

Chapter 7: Tracing

Source material:

- **Online tutorial:**
  https://www.nsnam.org/docs/release/3.28/tutorial/html/tracing.html

More advanced tracing with ns-3

- You can use ascii, pcap tracing or logging to get info from your sim
  - Need to write code to parse output
  - The info you want may not be obtainable by pre-defined mechanisms
- There is another way in ns-3
  - Add your own traces to events you care about
  - Produce output in a convenient form
  - Add hooks to the core that can be accessed by other users later

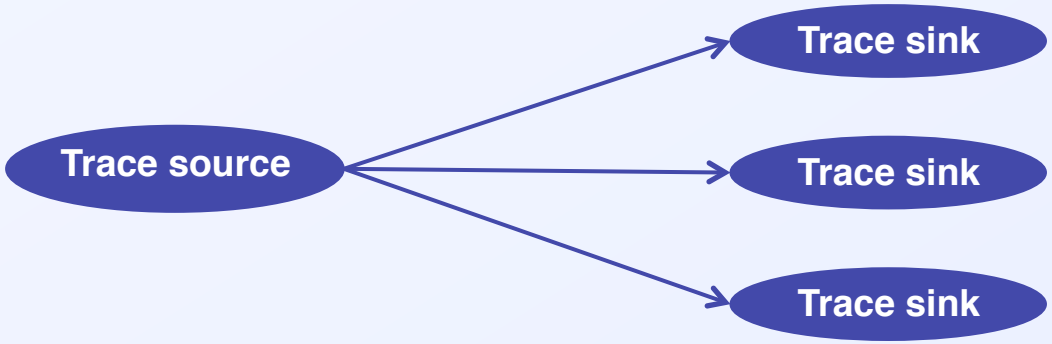
You could use print statements, but I wouldn't recommend it

- You may have to dig deep inside the ns-3 core to find the info you want
- More print statements ⇒ need way to enable/disable specific ones
  - Congratulations! You've just re-invented the ns-3 logging system!
- You could add logging statements to the core
  - Remember: ns-3 is open-source, evolving system
    - Core will bloat to include all possible log messages
    - Unwieldy gigantic log files ⇒ effectively useless
    - No guarantee specific log messages will survive releases

Logging ≠ Tracing

The basic idea: trace sources and sinks

- You need:
  1. Trace source: signals sim event and provides access to the data
  2. Trace sink: consumes the trace info, do something useful with it
  3. Mechanism to connect trace source to trace sink
- Note: there can be many sinks connected to the same trace source



A simple low-level example: callbacks

- This is the key to the way tracing works
- In C/C++ you can pass a pointer to a function: callback mechanism

A numeric function

A function to integrate numeric functions

Integrate myFunction

```
float
myFunction(float x)
{
  return x*x;
}
        ⋮
float
Integrate(float (*func)(float), float lo, float hi)
{
  …
}
        ⋮
Integrate(&myFunction, 0, 1);
```

- Trace source maintains a list of callback functions added by the sinks
- When an event of interest happens
  - Trace source passes data to the callback functions
  - Callback functions are executed

A simple low-level example: `fourth.cc`

```
#include "ns-3/object.h"
#include "ns-3/uinteger.h"
#include "ns-3/traced-value.h"
#include "ns-3/trace-source-accessor.h"

#include <iostream>

using namespace ns-3;

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

  MyObject () {}
  TracedValue<int32_t> m_myInt;
};

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;
}
```

Goal: trace changes made to a variable

- **i.e., notify user whenever**
  - **+, ++, =, etc.**

Trace source

Trace sink

Connection

A simple-low level example: the trace source

MyObject is derived from Object, inherits public members

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

  MyObject () {}
  TracedValue<int32_t> m_myInt;
};
```

Always need GetTypeId method for an object

- Provides run-time info on type
- Allows objects to be located via path (more on this later)
- Provides objects with attributes
- Provides objects with trace sources

A simple-low level example: the trace source

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

  MyObject () {}
  TracedValue<int32_t> m_myInt;
};
```

TypeId class records meta-information about MyObject

Provides unique identifier

Record TypeId of base class

Default constructor is accessible

Provides hook to connect to trace source:
- **Name of source**
- **Help string**
- **Accessor = pointer to connect**

./src/core/model/traced-value.h

Provides infrastructure to
- **Overload operators**
- **Drive callback process**

```
template <typename T> class TracedValue
{
public:
...
private:

  T m_v; /** The underlying value. */
  TracedCallback<T,T> m_cb; /** The connected Callback. */
};
```

A simple low-level example: the trace sink

```
void IntTrace (int32_t oldValue, int32_t newValue)
{
  std::cout << "Traced " << oldValue << " to " << newValue << std::endl;
}
```

A simple void function

- **Takes in the old and new values of the traced variable**
  - **Will be supplied by the trace source**
- **Prints them to the screen**

A low-level example: the main function

ns-3 smart pointers provide garbage collection via reference counting
Track number of pointers to an object
Helps avoid memory leaks when you forget to delete an object

Wrapper for C++ "new"

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

  myObject->m_myInt = 1234;
}
```

Connecting source to sink

- **"MyInteger" = name of source**
- **Callback made from our sink function**
- **We'll talk about "Context" next…**

If all goes as planned: should trigger callback

```
$ find . -name '*.h' | xargs grep TraceConnectWithoutContext
./src/core/model/object-base.h:  bool TraceConnectWithoutContext (std::string name, const
CallbackBase &cb);
```

A low-level example: running fourth.cc

```
$ cp examples/tutorial/fourth.cc scratch/myfourth.cc
$ ./waf --run scratch/myfourth
```

```
Traced 0 to 1234
```

Assignment in: myObject->m_myInt = 1234; triggers callback

Seems almost anticlimactic, but we've illustrated the main steps in tracing

- **Define trace source**
- **Define trace sink**
- **Connect trace source to trace sink**

Using Config to connect to trace sources

- `TraceConnectWithoutContext` is not typical way to connect source to sink
- Normally done via the Config subsystem in ns-3, by using a config path
  - A string that looks like a file path
  - Represents chain of objects leading to the desired event (or attribute)
- e.g. in myfirst.cc

```
Config::ConnectWithoutContext("/NodeList/1/DeviceList/0/
$ns3::PointToPointNetDevice/MacTx", MakeCallback(&MyPacketTrace));
```

Trace source: (config path) "/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacTx"

Trace sink: callback function MyPacketTrace

- **Config::ConnectWithoutContext**
  Ties the trace source to the trace sink callback function

How to Find and Connect Trace Sources, and Discover Callback Signatures

- What trace sources are available (besides `MacT`)?
- How do I figure out the config path I need to connect to this source?
- What are the return type and arguments of my callback function?

