NS-3 Tutorial

Tom Henderson (University of Washington and Boeing Research & Technology)
Mathieu Lacage (Alcméon)

March 2013
ns-3 tutorial agenda

• 13h00-15h00: Getting started with ns-3
  • Overview of software and models
  • Basic structure of the core and important models
  • Running and understanding an existing example
  • Animation and visualization

• 15h00-15h30: 30-minute coffee break

• 15h30-17h00: Going further with ns-3
  • Writing and debugging your own examples
  • Integrating other tools and libraries
  • Parallel simulations
  • Emulation, virtual machine and testbed integration
  • Getting help and getting involved
Preliminaries

• ns-3 is written in C++, with bindings available for Python
  – simulation programs are C++ executables or Python programs
  – ~300,000 lines of mostly C++ (estimate based on cloc source code analysis)

• ns-3 is a GNU GPLv2-licensed project

• ns-3 is mainly supported for Linux, OS X, and FreeBSD

• ns-3 is not backwards-compatible with ns-2
Preliminaries (cont.)

• Where do I get ns-3?
  – http://www.nsnam.org

• Where do I get today's code?
What have people done with ns-3?

- ~300 publications to date
  - search of 'ns-3 simulator' on IEEE and ACM digital libraries

NS-3 Consortium Meeting
March 2013
What have people done with ns-3?

• Educational use (from ns-3 wiki)

### Using ns-3 in Education

This page is a resource for learning about ns-3 as an educational tool for networking education.

#### Papers

The [2011 Sigcomm Education workshop](http://example.com) had a paper regarding ns-3 use in the classroom:

- [An Open-source and Declarative Approach Towards Teaching Large-scale Networked Systems Programming](http://example.com)

#### Courses using ns-3

The following courses have used ns-3 as courseware or to support projects:

- Georgia Tech. ECE 6110, Dr. George Riley, Spring 2013 (also Fall 2011, Fall 2010)
- The University of Kansas EECS 780, EECS 882, and EECS 983, Dr. James Sterbenz, 2010 – 2012
- UPenn CIS 553/TCOM 512, Dr. Boon Thau Loo, Fall 2010
- Aalto University, Jose Costa-Requena and Markus Peukuri, Fall 2011
- Indian Institute of Technology Bombay, Bhaskaran Raman, Autumn 2008
- University of Rijeka
  - RM2-InfUniRi, Dr. Mario Radovan and Vedran Miletic, Spring 2013, also Spring 2012
  - RM-RiTeh, Dr. Mladen Tomić and Vedran Miletic, Spring 2013

#### Other resources

- Lalith Suresh’s [Lab Assignments using ns-3 page](http://example.com)
Software introduction

• Download the latest release
  – tar xjf ns-allinone-3.16.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 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**

<table>
<thead>
<tr>
<th>Devices</th>
<th>Applications</th>
<th>Protocols</th>
<th>Utilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>bridge</td>
<td>applications</td>
<td>aodv</td>
<td>config-store</td>
</tr>
<tr>
<td>csma</td>
<td>internet (IPv4/v6)</td>
<td>dsdv</td>
<td>flow-monitor</td>
</tr>
<tr>
<td>emu</td>
<td>energy</td>
<td>olsr</td>
<td>netanim</td>
</tr>
<tr>
<td>point-to-point</td>
<td>mpi</td>
<td>click</td>
<td>stats</td>
</tr>
<tr>
<td>lte</td>
<td>mobility</td>
<td>nix-vector-routing</td>
<td>topology-read</td>
</tr>
<tr>
<td>mesh</td>
<td>propagation</td>
<td>openflow</td>
<td></td>
</tr>
<tr>
<td></td>
<td>core</td>
<td>BRITE</td>
<td></td>
</tr>
</tbody>
</table>
Current models

- **devices**
  - bridge
  - csma
  - emu
  - point-to-point
  - virtual
  - mesh
  - wimax

- **applications**
  - Node class
  - NetDevice ABC
  - Address types (IPv4, MAC, etc.)
  - Queues
  - Socket ABC
  - IPv4 ABCs
  - Packet sockets

- **protocols**
  - internet (IPv4/v6)
  - energy
  - Packets
  - Packet Tags
  - Packet Headers
  - Pcap/ascii file writing

- **utilities**
  - ns3-vector-routing
  - aodv
  - dsdv
  - olsr
  - click
  - openflow
  - BRITE

