ns-3 Training

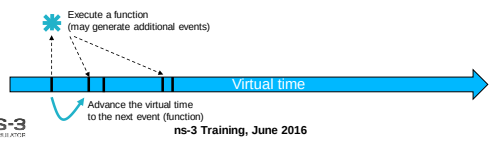
ns-3 Annual Meeting
June 2016

Software overview

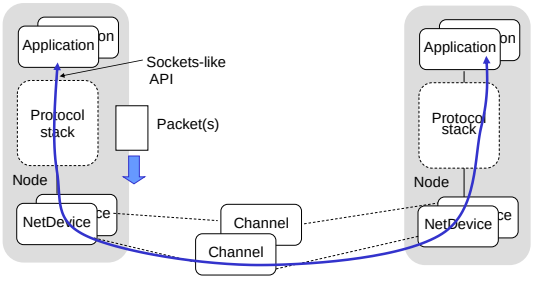
- ns-3 is written in C++, with bindings available for Python
  - simulation programs are C++ executables or Python programs
  - ~350,000 lines of C++ (cloc estimate)
  - almost exclusively C++98, beginning to use C++11
- ns-3 is a GNU GPLv2-licensed project
- ns-3 is mainly supported for Linux, OS X, and FreeBSD
  - Windows Visual Studio port available
- ns-3 is not backwards-compatible with ns-2

Discrete-event simulation basics

- Simulation time moves in discrete jumps from event to event
- C++ functions schedule events to occur at specific simulation times
- A simulation scheduler orders the event execution
- Simulation::Run() executes a single-threaded event list
- Simulation stops at specific time or when events end



The basic ns-3 architecture



Software orientation

Key differences from other network simulators:

1) Command-line, Unix orientation
   - vs. Integrated Development Environment (IDE)

2) Simulations and models written directly in C++ and Python
   - vs. a domain-specific simulation language

ns-3 does not have a graphical IDE

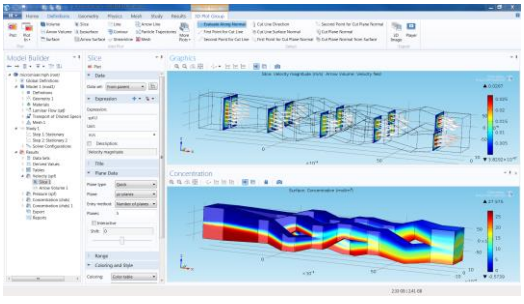
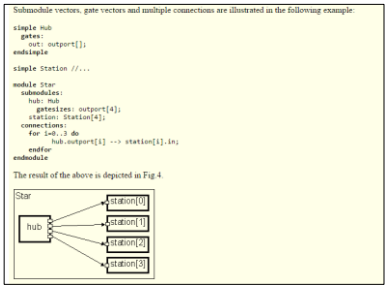


Figure source: https://www.comsol.com/comsol-multiphysics

ns-3 not written in a high-level language

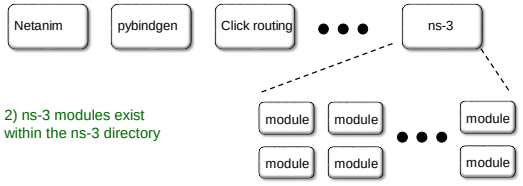


Example of OMNeT++ Network Description (NED) language
Figure excerpted from http://www.ewh.ieee.org/soc/es/Nov1999/18/ned.htm

Software organization

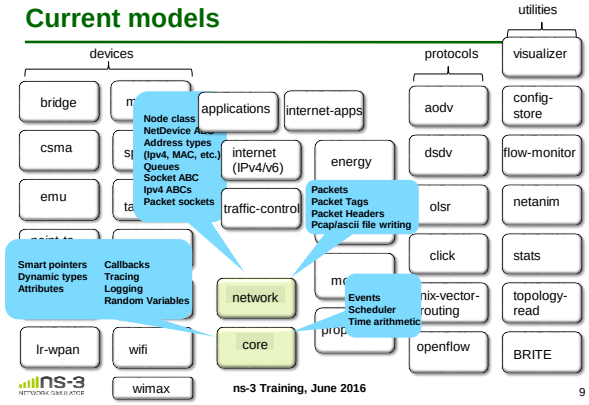
- Two levels of ns-3 software and libraries

1) Several supporting libraries, not system-installed, can be in parallel to ns-3



2) ns-3 modules exist within the ns-3 directory

Current models



Module organization

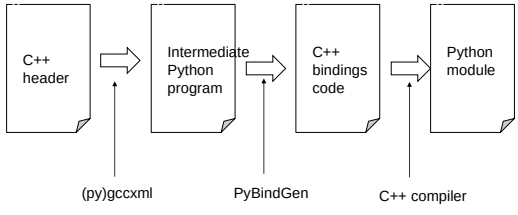
- models/
- examples/
- tests/
- bindings/
- doc/
- wscript

ns-3 programs

- ns-3 programs are C++ executables that link the needed shared libraries
  - or Python programs that import the needed modules
- The ns-3 build tool, called 'waf', can be used to run programs
- waf will place headers, object files, libraries, and executables in a 'build' directory

Python bindings

- ns-3 uses a program called PyBindGen to generate Python bindings for all libraries



Integrating other tools and libraries

Other libraries

- more sophisticated scenarios and models typically leverage other libraries
- ns-3 main distribution uses optional libraries (libxml2, gsl, mysql) but care is taken to avoid strict build dependencies
- the 'bake' tool (described later) helps to manage library dependencies
- users are free to write their own Makefiles or wscripts to do something special

Gnuplot

- `src/tools/gnuplot.{cc,h}`
- C++ wrapper around gnuplot
- classes:
  - `Gnuplot`
  - `GnuplotDataset`
    - `Gnuplot2dDataset, Gnuplot2dFunction`
    - `Gnuplot3dDataset, Gnuplot3dFunction`

Enabling gnuplot for your code

- `examples/wireless/wifi-clear-channel-cmu.cc`

```
CommandLine cmd;
cmd.Parse (argc, argv);

Gnuplot gnuplot = Gnuplot ("clear-channel.eps");

for (uint32_t i = 0; i < modes.size (); i++)
  {
    std::cout << modes[i] << std::endl;
    Gnuplot2dDataset dataset (modes[i]);
```

produce a plot file that will generate an EPS figure

one dataset per mode

```
    uint32_t pktsRecvd = experiment.Run (wifi, wifiPhy, wifiMac, wifiChannel);
    dataset.Add (rss, pktsRecvd);
  }

gnuplot.AddDataset (dataset);
```