```
void MyPacketTrace (Ptr< const Packet > packet)
```

```
void IntTrace (int32_t oldValue, int32_t newValue)
```

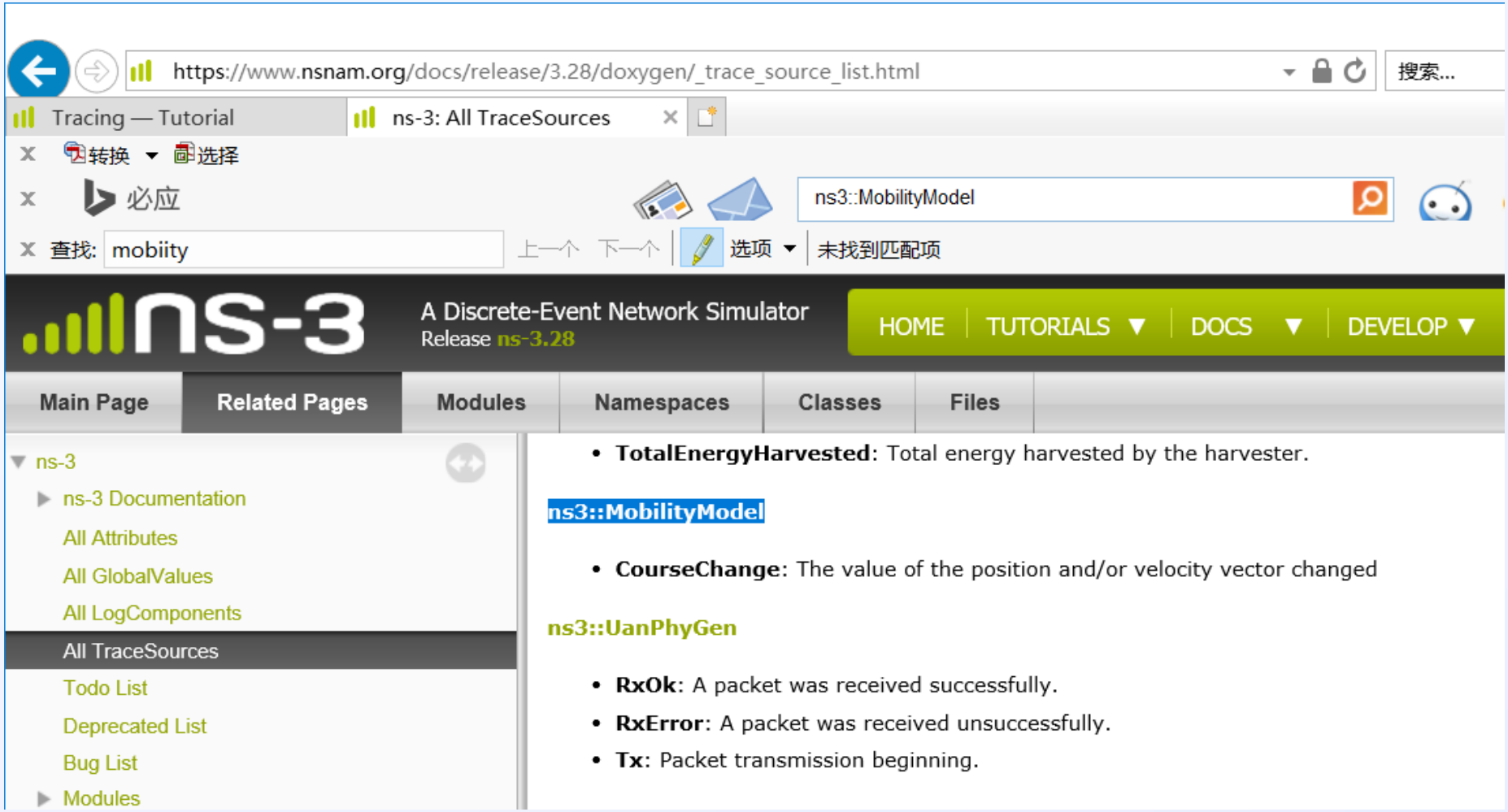
Doxygen and a little detective work will go a long way here

What trace sources are available?

- **Doxygen has the answer!**
  - **Trace sources:** https://www.nsnam.org/docs/release/3.28/doxygen/_trace_source_list.html
  - **Attributes:** https://www.nsnam.org/docs/release/3.28/doxygen/_attribute_list.html
  - **Global values:** https://www.nsnam.org/docs/release/3.28/doxygen/_global_value_list.html

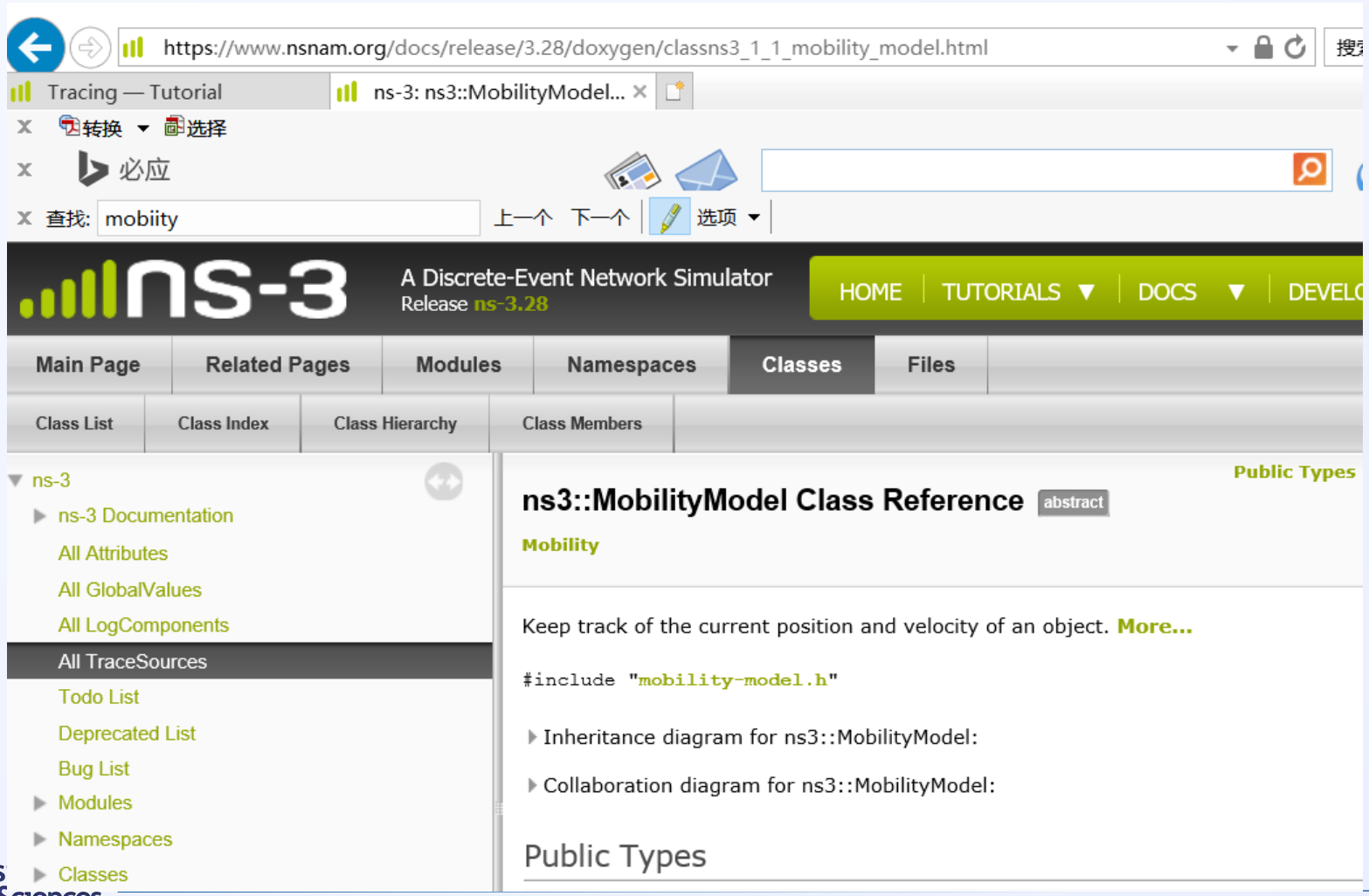
What trace sources are available?

https://www.nsnam.org/docs/release/3.28/doxygen/_trace_source_list.html



What string do I use to connect?

click on ns3::MobilityModel and go to ns3::MobilityModel Class Reference site



What string do I use to connect?

- **Scroll down the page to locate the Config Paths and TraceSources**

Detailed Description

Keep track of the current position and velocity of an object.

All space coordinates in this class and its subclasses are understood to be meters or meters/s. i.e., they are a

This is a base class for all specific mobility models.

Config Paths

ns3::MobilityModel is accessible through the following paths with **Config::Set** and **Config::Connect**:

- "/NodeList/[i]/$ns3::MobilityModel"

Attributes

TraceSources

- **CourseChange**: The value of the position and/or velocity vector changed
  Callback signature: **ns3::MobilityModel::TracedCallback**

Size of this type is 48 bytes (on a 64-bit architecture).

Definition at line **39** of file **mobility-model.h**.

trace source: /NodeList/[i]/$ns3::MobilityModel/CourseChange

What return value and arguments for the callback function?

TraceSources

- **CourseChange**: The value of the position and/or velocity vector changed

Callback signature: **ns3::MobilityModel::TracedCallback**

Size of this type is 48 bytes (on a 64-bit architecture).

Follow this link to find the return value and arguments of the callback function

typedef void(* ns3::MobilityModel::TracedCallback) (Ptr< const MobilityModel > model)

TracedCallback signature.

Parameters

[in] **model** Value of the MobilityModel.

Definition at line 87 of file mobility-model.h.

What return value and arguments for the callback function?

- The somewhat easy way
  - The return value is always “void”
  - Traced callback signature

    or in "src/mobility/model/mobility-model.h”

