ns-3 tutorial
ns-3 tutorial agenda

• 3:00-4:30: ns-3 current capabilities
  • Project overview
  • Walkthrough of basic simulation scenario
  • Parallel simulations and visualization
  • Emulation

• 4:30-4:40: 10-minute break

• 4:40-5:45: Work in progress
  • ns-3 development process
  • Automation
  • Direct code execution
  • Virtual machine and testbed integration

• 5:45-6:00: Q & A
Tutorial presenters

• Tom Henderson, UW/Boeing
• George Riley, Georgia Tech
• Felipe Perrone, Bucknell
• Mathieu Lacage, INRIA
ns-3 and GEC9

• ns-3 may be of interest to GENI researchers
• ns-3 and GENI are working on similar issues of experimentation workflow
• Several additional ns-3 events at GENI
  – Control Frameworks WG talk
  – Experimentation Services WG talk
  – ns-3 developers meeting, tomorrow
    – http://www.nsnam.org/wiki/
Acknowledgment of support
ns-3 Introduction

ns-3 is a free, open source software project building and maintaining a discrete-event network simulator for research and education

Technical goals:

• Build and maintain a simulation core aligned with the needs of the research community
• Help to improve the technical rigor of network simulation practice
ns-3: a brief history

1988: REAL (Keshav)
1990s: ns-1
1996: ns-2
1997-2000: DARPA VINT
2001-04: DARPA SAMAN, NSF CONSER
2006: NSF CISE CRI Award
2010: NSF CISE CRI Award
June 2008: ns-3.1
August 2010: ns-3.9

Inputs: yans, GTNetS, ns-2

ns-3 core development (2006-08)

quarterly releases

ns-3
ns-3 themes

• Research and education focus
  – Build and maintain simulation core, integrate models developed by other researchers
  – Support research-driven workflows

• Open source development model
  – Research community maintains the models

• Leverage available tools and models
  – Write programs to work together

• Enforce core coding/testing standards
ns-3 software overview

• ns-3 is written in C++, with bindings available for Python
  – simulation programs are C++ executables or Python programs
  – Python is often a glue language, in practice

• ns-3 is a GNU GPLv2-licensed project

• ns-3 is not backwards-compatible with ns-2
Relationship to ns-2

- Decided that we did not have resources to maintain backward compatibility with ns-2
  - OTcl and split-implementation models
  - Different level of abstraction
- Continuing to maintain ns-2 and nam
  - Possible to construct hybrid simulations
- Several models already ported to ns-3
  - Random number generators, OLSR, error models, recent WiFi Phy models
Goals of this tutorial

1) Describe what the software can do now

2) Summarize some current work in progress

3) Describe how the ns-3 project operates

4) Help you to get involved
Introductory Software Overview
Getting started: Linux

• Working from development version

  sudo apt-get install build-essential g++ python
  mercurial (for Ubuntu)
  hg clone http://code.nsnam.org/ns-3-allinone
  cd ns-3-allinone
  ./download.py
  ./build.py
  cd ns-3-dev

http://www.nsnam.org/tutorials/ns-3-allinone-geni.tar.bz2
Getting started: Windows

• Install build tools
  – Cygwin (g++, wget)
  – Python (http://www.python.org)

• Download

• Build
  – ./waf configure
  – ./test.py (runs unit tests)

• (rest of instructions similar to Linux)
Running programs

• Programs are built as
  build/<variant>/path/program-name
  – programs link shared library libns3.so

• Using ./waf --shell
  ./waf --shell
  ./build/debug/samples/main-simulator

• Using ./waf --run
  ./waf --run examples/csma-bridge
  ./waf --pyrun examples/csma-bridge.py
ns-3 uses 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

- For those familiar with autotools:
  - configure -> ./waf -d [optimized|debug] configure
  - make -> ./waf
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

/* -*- Mode:C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
#include "ns3/simulator.h"
#include "ns3/nstime.h"
#include <iostream>

using namespace ns3;

class MyModel {
public:
    void Start (void);
};

void
MyModel::Start (void)
{
    std::cout << "Starting" << std::endl;
}

static void
random_function (MyModel *model)
{
    std::cout << "random function received event at " <<
        Simulator::Now ().GetSeconds () << "s" << std::endl;
    model->Start ();
}

int main (int argc, char *argv[])
{
    MyModel model;
    Simulator::Schedule (Seconds (10.0), &random_function, &model);
    Simulator::Run ();
    Simulator::Destroy ();
}
A software organization view

Node class
NetDevice ABC
Address types (Ipv4, MAC, etc.)
Queues
Socket ABC
Ipv4 ABCs
Packet sockets

helper

Routing
Internet stack
Devices

node
mobility

common
simulator

core

High-level wrappers for everything else
No smart pointers
Aimed at scripting

Smart pointers
Dynamic type system
Attributes

Callbacks, Tracing
Logging
Random Variables

Packets
Packet Tags
Packet Headers
Pcap/ascii file writing

Mobility models (static, random walk, etc)