**Smart pointers**
- Dynamic types
- Attributes
- Callbacks
- Tracing
- Logging
- Random Variables

**NS-3 Consortium Meeting**
March 2013
Module organization

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

• Two levels of ns-3 build

1) build.py (a custom Python build script to control an ordered build of ns-3 and its libraries)

2) waf, a build system written in Python
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
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
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() gets it all started
•Simulation stops at specific time or when events end
Simulator example

```cpp
#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
# 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)
```
Command-line arguments

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

```c
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

```bash
./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
  - Avoid 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
- Time objects can be set by floating-point values and can export floating-point

  ```
  double timeDouble = t.GetSeconds();
  ```
Events in ns-3

• Events are just function calls that execute at a simulated time
  – i.e. callbacks
• 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.
- A "RealTime" simulation implementation is possible — aligns the simulation time to wall-clock time.
Random Variables

- 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.xlabel('t')
plt.ylabel('f(t) for t \in \text{range}(10000)')
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
  – Period of PRNG is $3.1 \times 10^{57}$

• Partitions a pseudo-random number generator into uncorrelated *streams* and *substreams*
  – Each RandomVariableStream gets its own stream
  – This stream partitioned into substreams
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 an uncorrelated substream.
Putting it together

• Example of scheduled event

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

```cpp
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...
Build variations

• Configure 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 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
APIs

• Most of the ns-3 API is documented with Doxygen
  – http://www.stack.nl/~dimitri/doxygen/
Review of topics covered

- Software layout
- Software build
- Library documentation
- Basic discrete-event simulation concepts
- Control of randomness
- Simulation time
- A simple C++ ns-3 program
- A simple Python ns-3 program
Walkthrough of WiFi Internet example
The basic model

Application

Protocol stack

NetDevice

Node

Sockets-like API

Packet(s)

Channel

Protocol stack

NetDevice

Node

Application
Example program

- examples/wireless/wifi-simple-adhoc-grid.cc

- examine wscript for necessary modules
  - 'internet', 'mobility', 'wifi', 'config-store', 'tools'
  - we'll add 'visualizer'

- ./waf configure --enable-examples --enable-modules=...
Example program

- (5x5) grid of WiFi ad hoc nodes
- OLSR packet routing
- Try to send packet from one node to another

Source (node 24) by default

• Goal is to read and understand the high-level ns-3 API

Sink (node 0) by default
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

WifiChannel

Nodes are architected for multiple interfaces
Internet Stack

• Internet Stack
  – Provides IPv4 and some IPv6 models currently

• No non-IP stacks presently in ns-3
  – but no dependency on IP in the devices, Node, Packet, etc.
  – some activity on IEEE 802.15.4-based models
Other basic models in ns-3

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

```c
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);

YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::FriisPropagationLossModel");
wifiPhy.SetChannel (wifiChannel.Create ());

// Add a non-QoS upper mac, and disable rate control
NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
wifi.SetStandard (WIFI_PHY_STANDARD_80211b);
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
    "DataMode",StringValue (phyMode),
    "ControlMode",StringValue (phyMode));

// Set it to adhoc mode
wifiMac.SetType ("ns3::AdhocWifiMac");
NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, c);

MobilityHelper mobility;
```
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"
Helper Objects

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

- (5x5) grid of WiFi ad hoc nodes
- OLSR packet routing
- Try to send packet from one node to another

Let’s look closely at how these objects are created

Source (node 24) by default
Sink (node 0) by default
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);
...
Mobility models in ns-3

• The MobilityModel interface:
  – void SetPosition (Vector pos)
  – Vector GetPosition ()

• StaticMobilityModel
  – Node is at a fixed location; does not move on its own

• RandomWaypointMobilityModel
  – (works inside a rectangular bounded area)
  – Node pauses for a certain random time
  – Node selects a random waypoint and speed
  – Node starts walking towards the waypoint
  – When waypoint is reached, goto first state

• RandomDirectionMobilityModel
  – works inside a rectangular bounded area)
  – Node selects a random direction and speed
  – Node walks in that direction until the edge
  – Node pauses for random time
  – Repeat
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

• Several options exist:
  – native ns-3 TCP
    – Tahoe, Reno, NewReno (others in development)
  – TCP simulation cradle (NSC)
  – Use of virtual machines or DCE (more on this later)

• To enable NSC:
  
  `internetStack.SetNscStack ("liblinux2.6.26.so");`
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

Accuracy

• Have shown NSC to be very accurate – able to produce packet traces that are almost identical to traces measured from a test network

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++

Figure reference: S. Jansen, Performance, validation and testing with the Network Simulation Cradle. MASCOTS 2006.
IPv4 address configuration

- An Ipv4 address helper can assign addresses to devices in a NetDevice container