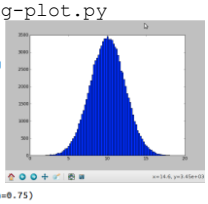
Add data to dataset

Add dataset to plot

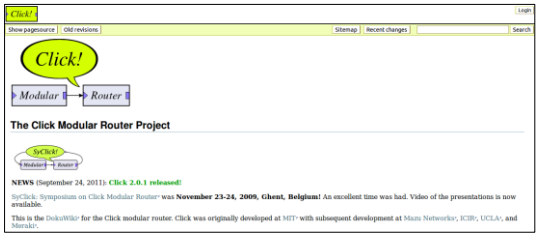
Matplotlib

- `src/core/examples/sample-rng-plot.py`

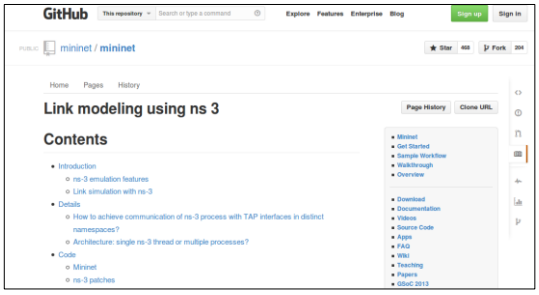
```
# Demonstrate use of ns-3 as a random number generator integrated
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial

import numpy as np
import matplotlib.pyplot as plt
import ns.core

# mu, var = 100, 225
rng = ns.core.NormalVariable(100.0, 225.0)
x = [rng.GetValue() for t in range(10000)]

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.title('ns-3 histogram')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



Click Modular Router



mininet emulator



Co-simulation frameworks have emerged

- PNNL's FNCS framework integrates ns-3 with transmission and distribution simulators

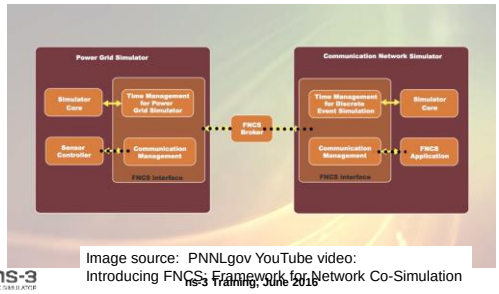


Image source: PNNLgov YouTube video:
Introducing FNCS: Framework for Network Co-Simulation

FAQs

- Does ns-3 have a Windows version?
  - Yes, for Visual Studio 2012
  - http://www.nsnam.org/wiki/Ns-3_on_Visual_Studio_2012
- Does ns-3 support Eclipse or other IDEs?
  - Instructions have been contributed by users
  - http://www.nsnam.org/wiki/HOWTO_configure_Eclipse_with_ns-3
- Is ns-3 provided in Linux or OS X package systems (e.g. Debian packages)?
  - Not officially, but some package maintainers exist
- Does ns-3 support NRL protolib applications?
  - Not yet

Summarizing

- ns-3 models are written in C++ and compiled into libraries
  - Python bindings are optionally created
- ns-3 programs are C++ executables or Python programs that call the ns-3 public API and can call other libraries
- ns-3 is oriented towards the command-line
- ns-3 uses no domain specific language
- ns-3 is not compatible with ns-2

Finding documentation and code

Resources

Web site:

http://www.nsnam.org

Mailing lists:

https://groups.google.com/forum/#!forum/ns-3-users

http://mailman.isi.edu/mailman/listinfo/ns-developers

Wiki:

http://www.nsnam.org/wiki/

Tutorial:

http://www.nsnam.org/docs/tutorial/tutorial.html

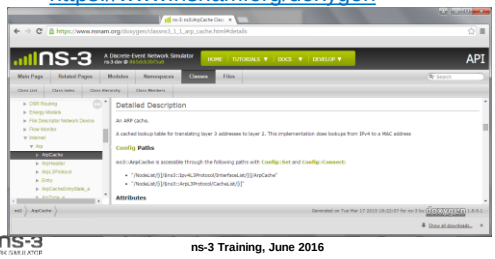
IRC: #ns-3 at freenode.net

Suggested steps

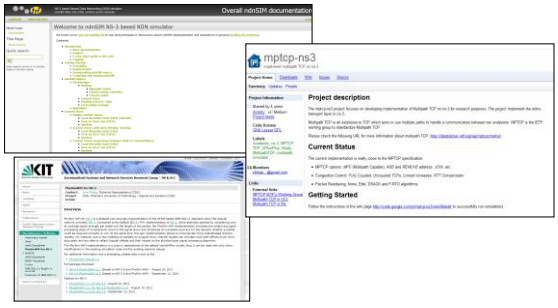
- Work through the ns-3 tutorial
- Browse the source code and other project documentation
  - manual, model library, Doxygen, wiki
  - ns-3 Consortium tutorials (May 2014)
    - https://www.nsnam.org/consortium/activities/training/
- Ask on ns-3-users mailing list if you still have questions
  - We try to answer most questions

APIs

- Most of the ns-3 API is documented with Doxygen
  - https://www.nsnam.org/doxygen



Contributed code and associated projects



Reading existing code

- Much insight can be gained from reading ns-3 examples and tests, and running them yourselves
- Several core features of ns-3 are only demonstrated in the core test suite (src/core/test)
- Stepping through code with a debugger is informative
  - callbacks and templates make it more challenging than usual
- 'find src -name "*.h" | xargs grep "string..." '

Review

- ns-3 primarily aims to support the networking researcher
- ns-3 is C++ under GPLv2, with Python bindings
- ns-3 is mainly designed for low-level coding, work at the command line, and composition with other tools
  - most users edit or extend the source code
- ns-3 tries to operate according to open source project best current practices
  - everyone is a volunteer
- search for what you need, ask questions when you get stuck, and think about contributing back to the project

ns-3 Waf build system

ns-3 Annual Meeting
June 2016

Software introduction

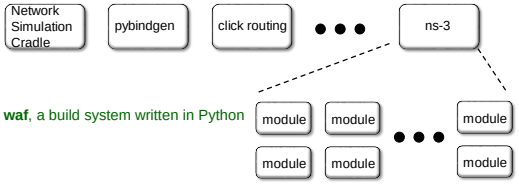
- Download the latest release
  - `wget http://www.nsnam.org/releases/ns-allinone-3.19.tar.bz2`
  - `tar xjf ns-allinone-3.19.tar.bz2`
- Clone the latest development code
  - `hg clone http://code.nsnam.org/ns-3-allinone`

Q. What is "**hg clone**"?
A. Mercurial (http://www.selenic.com) is our source code control tool.

Software building

- Two levels of ns-3 build

1) bake (a Python-based build system to control an ordered build of ns-3 and its libraries)



3) build.py (a custom Python build script to control an ordered build of ns-3 and its libraries) **<--- may eventually be deprecated**

ns-3 uses the 'waf' build system

- Waf is a Python-based framework for configuring, compiling and installing applications.
  - It is a replacement for other tools such as Autotools, Scons, CMake or Ant
  - http://code.google.com/p/waf/
- For those familiar with autotools:
- `configure` → `./waf configure`
- `make` → `./waf build`

waf configuration

- Key waf configuration examples

```
./waf configure
   --enable-examples
   --enable-tests
   --disable-python
   --enable-modules
```

- Whenever build scripts change, need to reconfigure

Demo: `./waf --help`
`./waf configure --enable-examples --enable-tests --enable-modules='core'`
Look at: `build/c4che/_cache.py`

wscript example

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

def build(bld):
    obj = bld.create_ns3_module('csma', ['network', 'applications'])
    obj.source = [
        'model/backoff.cc',
        'model/csma-net-device.cc',
        'model/csma-channel.cc',
        'helper/csma-helper.cc',
        ]
    headers = bld.new_task_gen(features=['ns3header'])
    headers.module = 'csma'
    headers.source = [
        'model/backoff.h',
        'model/csma-net-device.h',
        'model/csma-channel.h',
        'helper/csma-helper.h',
        ]

    if bld.env['ENABLE_EXAMPLES']:
        bld.add_subdirs('examples')

    bld.ns3_python_bindings()
```


waf build

- Once project is configured, can build via `./waf build` or `./waf`
- waf will build in parallel on multiple cores
- waf displays modules built at end of build

Demo: `./waf build`

Look at: `build/` libraries and executables

Running programs

- `./waf shell` provides a special shell for running programs
  - Sets key environment variables

```
./waf --run sample-simulator
./waf --pyrun src/core/examples/sample-simulator.py
```


Build variations

- Configuring a build type is done at waf configuration time
- debug build (default): all asserts and debugging code enabled

```
./waf -d debug configure
```

- optimized

```
./waf -d optimized configure
```

- static libraries

```
./waf --enable-static configure
```


Controlling the modular build

- One way to disable modules:
  - `./waf configure --enable-modules='a','b','c'`
- The `.ns3rc` file (found in utils/ directory) can be used to control the modules built
- Precedence in controlling build
  1) command line arguments
  2) .ns3rc in ns-3 top level directory
  3) .ns3rc in user's home directory

Demo how .ns3rc works

Building without wscript

- The scratch/ directory can be used to build programs without wscripts

Demo how programs can be built without wscripts

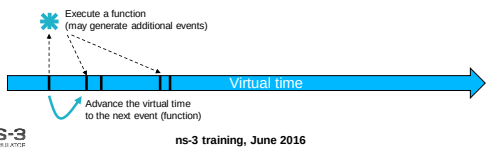
ns-3 Training

Simulator core

ns-3 training, June 2016

Simulator core

- Simulation time
- Events
- Simulator and Scheduler
- Command line arguments
- Random variables



Simulator example

```
#include <iostream>
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include "ns3/command-line.h"
#include "ns3/double.h"
#include "ns3/random-variable-stream.h"

using namespace ns3;
```

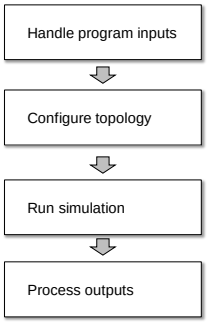
```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  MyModel model;
  Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
  v->SetAttribute ("Min", DoubleValue (10));
  v->SetAttribute ("Max", DoubleValue (20));

  Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);

  Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);

  EventId id = Simulator::Schedule (Seconds (30.0), &CancelledEvent);
  Simulator::Cancel (id);

  Simulator::Run ();

  Simulator::Destroy ();
}
```


Simulator example (in Python)

```
# Python version of sample-simulator.cc

import ns.core
```

```
def main(dummy_argv):

    model = MyModel()
    v = ns.core.UniformRandomVariable()
    v.SetAttribute("Min", ns.core.DoubleValue (10))
    v.SetAttribute("Max", ns.core.DoubleValue (20))

    ns.core.Simulator.Schedule(ns.core.Seconds(10.0), ExampleFunction, model)

    ns.core.Simulator.Schedule(ns.core.Seconds(v.GetValue()), RandomFunction, model)

    id = ns.core.Simulator.Schedule(ns.core.Seconds(30.0), CancelledEvent)
    ns.core.Simulator.Cancel(id)

    ns.core.Simulator.Run()

    ns.core.Simulator.Destroy()

if __name__ == '__main__':
    import sys
    main(sys.argv)
```


Simulation program flow



Command-line arguments

- Add CommandLine to your program if you want command-line argument parsing

```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);
```

- Passing --PrintHelp to programs will display command line options, if CommandLine is enabled

```
./waf --run "sample-simulator --PrintHelp"
```

```
--PrintHelp: Print this help message.
--PrintGroups: Print the list of groups.
--PrintTypeIds: Print all TypeIds.
--PrintGroup=[group]: Print all TypeIds of group.
--PrintAttributes=[typeid]: Print all attributes of typeid.
--PrintGlobals: Print the list of globals.
```


Time in ns-3

- Time is stored as a large integer in ns-3
  - Minimize floating point discrepancies across platforms
- Special Time classes are provided to manipulate time (such as standard operators)
- Default time resolution is nanoseconds, but can be set to other resolutions
  - Note: Changing resolution is not well used/tested
- Time objects can be set by floating-point values and can export floating-point values

```
double timeDouble = t.GetSeconds();
```

  - Best practice is to avoid floating point conversions where possible

Events in ns-3

- Events are just function calls that execute at a simulated time
  - i.e. callbacks
  - this is another difference compared to other simulators, which often use special "event handlers" in each model
- Events have IDs to allow them to be cancelled or to test their status

Simulator and Schedulers

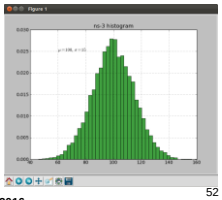
- The Simulator class holds a scheduler, and provides the API to schedule events, start, stop, and cleanup memory
- Several scheduler data structures (calendar, heap, list, map) are possible
- "RealTime" simulation implementation aligns the simulation time to wall-clock time
  - two policies (hard and soft limit) available when the simulation and real time diverge