```
typedef void (* TracedCallback)(Ptr<const MobilityModel> model);
```

This is the argument we need!

- So for callback using Config::ConnectWithoutContext

```
void CourseChange(Ptr<const MobilityModel> model)
```

- And for callback using Config::Connect

```
void CourseChange(std::string path,Ptr<const MobilityModel> model)
```

Context (= config path) gets passed here

Learn from other people's code

- look for config path in someone else's code
  - `$ find -name "*.cc" | xargs grep CourseChange | grep Connect`
    - Inside "`./src/mobility/examples/main-random-walk.cc`":

```
Config::Connect ("/NodeList/*/$ns-3::MobilityModel/CourseChange",
                 MakeCallback (&CourseChange));
```

  - Config path = "/NodeList/*/$ns-3::MobilityModel/CourseChange"
    - And we then find the declaration of the "CourseChange" function:

```
static void CourseChange (std::string foo, Ptr<const MobilityModel> mobility)
```

A real example

From W. Richard Stevens, "TCP/IP Illustrated, Vol. I: The Protocols", Addison-Wesley (1994)

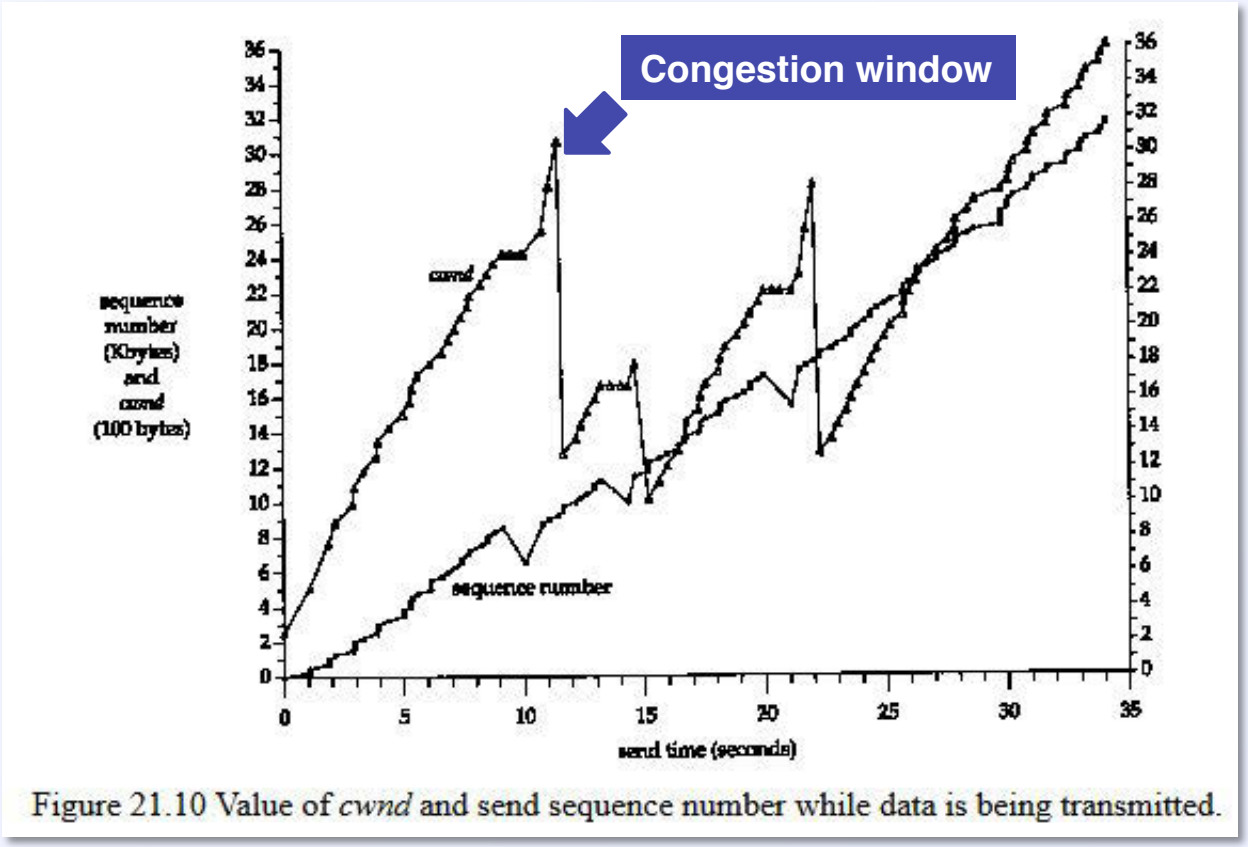


Figure 21.10 Value of *cwnd* and send sequence number while data is being transmitted.

Goal: simulate a TCP congestion window and look at the effect of dropped packets

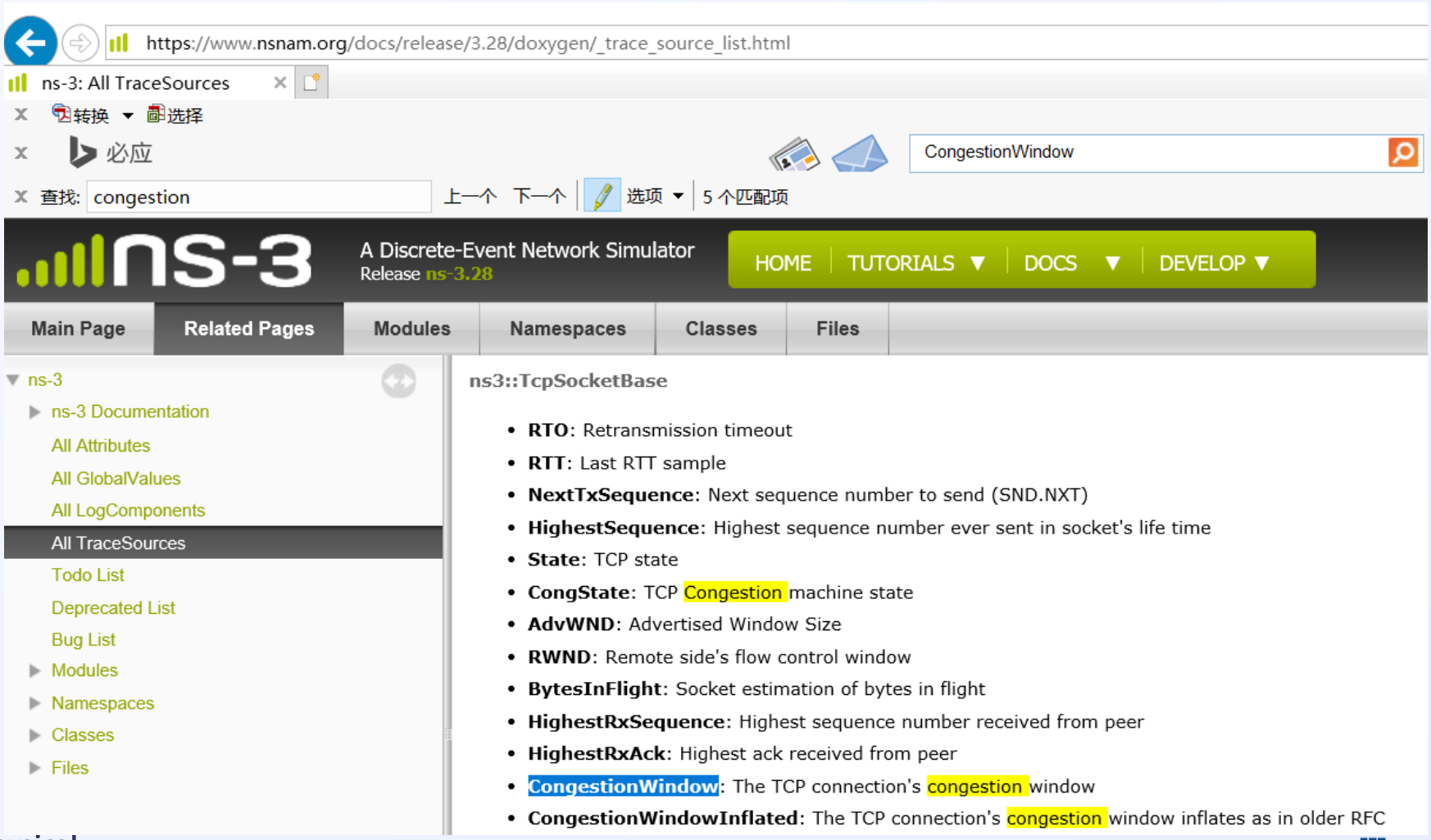
Congestion windows 101 (from Wikipedia)

- Congestion collapse occurs at choke points in the network
  - Wherever total incoming traffic to a node > outgoing bandwidth
  - e.g., connection points between LAN and WAN
- TCP uses network congestion avoidance algorithm
- Congestion window is part of TCP strategy to avoid congestion collapse
  - Limits total number of unacknowledged packets that may be in transit
  - The size of the window is adjusted dynamically by the sender
    - Based on how much congestion between sender and receiver
    - If all segments are received and the acks reach the sender on time
      - Add a constant to the window (typically 1 MSS = Max Segment Size)
    - Otherwise
      - Scale back by a set factor (typically 1/2)

Explains "sawtooth" shape of congestion window over time

Are there trace sources available?

- From Doxygen's "The list of all trace sources": search for congestionwindow



Are there trace sources available?

ns3::TcpSocketBase Class Reference

TraceSources

- **RTO**: Retransmission timeout