```cpp
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);
```
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

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

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

```c
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));
```

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

ns-3 sockets

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

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

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

```c
sk->SetReceiveCallback (MakeCallback (MySocketReceive));
```

```c
void MySocketReceive (Ptr<Socket> sk, Ptr<Packet> packet)
{
    ...
}
```
ns-3 tutorial agenda

• 13h00-15h00: Getting started with ns-3
  • Overview of software and models
  • Basic structure of the core and important models

• 15h00-15h30: 30-minute coffee break

• 15h40-17h15: Going further with ns-3
  • Running and understanding an existing example
  • Animation and visualization
  • Writing and debugging your own examples
  • Integrating other tools and libraries
  • Parallel simulations
  • Emulation, virtual machine and testbed integration
  • Getting help and getting involved
Attributes and default values

```c
// disable fragmentation for frames below 2200 bytes
Config::SetDefault("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue("2200");

// 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);
```
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 \texttt{ns3::Object} get several additional features
  – dynamic run-time object aggregation
  – an attribute system
  – smart-pointer memory management (Class \texttt{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

```cpp
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

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

• Syntax also supports string values:

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

![Diagram of Attribute and Value](image)
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

```cpp
Config::SetAttribute("/NodeList/5/DeviceList/3/Phy/TxGain", DoubleValue(1.0));
DoubleValue d; nodePtr->GetAttribute( "/NodeList/5/NetDevice/3/Phy/TxGain", v);
```

• Users can get Ptrs to instances also, and Ptrs to trace sources, in the same way
ns-3 attribute system

- Object attributes are organized and documented in the Doxygen
- Enables the construction of graphical configuration tools:
Attribute documentation

The list of all attributes.

[Core]

Collaboration diagram for 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:
  
  ```cpp
  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++:
  
  ```cpp
  Config::SetDefault ("ns3::Ipv4L3Protocol::CalcChecksum", BooleanValue (true));
  ```
- Set an attribute directly on a specific object:
  
  ```cpp
  Ptr<CsmaChannel> csmaChannel = ...;
  csmaChannel->SetAttribute ("DataRate", StringValue ("5Mbps"));
  ```
Object names

• It can be helpful to refer to objects by a string name
  – “access point”
  – “eth0”

• Objects can now be associated with a name, and the name used in the attribute system
Names example

NodeContainer n;
n.Create (4);
Names::Add ("client", n.Get (0));
Names::Add ("server", n.Get (1));
...

Names::Add ("client/eth0", d.Get (0));
...

Config::Set ("/Names/client/eth0/Mtu", UintegerValue (1234));

Equivalent to:

Config::Set ("/NodeList/0/DeviceList/0/Mtu", UintegerValue (1234));
Tracing and statistics

• Tracing is a structured form of simulation output

• Example (from ns-2):
  
  + 1.84375 0 2  cbr 210 -------  0 0.0 3.1 225 610
  - 1.84375 0 2  cbr 210 -------  0 0.0 3.1 225 610
  r 1.84471 2 1  cbr 210 -------  1 3.0 1.0 195 600
  r 1.84566 2 0  ack 40 -------  2 3.2 0.1  82  602
  + 1.84566 0 2  tcp 1000 -------  2 0.1 3.2 102 611

Problem: Tracing needs vary widely

  – would like to change tracing output without editing the core

  – would like to support multiple outputs
Tracing overview

- Simulator provides a set of pre-configured trace sources
  - Users may edit the core to add their own
- Users provide trace sinks and attach to the trace source
  - Simulator core provides a few examples for common cases
- Multiple trace sources can connect to a trace sink
Tracing in ns-3

• ns-3 configures multiple 'TraceSource' objects (TracedValue, TracedCallback)
• Multiple types of 'TraceSink' objects can be hooked to these sources
• A special configuration namespace helps to manage access to trace sources

```cpp
Config::Connect("/path/to/traced/value", callback1);
Config::Connect("/path/to/trace/source", callback2);
```

TraceSource  unattached
**NetDevice trace hooks**

- **Example:** CsmaNetDevice