Random Variables

from src/core/examples/sample-rng-plot.py

- Currently implemented distributions
  - Uniform: values uniformly distributed in an interval
  - Constant: value is always the same (not really random)
  - Sequential: return a sequential list of predefined values
  - Exponential: exponential distribution (poisson process)
  - Normal (gaussian), Log-Normal, Pareto, Weibull, triangular

```
# Demonstrate use of ns-3 as a random number generator integrated with
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial

import numpy as np
import matplotlib.pyplot as plt
import ns.core

# mu, var = 100, 225
rng = ns.core.NormalVariable(100.0, 225.0)
x = [rng.GetValue() for t in range(10000)]

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.title('ns-3 histogram')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



Random variables and independent replications

- Many simulation uses involve running a number of *independent replications* of the same scenario
- In ns-3, this is typically performed by incrementing the simulation *run number*
  - *not by changing seeds*

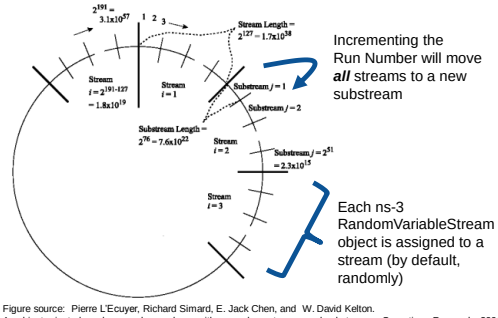
ns-3 random number generator

- Uses the MRG32k3a generator from Pierre L'Ecuyer
  - http://www.iro.umontreal.ca/~lecuyer/myftp/papers/streams00.pdf
  - Period of PRNG is 3.1x10^57
- Partitions a pseudo-random number generator into uncorrelated *streams* and *substreams*
  - Each RandomVariableStream gets its own stream
  - This stream partitioned into substreams

Key Terminology

- **Seed:** A set of values that generates an entirely new PRNG sequence
- **Stream:** The PRNG sequence is divided into non-overlapping intervals called streams
- **Run Number (substream):** Each stream is further divided to substreams, indexed by a variable called the run number.

Streams and Substreams



Incrementing the Run Number will move ***all*** streams to a new substream

Each ns-3 RandomVariableStream object is assigned to a stream (by default, randomly)

Figure source: Pierre L'Ecuyer, Richard Simard, E. Jack Chen, and W. David Kelton. An object-oriented random number package with many long streams and substreams. Operations Research, 2001.

Run number vs. seed

- If you increment the seed of the PRNG, the streams of random variable objects across different runs are not guaranteed to be uncorrelated
- If you fix the seed, but increment the run number, you will get uncorrelated streams

Setting the stream number

- The ns-3 implementation provides access to 2^64 streams
- 2^63 are placed in a pool for automatic assignment, and 2^63 are reserved for fixed assignment

```
<-------------------------------------------------------------------------->
^                                ^^                                      ^
|                                ||                                      |
stream 0           stream (2^63 - 1)  stream 2^63        stream (2^64 - 1)
<- automatically assigned ---------><- assigned by user ----------------->
```

- Users may optionally assign a stream number index to a random variable using the `SetStream ()` method.
  - This allows better control over selected random variables
  - Many helpers have AssignStreams () methods to do this across many such random variables

Putting it together

- Example of scheduled event

```
static void
RandomFunction (void)
{
  std::cout << "RandomFunction received event at "
            << Simulator::Now ().GetSeconds () << "s" << std::endl;
}
```

```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);

  MyModel model;
  Ptr<UniformRandomVariable> v = CreateObject<UniformRandomVariable> ();
  v->SetAttribute ("Min", DoubleValue (10));
  v->SetAttribute ("Max", DoubleValue (20));

  Simulator::Schedule (Seconds (10.0), &ExampleFunction, &model);

  Simulator::Schedule (Seconds (v->GetValue ()), &RandomFunction);
```

Demo real-time, command-line, random variables...

ns-3 Training

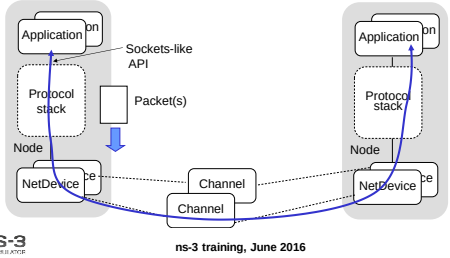
Node, Stacks, and Devices
ns-3 training, June 2016

Example walkthrough

- This section progressively builds up a simple ns-3 example, explaining concepts along the way
- Files for these programs are available on the ns-3 wiki

Example program

- `wns3-version1.cc`
  - Link found on wiki page
  - Place program in scratch/ folder



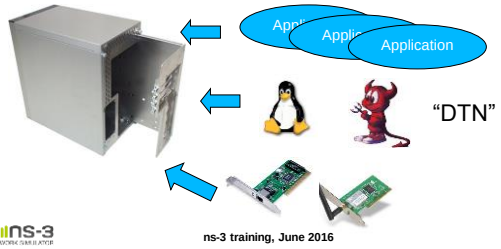
Fundamentals

Key objects in the simulator are Nodes, Packets, and Channels

Nodes contain Applications, “stacks”, and NetDevices

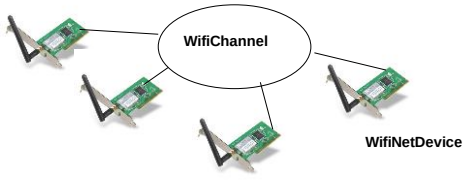
Node basics

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



- ns-3 Spectrum models relax this assumption

Nodes are architected for multiple interfaces

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

- Devices
  - WiFi, WiMAX, CSMA, Point-to-point, ...
- Error models and queues
- Applications
  - echo servers, traffic generator
- Mobility models
- Packet routing
  - OLSR, AODV, DSR, DSDV, Static, Nix-Vector, Global (link state)

Structure of an ns-3 program

```
int main (int argc, char *argv[])
{

  // Set default attribute values

  // Parse command-line arguments

  // Configure the topology; nodes, channels, devices, mobility

  // Add (Internet) stack to nodes

  // Configure IP addressing and routing

  // Add and configure applications

  // Configure tracing

  // Run simulation
}
```

Helper API

- The ns-3 “helper API” provides a set of classes and methods that make common operations easier than using the low-level API
- Consists of:
  - container objects
  - helper classes
- The helper API is implemented using the low-level API
- Users are encouraged to contribute or propose improvements to the ns-3 helper API

Containers

- Containers are part of the ns-3 “helper API”
- Containers group similar objects, for convenience
  - They are often implemented using C++ std containers
- Container objects also are intended to provide more basic (typical) API

The Helper API (vs. low-level API)

- Is not generic
- Does not try to allow code reuse
- Provides simple 'syntactical sugar' to make simulation scripts look nicer and easier to read for network researchers
- Each function applies a single operation on a "set of same objects”
- A typical operation is "Install()"

Helper Objects

- NodeContainer: vector of Ptr<Node>
- NetDeviceContainer: vector of Ptr<NetDevice>
- InternetStackHelper
- WifiHelper
- MobilityHelper
- OlsrHelper
- ... Each model provides a helper class

Installation onto containers

- Installing models into containers, and handling containers, is a key API theme

```
NodeContainer c;
c.Create (numNodes);
...
mobility.Install (c);
...
internet.Install (c);
...
```


Native IP models

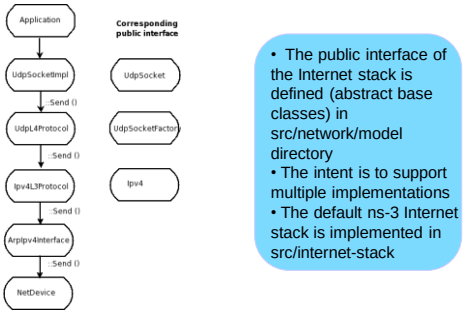
- IPv4 stack with ARP, ICMP, UDP, and TCP
- IPv6 with ND, ICMPv6, IPv6 extension headers, TCP, UDP
- IPv4 routing: RIPv2, static, global, NixVector, OLSR, AODV, DSR, DSDV
- IPv6 routing: RIPng, static

IP address configuration

- An Ipv4 (or Ipv6) address helper can assign addresses to devices in a NetDevice container

```
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
csmaInterfaces = ipv4.Assign (csmaDevices);

...

ipv4.NewNetwork ();  // bumps network to 10.1.2.0
otherCsmaInterfaces = ipv4.Assign (otherCsmaDevices);
```


Internet stack



- The public interface of the Internet stack is defined (abstract base classes) in src/network/model directory
- The intent is to support multiple implementations
- The default ns-3 Internet stack is implemented in src/internet-stack

ns-3 TCP

- Four options exist:
  - native ns-3 TCP
  - TCP simulation cradle (NSC)
  - Direct code execution (Linux/FreeBSD library)
  - Use of virtual machines
- To enable NSC:

```
internetStack.SetNscStack ("liblinux2.6.26.so");
```

Native TCP models

- TCP NewReno is baseline
  - TCP SACK under review for ns-3.26
- TCP congestion control recently refactored, and many TCP variants are under finalization
  - Present: BIC, Highspeed, Hybla, Illinois, Scalable, Vegas, Veno, Westwood, YeAH
  - Pending: CUBIC, H-TCP, SACK
- MP-TCP is under development from past summer project (for ns-3.27?)

ns-3 simulation cradle

- Port by Florian Westphal of Sam Jansen's Ph.D. work

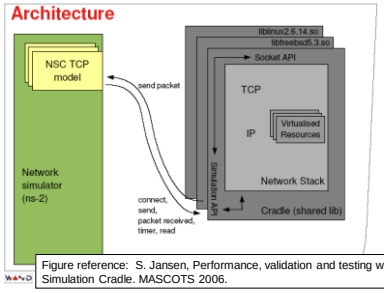


Figure reference: S. Jansen, Performance, validation and testing with the Network Simulation Cradle. MASCOTS 2006.

ns-3 simulation cradle

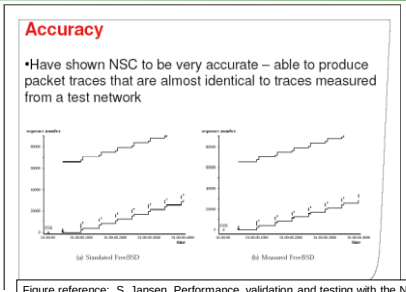


Figure reference: S. Jansen, Performance, validation and testing with the Network Simulation Cradle. MASCOTS 2006.

For ns-3:
- Linux 2.6.18
- Linux 2.6.26
- Linux 2.6.28

Others:
- FreeBSD 5
- lwip 1.3
- OpenBSD 3

Other simulators:
- ns-2
- OmNET++

Review of sample program (cont.)