  Callback signature: **ns3::TracedValueCallback::Time**
- **RTT**: Last RTT sample
  Callback signature: **ns3::TracedValueCallback::Time**

.....

- **CongestionWindow**: The TCP connection's congestion window
  Callback signature: **ns3::TracedValueCallback::Uint32**

.....

typedef void(* ns3::TracedValueCallback::Uint32) (uint32_t oldValue, uint32_t newValue)

TracedValue Callback signature for POD.

these are the arguments of your callback function

Parameters

[in] **oldValue** original value of the traced variable

[in] **newValue** new value of the traced variable

Definition at line **88** of file **traced-value.h**.

How to connect ? Config or myObject->TraceConnect?

- learn from other people's code

$ find -name “*.cc” | xargs grep CongestionWindow | grep Connect

```
junhuatang@ubuntu:~/workspace/ns-allinone-3.28/ns-3.28$ find -name "*.cc" | xar
gs grep CongestionWindow | grep Connect
./examples/tutorial/fifth.cc:  ns3TcpSocket->TraceConnectWithoutContext ("Conge
stionWindow", MakeCallback (&CwndChange));
./examples/tutorial/sixth.cc:  ns3TcpSocket->TraceConnectWithoutContext ("Conge
stionWindow", MakeBoundCallback (&CwndChange, stream));
./examples/tutorial/seventh.cc:  ns3TcpSocket->TraceConnectWithoutContext ("Con
gestionWindow", MakeBoundCallback (&CwndChange, stream));
./examples/tcp/tcp-large-transfer.cc:  Config::ConnectWithoutContext ("/NodeLis
t/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow", MakeCallback (&CwndTrac
er));
./examples/tcp/tcp-variants-comparison.cc:  Config::ConnectWithoutContext ("/No
deList/1/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow", MakeCallback (&Cwn
dTracer));
```

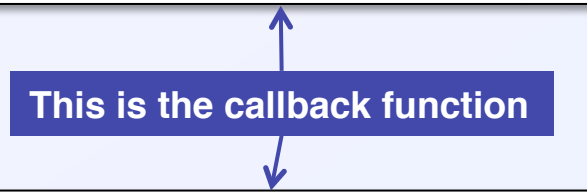
……

```
./src/test/ns3tcp/ns3tcp-cwnd-test-suite.cc:  ns3TcpSocket->TraceConnectWithout
Context ("CongestionWindowInflated", MakeCallback (&Ns3TcpCwndTestCase2::CwndCh
ange, this));
./src/test/ns3tcp/ns3tcp-cwnd-test-suite.cc:  ns3TcpSocket->TraceConnectWithout
Context ("CongestionWindow", MakeCallback (&Ns3TcpCwndTestCase2::CwndChangeNotI
nflated, this));
junhuatang@ubuntu:~/workspace/ns-allinone-3.28/ns-3.28$
```

How do we get started writing our script?

- The time-honored way: steal from someone else's code
  - `$ find -name "*.cc" | xargs grep CongestionWindow`
  - Look, e.g., in "`./src/test/ns-3tcp/ns-3tcp-cwnd-test-suite.cc`"
  - The connection between trace and source is done by