  ```
  CsmaNetDevice::Send()
  ```

  ```
  CsmaNetDevice::TransmitStart()
  ```

  ```
  CsmaNetDevice::Receive()
  ```

  ```
  NetDevice::ReceiveCallback
  ```

  ```
  CsmaChannel
  ```

  ```
  queue
  ```

  ```
  PhyTxBegin
  ```

  ```
  PhyTxEnd
  ```

  ```
  PhyTxDrop
  ```

  ```
  PhyRxBegin
  ```

  ```
  PhyRxEnd
  ```

  ```
  PhyRxDrop
  ```

  ```
  MacTx
  ```

  ```
  MacTxBackoff
  ```

  ```
  MacRx
  ```

  ```
  MacDrop
  ```

  ```
  Sniffer
  ```

  ```
  PromiscSniffer
  ```

  ```
  PhyRxEnd
  ```

  ```
  PhyRxDrop
  ```

  ```
  PhyTx
  ```

  ```
  TransmitStart()
  ```

  ```
  Receive()
  ```

  ```
  ns-3 Consortium Meeting
  March 2013
  ```
Enabling tracing in your code

• examples/tutorial/third.cc

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

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

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

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

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

Device helpers provide common API for enabling pcap traces

Global pcap tracing

Per-device pcap tracing
Discovering ns-3 trace sources

- various trace sources (e.g., packet receptions, state machine transitions) are plumbed through the system
- Organized with the rest of the attribute system

The list of all trace sources.

**ns3::WiFiNetDevice**
- Rx: Received payload from the MAC layer.
- Tx: Send payload to the MAC layer.

**ns3::WiFiPhy**
- State: The WiFiPhy state
- RxD: A packet has been received successfully.
- RxE: A packet has been received unsuccessfully.
- TxA: Packet transmission is starting.

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

**ns3::olsr::Agentimpl**
- Rx: Receive OLSR packet.
- Tx: Send OLSR packet.
- RoutingTableChanged: The OLSR routing table has changed.

**ns3::PacketSink**
Basic tracing

- Helper classes hide the tracing details from the user, for simple trace types—ascii or pcap traces of devices

```cpp
if (tracing == true)
{
    AsciiTraceHelper ascii;
    wifiPhy.EnableAsciiAll (ascii.CreateFileStream ("wifi-simple-adhoc-grid.tr"));
    wifiPhy.EnablePcap ("wifi-simple-adhoc-grid", devices);
    // Trace routing tables
    Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper> ("wifi-simple-adhoc-grid.routes", std::ios::out);
    olsr.PrintRoutingTableAllEvery (Seconds (2), routingStream);

    // To do-- enable an IP-level trace that shows forwarding events only
}
```
Multiple levels of tracing

- Highest-level: Use built-in trace sources and sinks and hook a trace file to them
- Mid-level: Customize trace source/sink behavior using the tracing namespace
- Low-level: Add trace sources to the tracing namespace
  - Or expose trace source explicitly
Highest-level of tracing

• Highest-level: Use built-in trace sources and sinks and hook a trace file to them

// Also configure some tcpdump traces; each interface will be traced
// The output files will be named
// simple-point-to-point.pcap-<nodeId>-<interfaceId>
// and can be read by the "tcpdump -r" command (use "-tt" option to
// display timestamps correctly)
PcapTrace pcaptrace ("simple-point-to-point.pcap");
pcaptrace.TraceAllIp ();
Mid-level of tracing

- Mid-level: Customize trace source/sink behavior using the tracing namespace

```cpp
void PcapTrace::TraceAllIp (void)
{
    NodeList::Connect ("/nodes/*/ipv4/(tx|rx)",
                        MakeCallback (&PcapTrace::LogIp, this));
}
```

Regular expression editing

Hook in a different trace sink
void
AsciiTrace::TraceAllQueues (void)
{
    Packet::EnableMetadata ();
    NodeList::Connect ("/nodes/*/devices/*/queue/enqueue",
                        MakeCallback (&AsciiTrace::LogDevQueueEnqueue, this));
    NodeList::Connect ("/nodes/*/devices/*/queue/dequeue",
                        MakeCallback (&AsciiTrace::LogDevQueueDequeue, this));
    NodeList::Connect ("/nodes/*/devices/*/queue/drop",
                        MakeCallback (&AsciiTrace::LogDevQueueDrop, this));
}
Lowest-level of tracing

- Low-level: Add trace sources to the tracing namespace