```
ApplicationContainer apps;
OnOffHelper onoff ("ns3::UdpSocketFactory",
                   InetSocketAddress ("10.1.2.2", 1025));
onoff.SetAttribute ("OnTime", StringValue ("Constant:1.0"));
onoff.SetAttribute ("OffTime", StringValue ("Constant:0.0"));
apps = onoff.Install (csmaNodes.Get (0));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (4.0));

PacketSinkHelper sink ("ns3::UdpSocketFactory",
       InetSocketAddress ("10.1.2.2", 1025));
apps = sink.Install (wifiNodes.Get (1));
apps.Start (Seconds (0.0));
apps.Stop (Seconds (4.0));
```

Traffic generator

Traffic receiver

Applications and sockets

- In general, applications in ns-3 derive from the ns3::Application base class
  - A list of applications is stored in the ns3::Node
  - Applications are like processes
- Applications make use of a sockets-like API
  - Application::Start () may call ns3::Socket::SendMsg() at a lower layer

Sockets API

Plain C sockets

```
int sk;
sk = socket(PF_INET, SOCK_DGRAM, 0);
```

```
struct sockaddr_in src;
inet_pton(AF_INET,"0.0.0.0",&src.sin_ad
    dr);
src.sin_port = htons(80);
bind(sk, (struct sockaddr *) &src,
    sizeof(src));
```

```
struct sockaddr_in dest;
inet_pton(AF_INET,"10.0.0.1",&dest.sin_
    addr);
dest.sin_port = htons(80);
sendto(sk, "hello", 6, 0, (struct
    sockaddr *) &dest, sizeof(dest));
```

```
char buf[6];
recv(sk, buf, 6, 0);
}
```

ns-3 sockets

```
Ptr<Socket> sk =
udpFactory->CreateSocket ();
```

```
sk->Bind (InetSocketAddress (80));
```

```
sk->SendTo (InetSocketAddress (Ipv4Address
    ("10.0.0.1"), 80), Create<Packet>
    ("hello", 6));
```

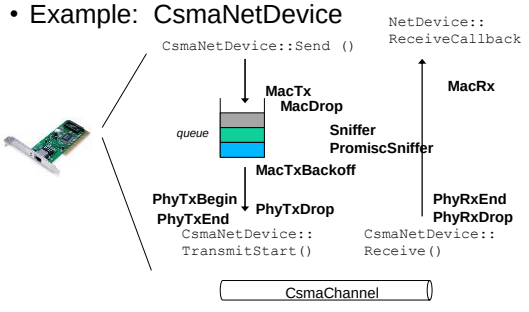
```
sk->SetReceiveCallback (MakeCallback
    (MySocketReceive));
•  [...] (Simulator::Run ())

void MySocketReceive (Ptr<Socket> sk,
    Ptr<Packet> packet)
{
...
}
```

New in ns-3.25: traffic control

- Patterned after Linux traffic control, allows insertion of software-based priority queues between the IP layer and device layer
  - pfifo_fast, RED, Adaptive RED, CoDel
  - planned for ns-3.26: FQ-CoDel, PIE, Byte Queue Limits (BQL)

NetDevice trace hooks

- Example: CsmaNetDevice



Nodes, Mobility, and Position

- ALL nodes have to be created before simulation starts
- Position Allocators setup initial position of nodes
  - List, Grid, Random position…
- Mobility models specify how nodes will move
  - Constant position, constant velocity/acceleration, waypoint…
  - Trace-file based from mobility tools such as SUMO, BonnMotion (using NS2 format)
  - Routes Mobility using Google API (*)

(*) Presented in WNS3 - 2015

Position Allocation Examples

- List

```
MobilityHelper mobility;
// place two nodes at specific positions (100,0) and (0,100)
Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
positionAlloc->Add (Vector (100, 0, 0));
positionAlloc->Add (Vector (0, 100, 0));
mobility.SetPositionAllocator(positionAlloc);
```

- Grid Position

```
MobilityHelper mobility;
// setup the grid itself: nodes are laid out started from (-100,-100) with 20 per row, the x
// interval between each object is 5 meters and the y interval between each object is 20 meters
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (-100.0),
                               "MinY", DoubleValue (-100.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (20.0),
                               "GridWidth", UintegerValue (20),
                               "LayoutType", StringValue ("RowFirst"));
```

- Random Rectangle Position

```
// place nodes uniformly on a straight line from (0, 1000)
MobilityHelper mobility;
Ptr<RandomRectanglePositionAllocator> positionAloc = CreateObject<RandomRectanglePositionAllocator>();
positionAloc->SetAttribute("X", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=100.0]"));
positionAloc->SetAttribute("Y", StringValue("ns3::ConstantRandomVariable[Constant=50.0]"));
mobility.SetPositionAllocator(positionAloc);
```


Mobility Model Example

- Constant Position

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
```

- Constant Speed

```
MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantVelocityMobilityModel");
mobility.Install (nodes);
Ptr<UniformRandomVariable> rvar = CreateObject<UniformRandomVariable>();
for (NodeContainer::Iterator i = nodes.Begin (); i != nodes.End (); ++i){
  Ptr<Node> node = (*i);
  double speed = rvar->GetValue(15, 25);
  node->GetObject<ConstantVelocityMobilityModel>()->SetVelocity(Vector(speed,0,0));
}
```

- Trace-file based

```
std::string traceFile = "mobility_trace.txt";
// Create Ns2MobilityHelper with the specified trace log file as parameter
Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);
ns2.Install (); // configure movements for each node, while reading trace file
```


Interesting ns-3 extensions

- ns-3-highway-mobility (https://code.google.com/p/ns-3-highway-mobility/)
  - Implement IDM and MOBIL change lane, highway class, traffic-lights.
  - Based on ns-3.8
  - No longer maintained
- Virtual Traffic Lights (PROMELA) (https://dsn.tm.kit.edu/misc_3434.php)
  - Manhattan IDM mobility model
  - NLOS propagation loss models
  - (Virtual) Traffic Light applications

Propagation Models

- Propagation Loss
  - ITUR1411, LogDistance, ThreeLogDistance, Range, TwoRayGround, Friis
  - Nakagami, Jakes
  - Obstacle model (*)

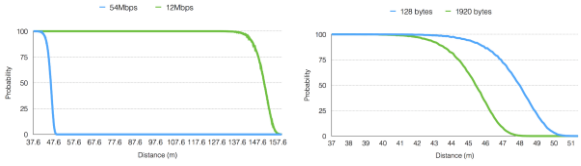
(*) Presented in WNS3 2015

- Propagation Delay
  - Constant Speed
  - Random

- Be careful when using `YansWifiChannelHelper::Default()` the LogDistance propagation model is added. Calling `AddPropagationLoss()` again will add a *second* propagation loss model.

Communication Range

- Depends on many factors
  - Propagation loss model and PHY configuration
  - Frame size (big vs small)
  - Transmission mode (6Mbps vs 54 Mbps)



Example program iterations

- Walk through four additional revisions of the example program
  - `wns3-version2.cc`
  - `wns3-version3.cc`
  - `wns3-version4.cc`

ns-3 Training: Packets

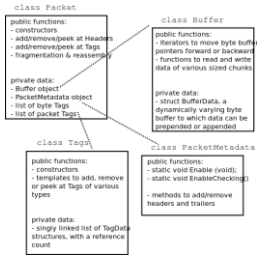
ns-3 Annual meeting
June 2016

ns-3 Packet

- Packet is an advanced data structure with the following capabilities
  - Supports fragmentation and reassembly
  - Supports real or virtual application data
  - Extensible
  - Serializable (for emulation)
  - Supports pretty-printing
  - Efficient (copy-on-write semantics)

ns-3 Packet structure

- Analogous to an mbuf/skbuff



Copy-on-write

- Copy data bytes only as needed

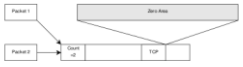


Figure 3.8: The TCP and the IP stacks hold references to a shared buffer.

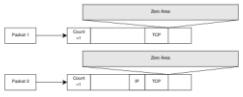


Figure 3.9: The IP stack inserts the IP header, triggers an un-share operation, completes the insertion.

Figure source: Mathieu Lacage's Ph.D. thesis

Headers and trailers

- Most operations on packet involve adding and removing an ns3::Header
- class ns3::Header must implement four methods:

```
Serialize()
Deserialize()
GetSerializedSize()
Print()
```


Headers and trailers (cont.)

- Headers are serialized into the packet byte buffer with Packet::AddHeader() and removed with Packet::RemoveHeader()
- Headers can also be 'Peeked' without removal

```
Ptr<Packet> pkt = Create<Packet> ();
UdpHeader hdr; // Note:  not heap allocated
pkt->AddHeader (hdr);
Ipv4Header iphdr;
pkt->AddHeader (iphdr);
```

Packet tags

- Packet tag objects allow packets to carry around simulator-specific metadata
  - Such as a "unique ID" for packets or
- Tags may associate with byte ranges of data, or with the whole packet
  - Distinction is important when packets are fragmented and reassembled
- Tags presently are not preserved across serialization boundaries (e.g. MPI)

PacketTag vs. ByteTag

- Two tag types are available: PacketTag and ByteTag
  - ByteTags run with bytes
  - PacketTags run with packets
- When Packet is fragmented, both copies of Packet get copies of PacketTags
- When two Packets are merged, only the PacketTags of the first are preserved
- PacketTags may be removed individually; ByteTags may be removed all at once

Tag example

- Here is a simple example illustrating the use of tags from the code in src/internet/model/udp-socket-impl.cc:

```
Ptr<Packet> p;  // pointer to a pre-existing packet
SocketIpTtlTag tag
tag.SetTtl (m_ipMulticastTtl); // Convey the TTL from
UDP layer to IP layer
p->AddPacketTag (tag);
```

- This tag is read at the IP layer, then stripped (src/internet/model/ipv4-l3-protocol.cc):

```
uint8_t ttl = m_defaultTtl;
SocketIpTtlTag tag;
bool found = packet->RemovePacketTag (tag);
if (found)
  {
    ttl = tag.GetTtl ();
  }
```