```
ns-3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",
MakeCallback (&ns-3TcpCwndTestCase1::CwndChange, this));
```




```
void CwndChange (uint32_t oldCwnd, uint32_t newCwnd)
```

  - We can also copy code from the function

```
void ns-3TcpCwndTestCase1::DoRun (void)
```

This is how `fifth.cc` was put together

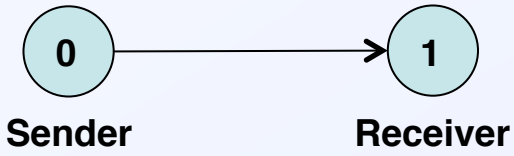
Avoiding a common mistake in ns-3

- ns-3 scripts execute in three separate stages
  - Configuration time
  - Simulation time (i.e., Simulator::Run)
  - Teardown time
- TCP uses sockets to connect nodes
  - Sockets are created dynamically during simulation time
    - Want to hook the `CongestionWindow` on the socket of the sender
  - Connection to trace sources is established during configuration time
  - Can't put the cart before the horse!
- Solution:
  1. Create socket at configuration time
  2. Hook trace source then
  3. Pass this socket object to system during simulation time

Next: `fifth.cc` walkthrough

Overview of `fifth.cc`

- Dumbbell topology, point-to-point network, like `first.cc`



- We will create our own application and socket
  - So we can access socket at configuration time
  - No helper, so we'll have to do the work manually
  - Connect to `CongestionWindow` trace source in sender socket
- Introduce errors into the channel between nodes
  - Dropped packets ⇒ interesting behavior in congestion window

Creating our own application: the MyApp class

```
class MyApp : public Application
{
public:

  MyApp ();
  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address,
              uint32_t packetSize, uint32_t nPackets, DataRate dataRate);

private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTx (void);
  void SendPacket (void);

  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
};
```

Initializes member variables

But this is the important bit: we will be able to
- Create socket
- Hook its CongestionWindow trace source
- Pass the socket to the application

All this at configuration time!

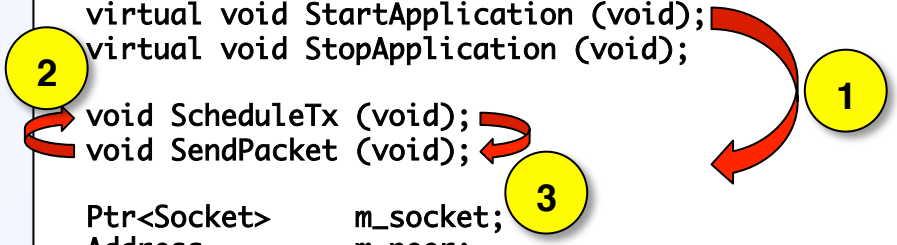
Creating our own application: the MyApp class

```
class MyApp : public Application
{
public:

  MyApp ();
  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address,
              uint32_t packetSize, uint32_t nPackets, DataRate dataRate);

private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTx (void);
  void SendPacket (void);

  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
};
```

We also need to override these with our own implementations that will be called by the simulator to start and stop

Creating our own application: the MyApp class

```
class MyApp : public Application
{
public:

  MyApp ();
  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address,
              uint32_t packetSize, uint32_t nPackets, DataRate dataRate);

private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);

  void ScheduleTx (void);
  void SendPacket (void);

  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
};
```



1. StartApplication **calls** SendPacket
2. SendPacket **calls** ScheduleTx
3. ScheduleTx **set up next** SendPacket **call**
4. ...
5. **Until** StopApplication

The trace sinks

- We want to trace 2 types of events
  - Updates to the congestion window:

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

  - Dropped packets

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

Introducing errors in the channel

Determines which packets will not be received due to transmission errors, corresponding to underlying

- Distribution = random variable
- Rate ↔ mean duration between errors
- Unit = per-bit, per-byte, or per-packet

```
Ptr<RateErrorModel> em = CreateObjectWithAttributes<RateErrorModel> (
      "RanVar", RandomVariableValue (UniformVariable (0., 1.)),
      "ErrorRate", DoubleValue (0.00001));

devices.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));
```

Will cause randomly dropped packets in the receiver device

Setting the application on the receiver node

```
uint16_t sinkPort = 8080;
Address sinkAddress (InetSocketAddress (interfaces.GetAddress (1), sinkPort));
PacketSinkHelper packetSinkHelper ("ns-3::TcpSocketFactory",
                                   InetSocketAddress (Ipv4Address::GetAny (), sinkPort));
ApplicationContainer sinkApps = packetSinkHelper.Install (nodes.Get (1));
sinkApps.Start (Seconds (0.));
sinkApps.Stop (Seconds (20.));
```

- `PacketSink` receives and consumes traffic generated to an IP address and port
- `PacketSinkHelper` creates sockets using an "object factory"
  - Object factories are used to mass produce similarly configured objects
  - Factory method doesn't require you to know the type of the objects created

Connecting the sender's socket, and connecting the trace source

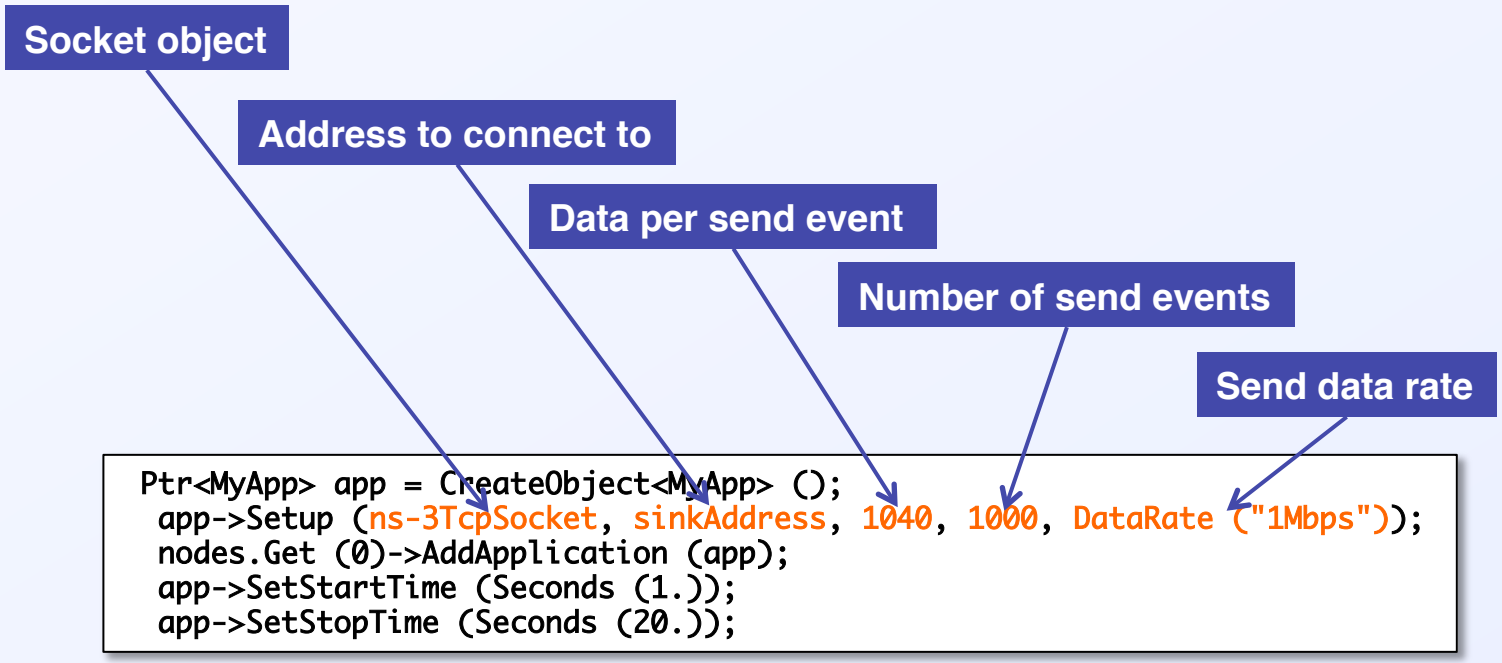
Another way of creating a socket using a socket factory