```cpp
Config::Connect("/NodeList/.../Source",
    MakeCallback(&ConfigTest::ChangeNotification, this));
```
Review of topics covered

- Structure of an ns-3 program
- Fundamental classes
  - Nodes, NetDevices, Channels, Applications
- Node and device containers
- Helper APIs, and Install pattern
- Wifi and Internet stack architecture
- Attributes and default values
- Tracing
Animation and visualization
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

FlowMonitor architecture

• Basic classes
  – FlowMonitor
  – FlowProbe
  – FlowClassifier
  – FlowMonitorHelper

• Ipv4 only

FlowMonitor statistics

- Statistics gathered

FlowMonitor configuration

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

```cpp
// 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: 386660
  Throughput: 0.233963 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Bytes: 3848412
  Rx Bytes: 274740
  Throughput: 0.20961 Mbps
```

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 
Built-in data output   : not enabled (option --enable-sudo not selected)
```

- If program supports CommandLine parsing, pass the option
  
  --SimulatorImplementationType=
  \texttt{ns3::VisualSimulatorImpl}

- Alternatively, pass the "--vis" option
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 metadata.
• Node position statistics with node trajectory plotting (path of a mobile node).
• Print brief packet-meta data on packets
Writing and debugging your own examples
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
Reading existing code

• Much insight can be gained from reading ns-3 examples and tests, and running them yourselves

• Many core features of ns-3 are only demonstrated in the core test suite (src/core/test)

• Stepping through code with a debugger can be done, but callbacks and templates make it more challenging than usual
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>

- Note: valgrind can be run similarly
  
  ./waf --command-template="valgrind %s" --run <program-name>
Testing

• Can you trust ns-3 simulations?
  – Can you trust any simulation?
    • Onus is on the simulation project to validate and document results
    • Onus is also on the researcher to verify results

• ns-3 strategies:
  – regression and unit tests
    • Aim for event-based rather than trace-based
  – validation of models on testbeds
  – reuse of code
Test framework

- ns-3-dev is checked nightly on multiple platforms
  - Linux gcc-4.x, i386 and x86_64, OS X i386, FreeBSD 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
Integrating other tools and libraries
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
Matplotlib

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

```python
# 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()
```
Other libraries

- ns-3 supports additional libraries (click, openflow, nsc)
- ns-3 has optional libraries (libxml2, gsl, mysql)
- both are typically enabled/disabled through the wscript
- users are free to write their own Makefiles or wscripts to do something special
Scaling to multiple machines
Overview

• Parallel and distributed discrete event simulation
  – Allows single simulation program to run on multiple interconnected processors
  – Reduced execution time! Larger topologies!

• Terminology
  – Logical process (LP)
  – Rank or system id
Simulation size record

- Simulation on a HPC cluster at the U.S. Mobile Network Modeling Institute (2011) *
  - 176 cores, 3 TB of memory
  - 360,448,000 simulated nodes
  - 413,704.52 packet receive events per second [wall-clock]

Quick and Easy Example

Figure 1. Simple point-to-point topology
Quick and Easy Example

Figure 2. Simple point-to-point topology, distributed

NS-3 Consortium Meeting
March 2013
Implementation Details

• LP communication
  – Message Passing Interface (MPI) standard
  – Send/Receive time-stamped messages
  – MpiInterface in ns-3

• Synchronization
  – Conservative algorithm using lookahead
  – DistributedSimulator in ns-3
Implementation Details (cont.)

• Assigning rank
  – Currently handled manually in simulation script
  – Next step, MpiHelper for easier node/rank mapping

• Remote point-to-point links
  – Created automatically between nodes with different ranks through point-to-point helper
  – Packet sent across using MpiInterface
Implementation Details (cont.)

• Distributing the topology
  – All nodes created on all LPs, regardless of rank
  – Applications are only installed on LPs with target node
Performance Test

• DARPA NMS campus network simulation
  – Allows creation of very large topologies
  – Any number of campus networks are created and connected together
  – Different campus networks can be placed on different LPs
  – Tested with 2 CNs, 4 CNs, and 6 CNs
Campus Network Topology

Links between campus: 2Gbps, 200ms
Links on campus (non-LAN): 1Gbps, 5ms
Links on campus (LAN): 100Mbps, 1ms

Net 0

Connecting to other campus

Net 1

Net 2

Net 3

5 sub LANs with 42 hosts (+1 switch)

nodes per campus = 538
2 Campus Networks

Figure 5. Execution time with 2 campus networks

![Graph showing execution time with 2 campus networks.]