Packet metadata

- Packets may optionally carry metadata
  - record every operation on a packet's buffer
  - implementation of Packet::Print for pretty-printing of the packet
  - sanity check that when a Header is removed, the Header was actually present to begin with
- Not enabled by default, for performance reasons
- To enable, insert one or both statements:

```
Packet::EnablePrinting ();
Packet::EnableChecking ();
```


Ptr<Packet>

- Packets are reference counted objects that support the smart pointer class `Ptr`
- Use a templated "Create" method instead of CreateObject for ns3::Objects
- Typical creation:
  - `Ptr<Packet> pkt = Create<Packet> ();`
- In model code, Packet pointers may be const or non-const; often Packet::Copy() is used to obtain non-const from const
  - `Ptr<const Packet> cpkt = ...;`
  - `Ptr<Packet> p = cpkt->Copy ();`

ns-3 Training

ns-3 Objects

ns-3 training, June 2016

Object metadata system

- ns-3 is, at heart, a C++ object system
- ns-3 objects that inherit from base class ns3::Object get several additional features
  - smart-pointer memory management (Class Ptr)
  - dynamic run-time object aggregation
  - an attribute system

Smart pointers

- Smart pointers in ns-3 use reference counting to improve memory management
- The class `ns3::Ptr` is semantically similar to a traditional pointer, but the object pointed to will be deleted when all references to the pointer are gone
- ns-3 heap-allocated objects should use the templated Create<>() or CreateObject<> () methods

Examples

```
Ptr<WifiNetDevice> dev =
   CreateObject<WifiNetDevice> ();

Ptr<Packet> pkt = Create<Packet> ();
```

(instead of `Packet* = new Packet;`)

why Create<> vs CreateObject<>?

- two different base classes; generally use CreateObject<>(), but Create<> for Packet

Dynamic run-time object aggregation

- This feature is similar to "Component Object Model (COM)"-- allows interfaces (objects) to be aggregated at run-time instead of at compile time
- Useful for binding dissimilar objects together without adding pointers to each other in the classes

Usage

- ns-3 Node protocol stacks are added via aggregation
  - The IP stack can be found from a Node pointer without class Node knowing about it
- Energy models are typically aggregated to nodes
- To find interfaces, use GetObject<>(); e.g.

```
Ptr<Ipv4> ipv4 = m_node->GetObject<Ipv4> ();
```


Attributes and default values

```
// disable fragmentation for frames below 2200 bytes
Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue ("22
00"));
// turn off RTS/CTS for frames below 2200 bytes
Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue ("2200"));
// Fix non-unicast data rate to be the same as that of unicast
Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                    StringValue (phyMode));

NodeContainer c;
c.Create (numNodes);

// The below set of helpers will help us to put together the wifi NICs we want
WifiHelper wifi;
if (verbose)
  {
    wifi.EnableLogComponents ();  // Turn on all Wifi logging
  }

YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
// set it to zero; otherwise, gain will be added
wifiPhy.Set ("RxGain", DoubleValue (-10) );
// ns-3 supports RadioTap and Prism tracing extensions for 802.11b
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
```

ns-3 attribute system

Problem: Researchers want to identify all of the values affecting the results of their simulations
  - and configure them easily

ns-3 solution: Each ns-3 object has a set of attributes:
  - A name, help text
  - A type
  - An initial value
- Control all simulation parameters for static objects
- Dump and read them all in configuration files
- Visualize them in a GUI
- Makes it easy to verify the parameters of a simulation

Short digression: Object metadata system

- ns-3 is, at heart, a C++ object system
- ns-3 objects that inherit from base class ns3::Object get several additional features
  - dynamic run-time object aggregation
  - an attribute system 
  - smart-pointer memory management (Class Ptr)

We focus here on the attribute system

Use cases for attributes

- An Attribute represents a value in our system
- An Attribute can be connected to an underlying variable or function
  - e.g. TcpSocket::m_cwnd;
  - or a trace source

Use cases for attributes (cont.)

- What would users like to do?
  - Know what are all the attributes that affect the simulation at run time
  - Set a default initial value for a variable
  - Set or get the current value of a variable
  - Initialize the value of a variable when a constructor is called
- The attribute system is a unified way of handling these functions

How to handle attributes

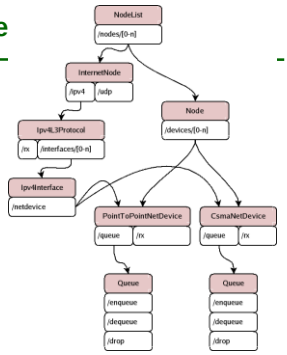
- The traditional C++ way:
  - export attributes as part of a class's public API
  - walk pointer chains (and iterators, when needed) to find what you need
  - use static variables for defaults
- The attribute system provides a more convenient API to the user to do these things

Navigating the attributes

- Attributes are exported into a string-based namespace, with filesystem-like paths
  - namespace supports regular expressions
- Attributes also can be used without the paths
  - e.g. `"ns3::WifiPhy::TxGain"`
- A Config class allows users to manipulate the attributes

Attribute namespace

- strings are used to describe paths through the namespace




```
Config::Set ("/NodeList/1/$ns3::Ns3NscStack<linux2.6.26>/net.ipv4.tcp_sack", StringValue ("0"));
```


Navigating the attributes using paths

- Examples:
  - Nodes with NodeIds 1, 3, 4, 5, 8, 9, 10, 11:
    `"/NodeList/[3-5]|[8-11]|1"`
  - UdpL4Protocol object instance aggregated to matching nodes:
    `"/$ns3::UdpL4Protocol"`

What users will do

- e.g.: Set a default initial value for a variable

```
Config::Set ("ns3::YansWifiPhy::TxGain",
  DoubleValue (1.0));
```

- Syntax also supports string values:

```
Config::Set ("YansWifiPhy::TxGain",
  StringValue ("1.0"));
```

Value

Attribute

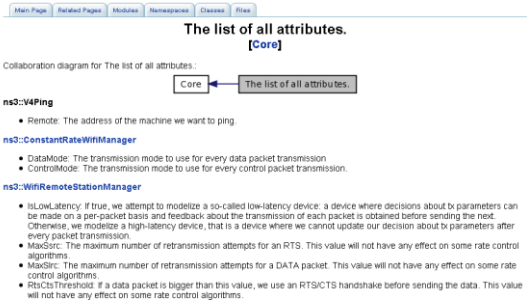
Fine-grained attribute handling

- Set or get the current value of a variable
  - Here, one needs the path in the namespace to the right instance of the object

```
Config::SetAttribute("/NodeList/5/DeviceList/3/$n
  s3::WifiNetDevice/Phy/$ns3::YansWifiPhy/TxGain",
  DoubleValue(1.0));
DoubleValue d; nodePtr->GetAttribute (
  "/NodeList/5/NetDevice/3/$ns3::WifiNetDevice/Phy
  /$ns3::YansWifiPhy/TxGain", d);
```

- Users can get Ptrs to instances also, and Ptrs to trace sources, in the same way

Attribute documentation



Main Page | Related Pages | Modules | Namespaces | Classes | Files

The list of all attributes.
[Core]

Collaboration diagram for The list of all attributes.:

Core ← The list of all attributes.

ns3::V4Ping

- Remote: The address of the machine we want to ping.

ns3::ConstantRateWifiManager

- DataMode: The transmission mode to use for every data packet transmission
- ControlMode: The transmission mode to use for every control packet transmission.

ns3::WifiRemoteStationManager

- IsLowLatency: If true, we attempt to modelize a so-called low-latency device: a device where decisions about tx parameters can be made on a per-packet basis and feedback about the transmission of each packet is obtained before sending the next. Otherwise, we modelize a high-latency device, that is a device where we cannot update our decision about tx parameters after every packet transmission.
- MaxSsrc: The maximum number of retransmission attempts for an RTS. This value will not have any effect on some rate control algorithms.
- MaxSlrc: The maximum number of retransmission attempts for a DATA packet. This value will not have any effect on some rate control algorithms.
- RtsCtsThreshold: If a data packet is bigger than this value, we use an RTS/CTS handshake before sending the data. This value will not have any effect on some rate control algorithms.

Options to manipulate attributes

- Individual object attributes often derive from default values
  - Setting the default value will affect all subsequently created objects
  - Ability to configure attributes on a per-object basis
- Set the default value of an attribute from the command-line:

```
CommandLine cmd;
cmd.Parse (argc, argv);
```

- Set the default value of an attribute with NS_ATTRIBUTE_DEFAULT
- Set the default value of an attribute in C++:

```
Config::SetDefault ("ns3::Ipv4L3Protocol::CalcChecksum",
BooleanValue (true));
```

- Set an attribute directly on a specic object:

```
Ptr<CsmaChannel> csmaChannel = ...;
csmaChannel->SetAttribute ("DataRate",
StringValue ("5Mbps"));
```

Summary on ns-3 objects

- ns-3 objects that inherit from base class ns3::Object get several additional features
  1. smart-pointer memory management (Class Ptr)
  2. dynamic run-time object aggregation
  3. an attribute system
- These types of objects are allocated on the heap, not on the stack

ns-3 Training

Debugging support

ns-3 training, June 2016

Writing and debugging new programs

- Choosing between Python and C++
- Reading existing code
- Understanding and controlling logging code
- Error conditions
- Running programs through a debugger

Python bindings

- ns-3 uses the 'pybindgen' tool to generate Python bindings for the underlying C++ libraries
- Existing bindings are typically found in the bindings/ directory of a module
- Some methods are not provided in Python (e.g. hooking trace sources)
- Generating new bindings requires a toolchain documented on the ns-3 web site

Debugging support

- Assertions: NS_ASSERT (expression);
  - Aborts the program if expression evaluates to false
  - Includes source file name and line number
- Unconditional Breakpoints: NS_BREAKPOINT ();
  - Forces an unconditional breakpoint, compiled in
- Debug Logging (not to be confused with tracing!)
  - Purpose
    - Used to trace code execution logic
    - For debugging, not to extract results!
  - Properties
    - NS_LOG* macros work with C++ IO streams
    - E.g.: NS_LOG_UNCOND ("I have received " << p->GetSize () << " bytes");
    - NS_LOG macros evaluate to nothing in optimized builds
    - When debugging is done, logging does not get in the way of execution performance

Debugging support (cont.)