```
Ptr<Socket> ns-3TcpSocket = Socket::CreateSocket (nodes.Get (0), TcpSocketFactory::GetTypeId ());
ns-3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeCallback (&CwndChange));
```

This should look familiar by now

Because we created our own app and socket at configuration time we can hook into its `CongestionWindow` trace source

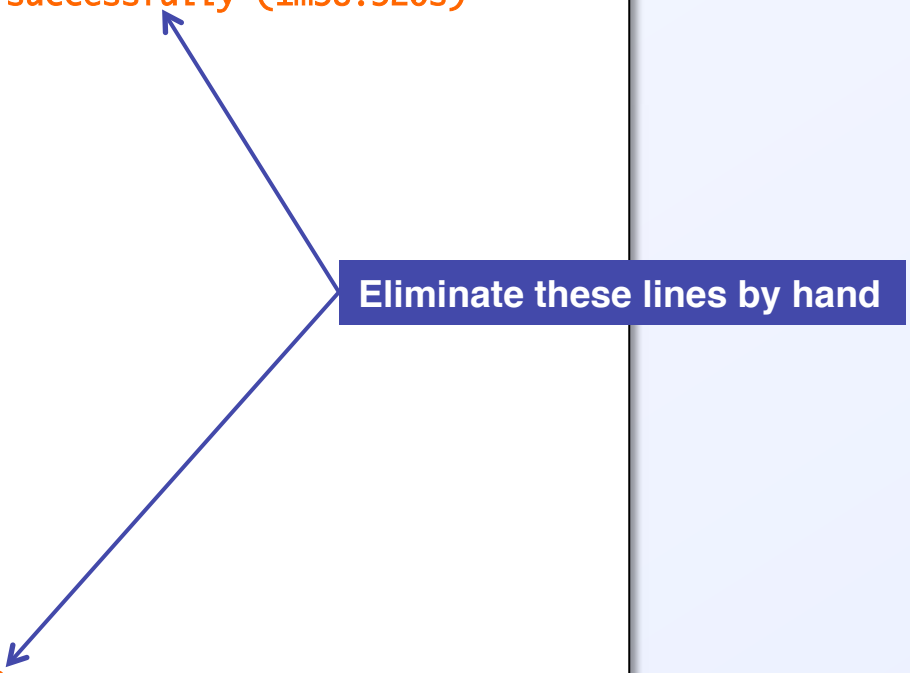
Setting up the application on the sender node



Running `fifth.cc`: text output

```
$ ./waf -run scratch/myfifth > cwnd.dat 2>&1
```

```
Waf: Entering directory `/mypath/ns-3-dev/build'
Waf: Leaving directory `/mypath/ns-3-dev/build'
'build' finished successfully (1m58.520s)
1       536
1.00919 1072
1.01511 1608
1.02163 2144
1.02995 2680
1.03827 3216
1.04659 3752
1.05491 4288
1.06323 4824
1.07155 5360
1.07987 5896
1.08819 6432
1.09651 6968
1.10483 7504
1.11315 8040
1.12147 8576
1.12979 9112
RxDrop at 1.13692
1.13811 9648
1.15475 2900
1.15563 3436
          ⋮
```



Running `fifth.cc`: plotting the results

Original by W. Richard Stevens

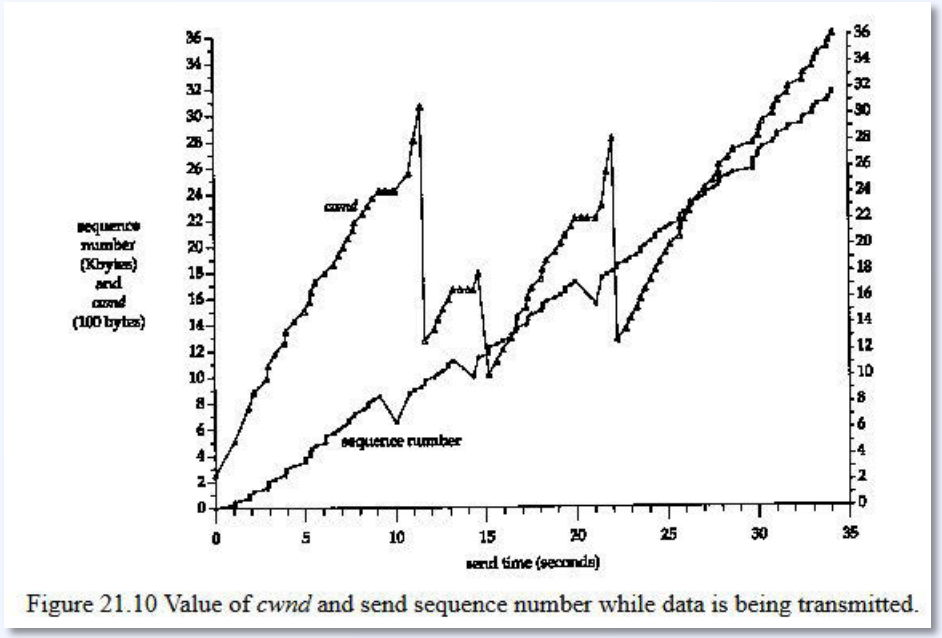
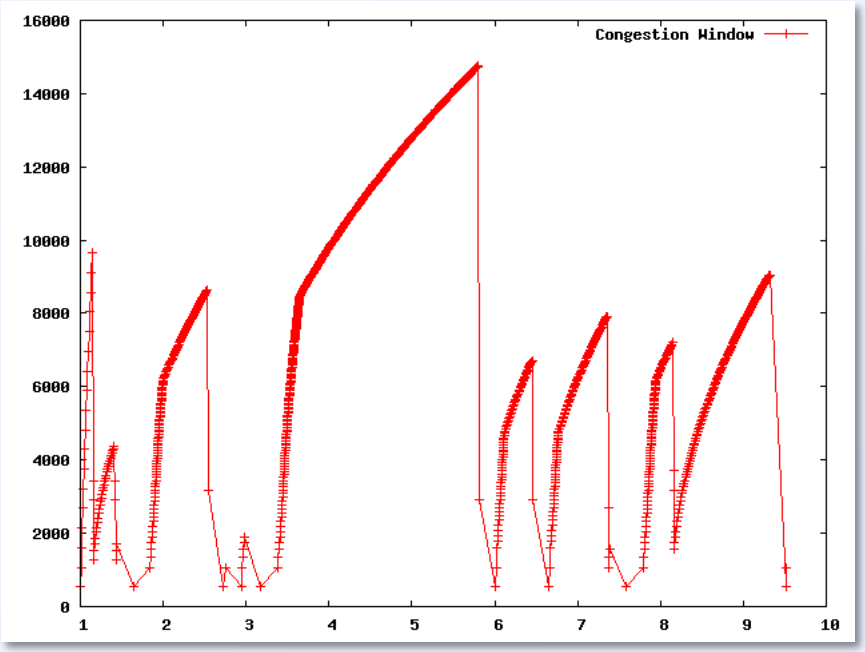


Figure 21.10 Value of *cwnd* and send sequence number while data is being transmitted.

Our result, using gnuplot



That's nice, but…

- Remember after