Figure 6. Speedup with 2 LPs

![Graph showing speedup with 2 LPs.]

NS-3 Consortium Meeting
March 2013
Summary

• Distributed simulation in ns-3 allows a user to run a single simulation in parallel on multiple processors.

• By assigning a different rank to nodes and connecting these nodes with point-to-point links, simulator boundaries are created.

• Simulator boundaries divide LPs, and each LP can be executed by a different processor.

• Distributed simulation in ns-3 offers solid performance gains in time of execution for large topologies.
emulation and testbeds
Emulation support

• Support moving between simulation and testbeds or live systems

• A real-time scheduler, and support for two modes of emulation
  
  GlobalValue::Bind ("SimulatorImplementationType", 
  StringValue ("ns3::RealTimeSimulatorImpl"));
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
"Tap" mode: netns and ns-3 integration

Linux (FC 12 or Ubuntu 9.10) machine

Container

tapX

ns-3

ghost node

Wifi

TapBridge

Wireless

/dev/tunX

ghost node

Container

tapY

Tap device pushed into namespaces; no bridging needed
Example: ORBIT and ns-3

• Support for use of Rutgers WINLAB ORBIT radio grid
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

• 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
Container-based virtual machines and ns-3
What is CORE?

- The Common Open Research Emulator (CORE) is a Python framework and GUI for emulating networks using lightweight Virtualization native to Linux and FreeBSD kernels.
Wired networks
Visualize routing state
Lightweight VMs
Double-click for shell

Wireless networks
Technical Goals

- CORE provides Python libraries for using Linux network namespaces in network emulation experiments
  - CORE + ns-3 integrates realism of namespace with wireless device models
- CORE is a graphical controller that users find intuitive
  - CORE GUI could eventually be used for ns-3 authoring/visualization
Virtual Interfaces

• Ordinary CORE
  – Virtual Ethernet pairs (veth) are installed into a namespace and joined to a bridge.
  – For wireless networks (WLANs), ebtables rules govern pairwise connectivity.

• CORE + ns-3
  – TUN/TAP device installed into a namespace, socket held by simulation.
  – Simulation runs with real-time scheduler.
Mobility demonstration

Canvas-based mobility
• ns-3 ConstantPosition MobilityModel
• users can drag nodes around and change topology

ns-3 mobility visualization
• ns-3 RandomWalk Mobility Model
• users can observe Linux namespace state (e.g. OSPF adjacencies) as nodes move in the ns-3 realm
Scaling time in virtualized environments

• Synchronized Network Emulation - RWTH Aachen University
  – Modified Xen

• VAN Testbed – Telcordia/CERDEC
  – Modified Xen

• Linux Time namespace - Jeff Dike (UML creator)
  – Add a time namespace to the Linux kernel, allowing for gettimeofday() offsets
Direct Code Execution
Goals

• Lightweight virtualization of kernel and application processes, interconnected by simulated networks

• Benefits:
  – Implementation realism in controlled topologies or wireless environments
  – Model availability

• Limitations:
  – Not as scalable as pure simulation
  – Runs in real-time
  – Integration of the two environments
Direct Code Execution

- Developed by Mathieu Lacage and Frederic Urbani, INRIA, Hajime Tazaki (WIDE)
- Run unmodified application binaries in ns-3
  - Also, can run entire Linux stack in ns-3

Figure 4.6: The Linux network stack running inside ns-3

Figure source: Mathieu Lacage

http://www-sop.inria.fr/members/Frederic.Urbani/ns3dceccnx/index.html
NEPI
Network Experiment Management Framework (NEPI)

- Network experiment management framework to automate experiment life-cycle
- Allows scenarios involving heterogeneous resources (ns-3, PlanetLab, netns, …)
- Wiki: [http://nepi.inria.fr](http://nepi.inria.fr)

**Figure source:** Alina Quereilhac, INRIA
Getting Help and Getting Involved
Resources

Web site:
http://www.nsnam.org

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

IRC:  #ns-3 at freenode.net

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

Code server:
http://code.nsnam.org

Wiki:
http://www.nsnam.org/wiki/index.php/Main_Page
Questions?