- Logging levels:
  - NS_LOG_ERROR (...): serious error messages only
  - NS_LOG_WARN (...): warning messages
  - NS_LOG_DEBUG (...): rare ad-hoc debug messages
  - NS_LOG_INFO (...): informational messages (eg. banners)
  - NS_LOG_FUNCTION (...):function tracing
  - NS_LOG_PARAM (...): parameters to functions
  - NS_LOG_LOGIC (...): control flow tracing within functions
- Logging "components"
  - Logging messages organized by components
  - Usually one component is one .cc source file
  - NS_LOG_COMPONENT_DEFINE ("OlsrAgent");
- Displaying log messages. Two ways:
  - Programatically:
    - LogComponentEnable("OlsrAgent", LOG_LEVEL_ALL);
  - From the environment:
    - NS_LOG="OlsrAgent" ./my-program

Running C++ programs through gdb

- The gdb debugger can be used directly on binaries in the build directory
- An easier way is to use a waf shortcut

```
./waf --command-template="gdb %s" --run <program-name>
```


Running C++ programs through valgrind

- valgrind memcheck can be used directly on binaries in the build directory
- An easier way is to use a waf shortcut

```
./waf --command-template="valgrind %s" --run <program-name>
```

- Note: disable GTK at configure time when running valgrind (to suppress spurious reports)
- `./waf configure --disable-gtk --enable-tests ...`

Testing

- ns-3 models need tests verifiable by others (often overlooked)
  - Onus is on the simulation project to validate and document results
  - Onus is also on the researcher to verify results
- ns-3 strategies:
  - regression tests
    - Aim for ***event-based*** rather than ***trace-based***
  - unit tests for verification
  - validation of models on testbeds where possible
  - reuse of code

Test framework

- ns-3-dev is checked nightly on multiple platforms
  - Linux gcc-4.x, i386 and x86_64, OS X, FreeBSD clang, and Cygwin (occasionally)
- `./test.py` will run regression tests

Walk through test code, test terminology (suite, case), and examples of how tests are run

Improving performance

- Debug vs optimized builds
  - ./waf -d debug configure
  - ./waf -d debug optimized
- Build ns-3 with static libraries
  - ./waf --enable-static
- Use different compilers (icc)
  - has been done in past, not regularly tested

ns-3 Training
Visualization

ns-3 Annual Meeting
June 2016

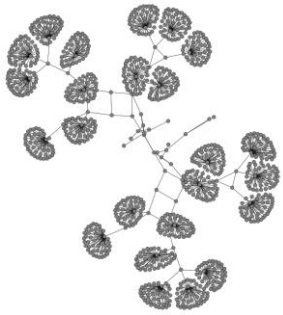
Overview

- No preferred visualizer for ns-3
- Several tools have been developed over the years, with some scope limitations
  - Pyviz
  - FlowMonitor (statistics with Pyviz linkage)
  - NetAnim (George Riley and John Abraham)

PyViz overview

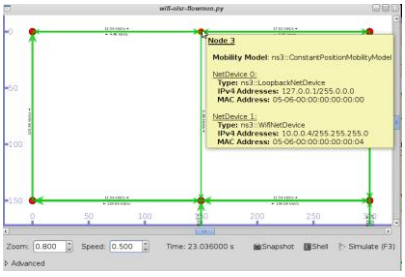
- Developed by Gustavo Carneiro
- Live simulation visualizer (no trace files)
- Useful for debugging
  - mobility model behavior
  - where are packets being dropped?
- Built-in interactive Python console to debug the state of running objects
- Works with Python and C++ programs

Pyviz screenshot (Graphviz layout)



Pyviz and FlowMonitor

- src/flow-monitor/examples/wifi-olsr-flowmon.py



Enabling PyViz in your simulations

- Make sure PyViz is enabled in the build

```
Sqlite stats data output       : not enabled (library 'sqlite3' not found)
Tap Bridge                     : enabled
PyViz visualizer               : enabled
Use sudo to set suid bit       : not enabled (option --enable-sudo not selected)
```

- If program supports CommandLine parsing, pass the option

```
--SimulatorImplementationType=
ns3::VisualSimulatorImpl
```

- Alternatively, pass the "--vis" option

FlowMonitor

- Network monitoring framework found in `src/flow-monitor/`
- Goals:
  - detect all flows passing through network
  - stores metrics for analysis such as bitrates, duration, delays, packet sizes, packet loss ratios

G. Carneiro, P. Fortuna, M. Ricardo, "FlowMonitor-- a network monitoring framework for the Network Simulator ns-3," Proceedings of NSTools 2009.

FlowMonitor architecture

- Basic classes
  - FlowMonitor
  - FlowProbe
  - FlowClassifier
  - FlowMonitorHelper
- IPv4 and IPv6

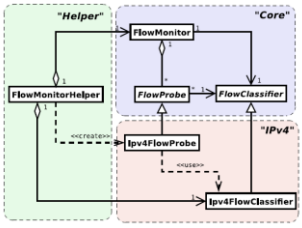
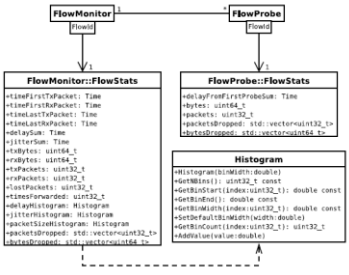


Figure credit: G. Carneiro, P. Fortuna, M. Ricardo, "FlowMonitor-- a network monitoring framework for the Network Simulator ns-3," Proceedings of NSTools 2009.

FlowMonitor statistics

- Statistics gathered



FlowMonitor configuration

- `example/wireless/wifi-hidden-terminal.cc`

```
// 8. Install FlowMonitor on all nodes
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();

// 9. Run simulation for 10 seconds
Simulator::Stop (Seconds (10));
Simulator::Run ();

// 10. Print per flow statistics
monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
  {
    // first 2 FlowIds are for ECHO apps, we don't want to display them
    if (i->first > 2)
      {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
        std::cout << "Flow " << i->first - 2 << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";
        std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
        std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
        std::cout << "  Throughput: " << i->second.rxBytes * 8.0 / 10.0 / 1024 / 1024  << " Mbps\n";
      }
  }
```

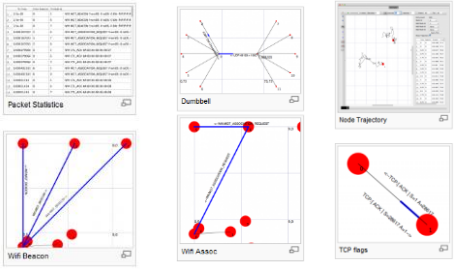
FlowMonitor output

- This program exports statistics to stdout
- Other examples integrate with PyViz

```
Hidden station experiment with RTS/CTS disabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
  Tx Bytes:   3847500
  Rx Bytes:   316464
  Throughput: 0.241443 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Bytes:   3848412
  Rx Bytes:   336756
  Throughput: 0.256924 Mbps
---------------------------------------------------------
Hidden station experiment with RTS/CTS enabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
  Tx Bytes:   3847500
  Rx Bytes:   306660
  Throughput: 0.233963 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Bytes:   3848412
  Rx Bytes:   274740
  Throughput: 0.20961 Mbps
```


NetAnim

- "NetAnim" by George Riley and John Abraham



NetAnim key features

- Animate packets over wired-links and wireless-links
  - limited support for LTE traces
- Packet timeline with regex filter on packet meta-data.
- Node position statistics with node trajectory plotting (path of a mobile node).
- Print brief packet-meta data on packets

ns-3 Training

Emulation support

ns-3 training, June 2016

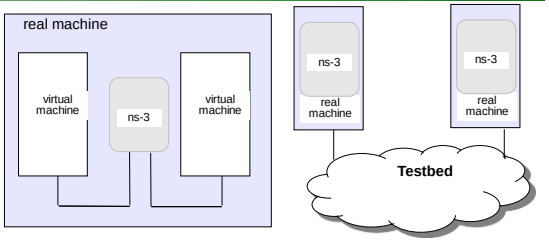
Outline

- Emulation modes
  - Tap Bridge
  - FdNetDevice
- Direct Code Execution (DCE)
  - Applications
  - Linux Kernel
  - DCE Cradle

Emulation support

- Support moving between simulation and testbeds or live systems
- A real-time scheduler, and support for two modes of emulation
- Linux is only operating system supported
- Must run simulator in real time
  - `GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3::RealTimeSimulatorImpl"));`
- Must enable checksum calculations across models
  - `GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));`
- Must run as root

ns-3 emulation modes



1) ns-3 interconnects real or virtual machines

2) testbeds interconnect ns-3 stacks

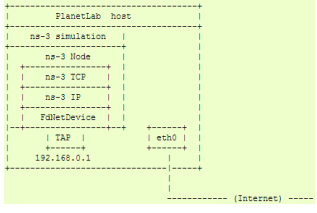
Various hybrids of the above are possible

Example use case: testbeds

- Support for use of Rutgers WINLAB ORBIT radio grid

Example use case: PlanetLab

- The PlanetLabFdNetDeviceHelper creates TAP devices on PlanetLab nodes using specific PlanetLab mechanisms (i.e. the vsys system), and associates the TAP device to a FdNetDevice in ns-3.