```
$ ./waf –run scratch/myfifth > cwnd.dat 2>&1
```

- We had to edit the file by hand to remove "junk" lines…

- But we said tracing gives you control over output format
- Is there a cleaner way to produce the output we need?
- Yes! Use trace helpers

Let's tweak `fifth.cc` to produce cleaner output ⇒ `sixth.cc`

A sixth.cc walkthrough: CwndChange callback

- **Modify the callback function:**

```
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;
}
```

Added to fifth.cc

Formatted output to the stream

- **Add lines in the main to create stream:**

Filename attached to stream

```
  AsciiTraceHelper asciiTraceHelper;
  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("sixth.cwnd");
  ns-3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",
                                             MakeBoundCallback (&CwndChange, stream));
```

Causes the stream argument to be added to the function callback

A sixth.cc walkthrough: RxDrop callback

- **Modify the callback function:**

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

Formatted output to the pcap file

Added to fifth.cc

- **Add lines in the main to create stream:**

Filename attached to file

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

Causes the file argument to be added to the function callback

Running sixth.cc

```
$ ./waf -run scratch/mysixth
```

Console

```
Waf: Entering directory `…
Waf: Leaving directory `…
'build' finished successfully (6.002s)
1       536
1.00919 1072
1.01511 1608
1.02163 2144
1.02995 2680
1.03827 3216
1.04659 3752
1.05491 4288
1.06323 4824
1.07155 5360
1.07987 5896
1.08819 6432
1.09651 6968
1.10483 7504
1.11315 8040
1.12147 8576
1.12979 9112
RxDrop at 1.13692
1.13811 9648
1.15475 2900
1.15563 3436
    ⋮
```

sixth.cwnd

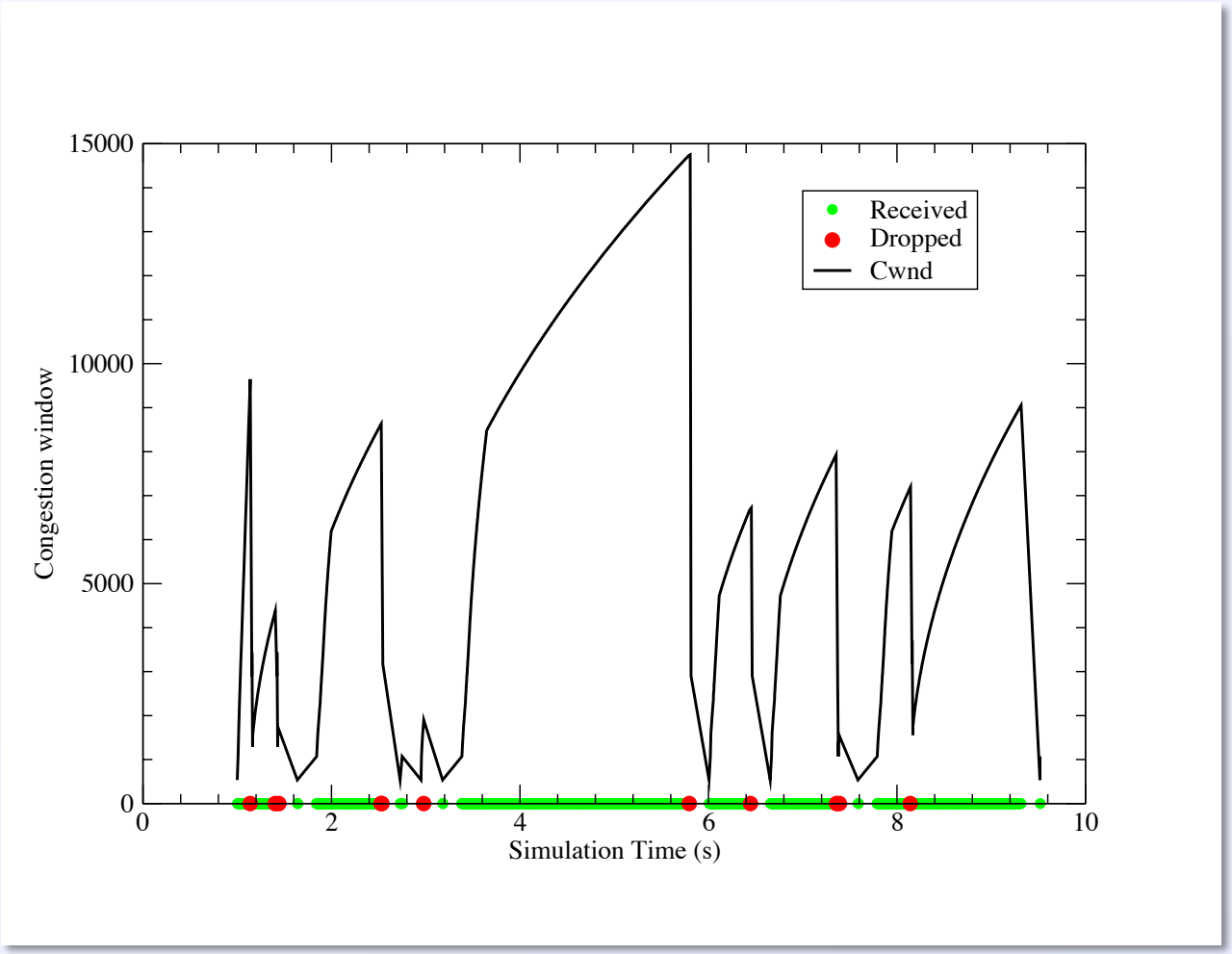
```
1       0       536
1.00919 536     1072
1.01511 1072    1608
1.02163 1608    2144
1.02995 2144    2680
1.03827 2680    3216
1.04659 3216    3752
1.05491 3752    4288
1.06323 4288    4824
1.07155 4824    5360
1.07987 5360    5896
1.08819 5896    6432
1.09651 6432    6968
1.10483 6968    7504
1.11315 7504    8040
1.12147 8040    8576
1.12979 8576    9112
1.13811 9112    9648
⋮
```

sixth.pcap