Example use case: mininet

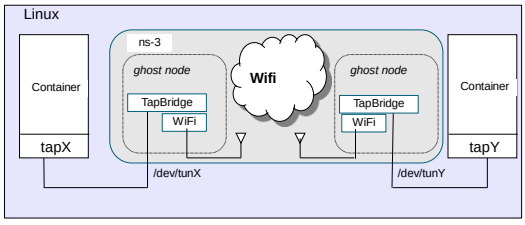
- Mininet is popular in the Software-Defined Networking (SDN) community
- Mininet uses "TapBridge" integration
- https://github.com/mininet/mininet/wiki/Link-modeling-using-ns-3

Emulation Devices

Device models

- File Descriptor Net Device (FdNetDevice)
  - read and write traffic using a file descriptor provided by the user
  - this file descriptor can be associated to a TAP device, to a raw socket, to a user space process generating/consuming traffic, etc.
- Tap Bridge
  - Integrate Tun/Tap devices with ns-3 devices

“TapBridge": netns and ns-3 integration

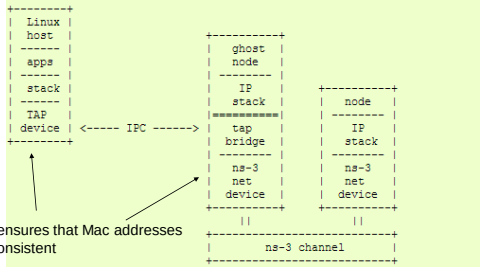


Tap device pushed into namespaces; no bridging needed

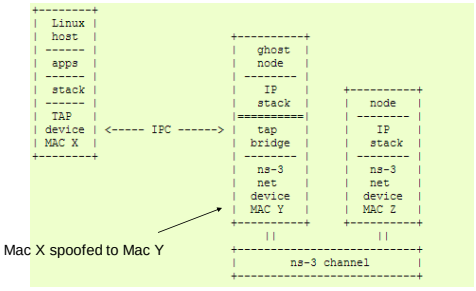
TapBridge modes

- ConfigureLocal (default mode)
  - ns-3 configures the tap device
  - useful for host to ns-3 interaction
- UseLocal
  - user has responsibility for device creation
  - ns-3 informed of device using “DeviceName” attribute
- UseBridge
  - TapDevice connected to existing Linux bridge

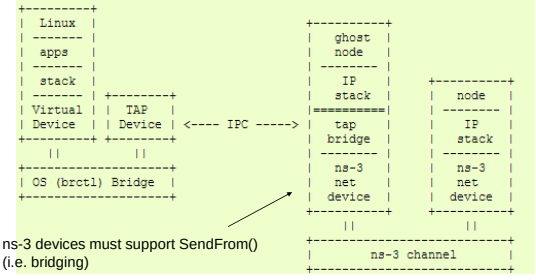
ConfigureLocal



UseLocal



UseBridge

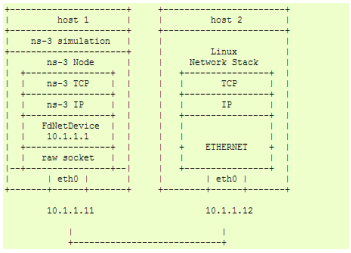


FdNetDevice

- Unified handling of reading/writing from file descriptor
- Three supported helper configurations:
  - EmuFdNetDeviceHelper (to associate the ns-3 device with a physical device in the host machine)
  - TapFdNetDeviceHelper (to associate the ns-3 device with the file descriptor from a tap device in the host machine) (not the same as TapBridge)
  - PlanetLabFdNetDeviceHelper (to automate the creation of tap devices in PlanetLab nodes, enabling ns-3 simulations that can send and receive traffic though the Internet using PlanetLab resource.

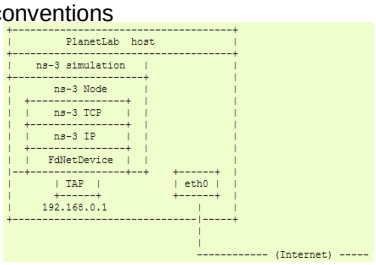
EmuFdNetDeviceHelper

- Device performs MAC spoofing to separate emulation from host traffic



PlanetLabFdNetDeviceHelper

- Special case of TapFdNetDeviceHelper where Tap devices configured according to PlanetLab conventions



ns-3 over host sockets

- Two publications about how to run ns-3 applications over real hosts and sockets
  - "Simulator-agnostic ns-3 Applications", Abraham and Riley, WNS3 2012
  - Gustavo Carneiro, Helder Fontes, Manuel Ricardo, "Fast prototyping of network protocols through ns-3 simulation model reuse", Simulation Modelling Practice and Theory (SIMPAT), vol. 19, pp. 2063–2075, 2011.

Generic Emulation Issues

- Ease of use
  - Configuration management and coherence
  - Information coordination (two sets of state)
    - e.g. IP/MAC address coordination
  - Output data exists in two domains
  - Debugging can be more challenging
- Error-free operation (avoidance of misuse)
  - Synchronization, information sharing, exception handling
    - Checkpoints for execution bring-up
    - Inoperative commands within an execution domain
    - Deal with run-time errors
  - Soft performance degradation (CPU) and time discontinuities

Tracing in NS-3

May, 2014

Walid Younes

Physical and Life Sciences

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Security, LLC, Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Chapter 7: Tracing

Source material:

- Online tutorial: http://www.nsnam.org/docs/release/3.14/tutorial/singlehtml/index.html#tracing
- T. Predojev (2012): http://wikienergy.cttc.es/images/2/2d/Ns-3-tutorial-complete.pdf
- M. Lacage (2009): http://www.nsnam.org/tutorials/ns-3-tutorial-tunis-apr09.pdf
- G. Riley (2008): http://www.wns2.org/docs/wns_tutorial-handout.pdf
- S. Kristiansen (2010): http://www.uio.no/studier/emner/matnat/ifi/INF5090/v11/undervisningsmateriale/INF5090-NS-3-Tutorial-2011-Oslo-slides.pdf

More advanced tracing with NS3

- 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 NS3
  - 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

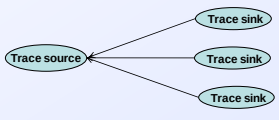
You could use print statements, but I wouldn't recommend it

- You may have to dig deep inside the NS3 core to find the info you want
- More print statements ⇒ need way to enable/disable specific ones
  - Congratulations! You've just re-invented the NS3 logging system!
- You could add logging statements to the core
  - Remember: NS3 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

Ex: a function to integrate functions

```
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 "ns3/object.h"
#include "ns3/uinteger.h"
#include "ns3/traced-value.h"
#include "ns3/trace-source-accessor.h"

#include <iostream>

using namespace ns3;

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

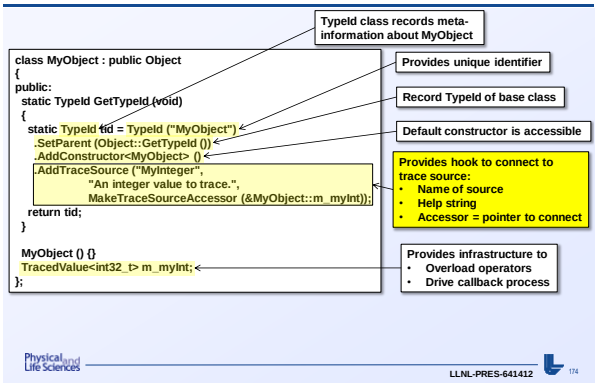
A simple-low level example: the trace source

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 located via path (more on this later)
- provides objects with attributes
- provides objects with trace sources

A simple-low level example: the trace source



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

NS3 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;
}
```

If all goes as planned: should trigger callback

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

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 ns3, by using a config path
  - A string that looks like a file path
  - Represents chain of objects leading to the desired event (or attribute)
- We encountered this in third.cc

```
"/NodeList/7/$ns3::MobilityModel/CourseChange"
```

Provides a path to the "CourseChange" attribute for node index 7

Let's dissect this config path

Dissecting the Config path in third.cc

```
"/NodeList/7/$ns3::MobilityModel/CourseChange"
```

- "NodeList" = predefined namespace in Config, lists all the nodes in the sim
- "NodeList/7" = node 7 (i.e., the eigth node)
- "$" = what follows designates an aggregated object

Let's take a short detour to discuss aggregated objects in NS3

Object aggregation in NS3

- Makes it possible for objects to access each other, and for users to easily access objects in an aggregation
- Avoids need to modify a base class to provide pointers to all possible connected objects
  - Class definitions would bloat uncontrollably
  - Solution: separate functionality belongs to separate classes

```
Node->AggregateObject(mobility);
```

Retrieving an aggregated object:

```
Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
```


Dissecting the Config path in third.cc

```
"/NodeList/7/$ns3::MobilityModel/CourseChange"
```

- "NodeList" = predefined namespace in Config, lists all the nodes in the sim
- "NodeList/7" = node 7 (i.e., the eigth node)
- "$" = what follows designates an aggregated object
- "/NodeList/7/$ns3::MobilityModel" = get the mobility model object aggregated to node 7 in the sim
- "CourseChange" = attribute of MobilityModel we are interested in
- Config will follow the chain of pointers to the attribute you specified

```
Config::Connect("/NodeList/7/$ns3::MobilityModel/CourseChange",MakeCallback(&CourseChange))
```

- Ties the trace source to the trace sink callback function
- Passes the context (= config path) to the callback function

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

- What trace sources are available (besides CourseChange)?
- 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 CourseChange(std::string context,Ptr<const MobilityModel> model)
```

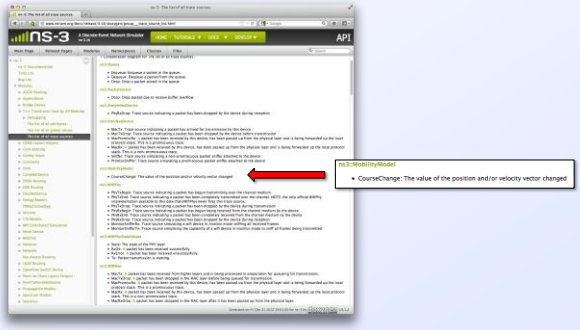
```
Void IntTrace (int32_t oldValue, int32_t newValue)
```

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

What trace sources are available?

- Doxygen has the answer!
  - Trace sources: http://www.nsnam.org/docs/release/3.16/doxygen/group___trace_source_list.html
  - Attributes: http://www.nsnam.org/docs/release/3.16/doxygen/group___attribute_list.html
  - Global values: http://www.nsnam.org/docs/release/3.16/doxygen/group___global_value_list.html

What trace sources are available?



What string do I use to connect?

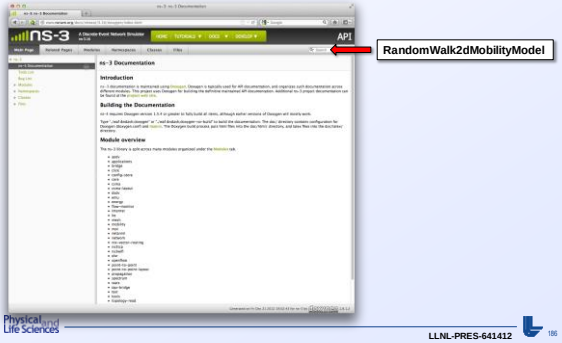
- method 1: look for config path in someone else's code
  - find -name "*.cc" | xargs grep CourseChange | grep Connect

```
⋮
./src/mobility/examples/main-random-walk.cc:  Config::Connect
("/NodeList/*/$ns3::MobilityModel/CourseChange",
⋮
```

  - Config path = "/NodeList/*/$ns3::MobilityModel/CourseChange"
    - you can use regular expressions in the path, so "*" ⇒ any node
    - "/NodeList/[3-5]|8|[0-1]" would match node indices 0,1,3,4,5,8

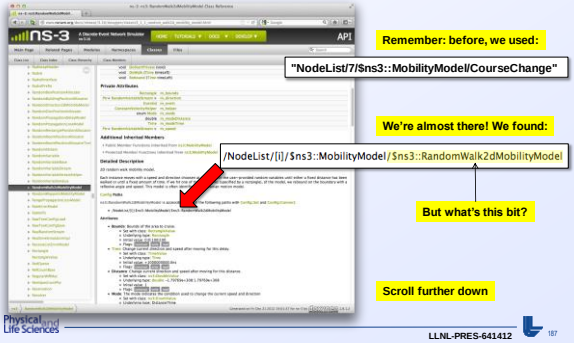
What string do I use to connect?

- **method 2**: Doxygen



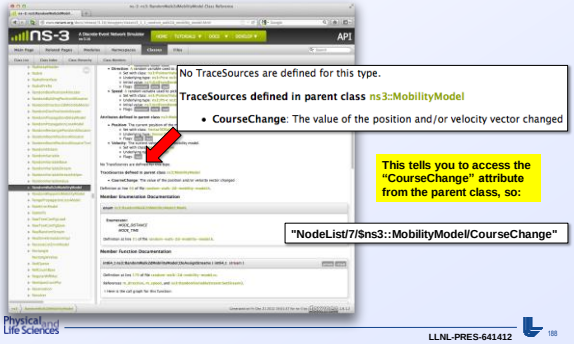
What string do I use to connect?

- **method 2**: Doxygen



What string do I use to connect?

- **method 2**: Doxygen



What are the return value and arguments for the callback function?

- The easy way: copy someone else's answer
  - find -name "*.cc" | xargs grep CourseChange | grep Connect

```
./src/mobility/examples/main-random-walk.cc:  Config::Connect
("/NodeList/*/$ns3::MobilityModel/CourseChange",
```

  - Inside "./src/mobility/examples/main-random-walk.cc":

```
Config::Connect ("/NodeList/*/$ns3::MobilityModel/CourseChange",
                 MakeCallback (&CourseChange));
```

  - And we then find the header of the "CourseChange" function:

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

But what if you can't find what you need in someone else's code?

What return value and arguments for the callback function?

- The somewhat easy way
  - The return value is always "void"
  - To get the list of formal arguments, look in the ".h" file for the model
    - Find the "TracedCallback" declaration
    - For ex,in "src/mobility/model/mobility-model.h"

```
TracedCallback<Ptr<const MobilityModel> > m_courseChangeTrace;
```

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

What about TracedValue?

- Remember we used the callback function

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

- How did we guess the right formal arguments to use?
- Find "TracedCallback" declaration in "src/core/model/traced-value.h"

```
template <typename T> class TracedValue
{
public:
  ...
private:
  T m_v;
  TracedCallback<T,T> m_cb;
};
```

Template class ⇒ you get to choose its type

- In fourth.cc, inside the MyObject class, we declared

```
TracedValue<int32_t> m_myInt;
```

  ⇒ T = int32_t
  ⇒ Two arguments in the callback, both of type T (i.e., int32_t)

A real example

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

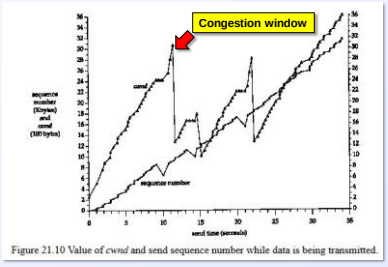


Figure 21.10 Value of *cwnd* and send sequence number while data is being transmitted.

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"

Look for "congestion":

ns3::TcpNewReno

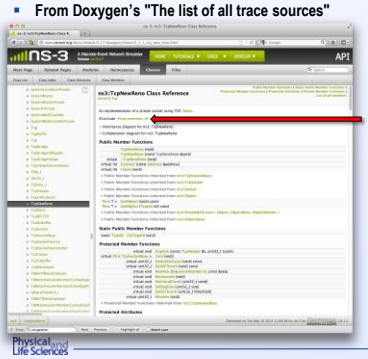
- CongestionWindow: The TCP connection's congestion window

We have our trace source: CongestionWindow

Click on "ns3::TcpNewReno" to access class info

Are there trace sources available?

- From Doxygen's "The list of all trace sources"



- Click on link to header file "tcp-newreno.h"
- Look for keyword "congestion"

Are there trace sources available?

- From Doxygen's "The list of all trace sources"

TracedValue<uint32_t> m_cWnd; //< Congestion window

this looks a lot like our fourth.cc example

we'll need a callback function of the form

void CwndTrace(uint32_t oldValue,uint32_t newValue)

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/ns3tcp/ns3tcp-cwnd-test-suite.cc"
  - The connection between trace and source is done by

```
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow",
MakeCallback (&Ns3TcpCwndTestCase1::CwndChange, this));
```

This is the callback function

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

  - We can also copy code from the function

```
void Ns3TcpCwndTestCase1::DoRun (void)
```

This is how fifth.cc was put together

Avoiding a common mistake in NS3

- NS3 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!

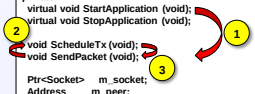
Creating our own application: the MyApp class

```
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

…

1. 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 are errored 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 ("ns3::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

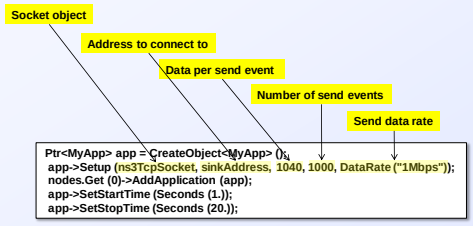
Another way of creating a socket using a socket factory

```
Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (nodes.Get (0), TcpSocketFactory::GetTypeId ());

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

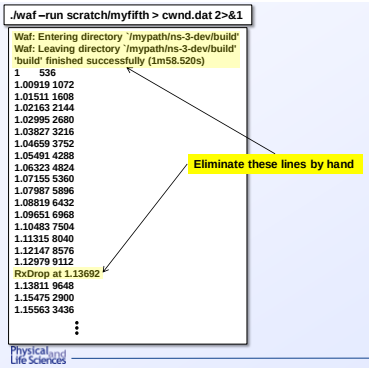
This should look familiar by now

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

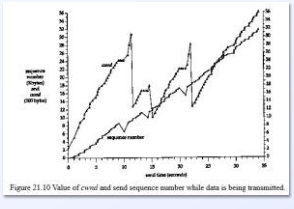


Running fifth.cc: text output

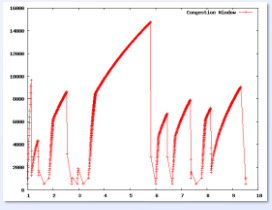


Running fifth.cc: plotting the results

Original by W. Richard Stevens



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");
ns3TcpSocket->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);
}
```

Added to fifth.cc

Formatted output to the pcap file

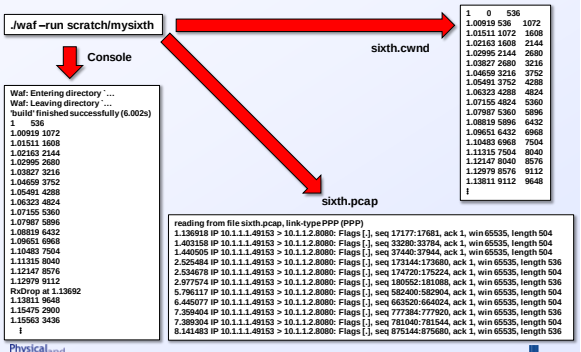
Add lines in the main to create stream:

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));
```

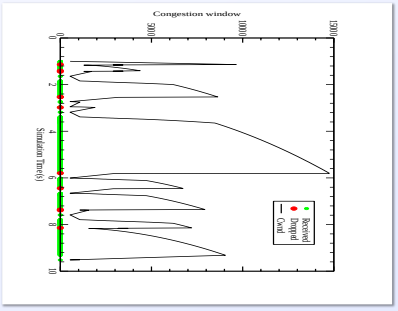
Causes the file argument to be added to the function callback

Running sixth.cc



Just for fun: 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

NS3 uses "mixin" classes to ensure tracing works the same way across all devices or interfaces

- Mixin classes in NS3 (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;
```

Pcap filename prefix
Device to trace
Type of trace
Override prefix

- 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 NS3 object name service

```
Names::Add ("server" ...);
Names::Add ("server/eth0" ...);
...
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 NS3 object name service to replace ids with meaningful names

e.g.,

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

↓

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

↓

```
prefix-server-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 NS3 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 NS3 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 NS3 object name service clarity
  - explicitFilename parameter lets you impose your own filenames

Conclusion

- NS3 is very powerful/comprehensive
- Lots of tools for you to use
  - Helpers
  - Containers
  - Logging
  - Tracing
  - Models
- Doxygen is your friend!

Practice! Practice! Practice!