```
reading from file sixth.pcap, link-type PPP (PPP)
1.136918 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 17177:17681, ack 1, win 65535, le
1.403158 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 33280:33784, ack 1, win 65535, le
1.440505 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 37440:37944, ack 1, win 65535, le
2.525484 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 173144:173680, ack 1, win 65535,
2.534678 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 174720:175224, ack 1, win 65535,
2.977574 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 180552:181088, ack 1, win 65535,
5.796117 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 582400:582904, ack 1, win 65535,
6.445077 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 663520:664024, ack 1, win 65535,
7.359404 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 777384:777920, ack 1, win 65535,
7.389304 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 781040:781544, ack 1, win 65535,
8.141483 IP 10.1.1.1.49153 > 10.1.1.2.8080: Flags [.], seq 875144:875680, ack 1, win 65535,
```

Find out more: what happens to uncorrupted packets?

PhyRxEnd: Trace source indicating a packet has been completely received by the device



Using trace helpers

- We've encountered trace helpers before

```
pointToPoint.EnablePcapAll ("second");
pointToPoint.EnablePcap ("second", p2pNodes.Get (0)->GetId (), 0);
csma.EnablePcap ("third", csmaDevices.Get (0), true);
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("myfirst.tr"));
```

- What other trace helpers are available?
- How do we use them?
- What do they have in common?

Two categories of trace helpers

- Device helpers
  - Trace is enabled for node/device pair(s)
  - Both pcap and ascii traces provided
  - Conventional filenames: `<prefix>-<node id>-<device id>`
- Protocol helpers
  - Trace is enabled for protocol/interface pair(s)
  - Both pcap and ascii traces provided
  - Conventional filenames: `<prefix>-n<node id>-i<interface id>`
    - "`n`" and "`i`" to avoid filename collisions with node/device traces

ns-3 uses “mixin” classes to ensure tracing works the same way across all devices or interfaces

- **Mixin classes in ns-3 (see Doxygen for more info):**

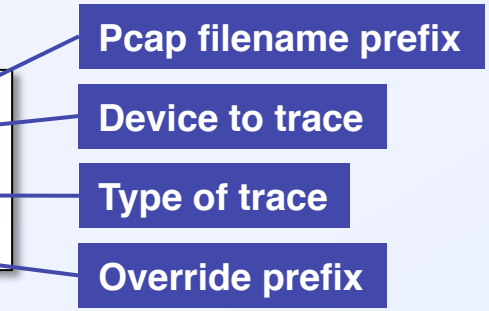
	PCAP	ASCII
Device Helper	PcapHelperForDevice	AsciiTraceHelperForDevice
Protocol Helper	PcapHelperForIpv4	AsciiTraceHelperForIpv4

- **These mixin classes each provide a single virtual method to enable trace**
  - **All device or protocols must implement this method**
  - **All other methods of the mixin class call this one method**
  - **Provides consistent functionality across different devices & interfaces**

The PcapHelperForDevice mixin class

- Every device must implement

```
virtual void EnablePcapInternal (std::string prefix,
                                 Ptr<NetDevice> nd,
                                 bool promiscuous,
                                 bool explicitFilename) = 0;
```



- All other methods of `PcapHelperForDevice` call this one

⇒ Consistency across devices

Pcap Tracing Device Helper: EnablePcap methods

- EnablePcap for various node/device pair(s)
  - Provide Ptr<NetDevice>
  - Provide device name using the ns-3 object name service

```
Names::Add ("server" ...);
Names::Add ("server/eth0" ...);a
...
helper.EnablePcap ("prefix", "server/ath0");
```

  - Provide a NetDeviceContainer
  - Provide a NodeContainer
  - Provide integer node and device ids
- Enable pcap tracing for all devices in the simulation

```
helper.EnablePcapAll ("prefix");
```

Pcap Tracing Device Helper: filename selection

- By convention: `<prefix>-<node id>-<device id>.pcap`
- Can use the ns-3 object name service to replace ids with meaningful names e.g.,

```
prefix-21-1.pcap
```

```
Names::Add("server1",serverNode);
Names::Add("server1/eth0",serverDevice);
```

```
prefix-server1-eth0.pcap
```

- You can override the naming convention, e.g.

```
void EnablePcap(std::string prefix,
                Ptr<NetDevice> nd,
                bool promiscuous,
                bool explicitFilename);
```

Set this to true
prefix becomes filename

Ascii Tracing Device Helpers

- The mixin class is `AsciiTraceHelperForDevice`
  - All device implement virtual `EnableAsciiInternal` method
  - All other methods of `AsciiTraceHelperForDevice` will call this one
- Can provide `EnableAscii` with `Ptr<NetDevice>`, string from name service, `NetDeviceContainer`, `NodeContainer`, integer node/device ids
- Or `helper.EnableAsciiAll("prefix");`
- Can also dump ascii traces to a single common file, e.g.,

```
Ptr<NetDevice> nd1;
Ptr<NetDevice> nd2;
...
Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream ("trace-file-name.tr");
...
helper.EnableAscii (stream, nd1);
helper.EnableAscii (stream, nd2);
```

- So there are twice as many trace methods as for pcap

Ascii Tracing Device Helpers: filename selection

- By convention: `<prefix>-<node id>-<device id>.tr`
- Using ns-3 object name service, can assign names to the id, then e.g.
  - "`prefix-21-1.tr`" → "`prefix-server-eth0.tr`"
- Many `EnableAscii` methods offer a `explicitFilename` option
  - Set to true ⇒ override naming convention, use your own file name

Pcap Tracing Protocol Helpers

- The mixin class is `PcapHelperForIpv4`
  - All device implement virtual `EnablePcapIpv4Internal` method
  - All other methods of `PcapHelperForIpv4` will call this one
- Can provide `EnablePcapIpv4` with `Ptr<Ipv4>`, string from name service, `Ipv4InterfaceContainer`, `NodeContainer`, integer node/device ids
- Or `helper.EnablePcapIpv4All("prefix");`
- Filename selection
  - Convention is `<prefix>-n<node id>-i<interface id>.pcap`
  - Can also use the ns-3 object name service clarity
  - `explicitFilename` parameter lets you impose your own filenames

Ascii Tracing Protocol Helpers

- The mixin class is `AsciiTraceHelperForIpv4`
  - All device implement virtual `EnableAsciiIpv4Internal` method
  - All other methods of `AsciiTraceHelperForIpv4` will call this one
- Can provide `EnableAsciiIpv4` with `Ptr<Ipv4>`, string from name service, `Ipv4InterfaceContainer`, `NodeContainer`, integer node/device ids
- Or `helper.EnableAsciiIpv4All("prefix");`
- Can also dump ascii traces to a single common file
  - So there are twice as many trace methods as for pcap
- Filename selection
  - Convention is `<prefix>-n<node id>-i<interface id>.pcap`
  - Can also use the ns-3 object name service clarity
  - `explicitFilename` parameter lets you impose your own filenames

Conclusion

- ns-3 is very powerful/comprehensive
- Lots of tools for you to use
  - Helpers
  - Containers
  - Logging
  - Tracing
  - Models
- Doxygen is your friend!

Practice! Practice! Practice!