Events
Scheduler
Time arithmetic
APIs

- Most of the ns-3 API is documented with Doxygen
  - [http://www.stack.nl/~dimitri/doxygen/](http://www.stack.nl/~dimitri/doxygen/)
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,
  - ...

```python
import pylab
import ns3

rng = ns3.NormalVariable(10.0, 5.0)
x = [rng.GetValue() for t in range(100000)]

pylab.hist(x, 100)
pylab.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](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 RandomVariable gets its own stream
  - This stream partitioned into substreams
Run number vs. seed

• If you increment the seed of the PRNG, the RandomVariable streams 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
walkthrough of packet-passing example
Sample program: geni-helper.cc

- Four Wifi ad hoc nodes
- One additional node connected via CSMA

- Goal is to read and understand the high-level ns-3 API
- Part 2 of tutorial will look at a low-level program
The basic model

Application

Protocol stack

Node

NetDevice

Sockets-like API

Packet(s)

Channel

Channel

NetDevice

Application

Protocol stack

Node

NetDevice
Fundamentals

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

Nodes contain Applications, “stacks”, and NetDevices
Node basics

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

“DTN”
NetDevices and Channels

NetDevices are strongly bound to Channels of a matching type

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.
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, 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.
Structure of an ns-3 program

```c
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 sample program

```cpp
#include <ctype.h>
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include "ns3/core-module.h"
#include "ns3/helper-module.h"
#include "ns3/node-module.h"
#include "ns3/simulator-module.h"
#include "ns3/nam-helper.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TcpGeni");
```
Review of sample program (cont.)

```c
int main (int argc, char *argv[]) {
    Config::SetDefault ("ns3::DropTailQueue::MaxPackets", StringValue ("10"));

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

    // Here, we will explicitly create three nodes. The first container contains
    // nodes 0 and 1 from the diagram above, and the second one contains nodes
    // 1 and 2. This reflects the channel connectivity, and will be used to
    // install the network interfaces and connect them with a channel.
    NodeContainer n0n1;
    n0n1.Create (2);

    // We create the channels first without any IP addressing information
    // First make and configure the helper, so that it will put the appropriate
    // attributes on the network interfaces and channels we are about to install.
    PointToPointHelper p2p;
    p2p.SetDeviceAttribute ("DataRate", DataRateValue (DataRate (1000000000)));
    p2p.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (10)));

    // And then install devices and channels connecting our topology.
    NetDeviceContainer dev0 = p2p.Install (n0n1);
    p2p.SetDeviceAttribute ("DataRate", DataRateValue (DataRate (1000000000)));
    NetDeviceContainer dev1 = p2p.Install (n1n2);
```
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
Sample program (revisit)

- Four Wifi ad hoc nodes
- One additional node connected via CSMA

Let's look closely at how these objects are created.
int main (int argc, char *argv[]) {
    CommandLine cmd;
    cmd.Parse (argc, argv);

    NodeContainer csmaNodes;
    csmaNodes.Create (2);
    NodeContainer wifiNodes;
    wifiNodes.Add (csmaNodes.Get (1));
    wifiNodes.Create (3);

    NetDeviceContainer csmaDevices;
    CsmaHelper csma;
    csma.SetChannelAttribute ("DataRate", StringValue ("5Mbps"));
    csma.SetChannelAttribute ("Delay", StringValue ("2ms"));
    csmaDevices = csma.Install (csmaNodes);

    Create empty node container
    Create two nodes
    Create empty node container
    Add existing node to it
    and then create some more nodes
Review of sample program (cont.)

NetDeviceContainer wifiDevices;
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
wifiPhy.SetChannel (wifiChannel.Create ());
WifiHelper wifi = WifiHelper::Default ();
wifiDevices = wifi.Install (wifiPhy, wifiNodes);

MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::RandomDiscPositionAllocator",
  "X", StringValue ("100.0"),
  "Y", StringValue ("100.0"),
  "Rho", StringValue ("Uniform:0:30");
Wifi Mobility
mobility.SetMobilityModel ("ns3::StaticMobilityModel");
mobility.Install (wifiNodes);
Mobility models

- 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

3D Cartesian coordinate system
Review of sample program (cont.)

```
Ipv4InterfaceContainer csmaInterfaces;
Ipv4InterfaceContainer wifiInterfaces;
InternetStackHelper internet;
internet.Install (NodeContainer::GetGlobal ());
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
    csmaInterfaces = ipv4.Assign (csmaDevices);
ipv4.SetBase ("10.1.2.0", "255.255.255.0");
    wifiInterfaces = ipv4.Assign (wifiDevices);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

**Ipv4 configuration**

Routing
Internet stack

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

• Three options exist:
  – native ns-3 TCP
  – TCP simulation cradle (NSC)
  – Use of virtual machines (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

```
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);
```
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));
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));

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

```c
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));

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

• […] (Simulator::Run ())
Review of sample program (cont.)

```cpp
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));

std::ofstream ascii;
ascii.open("wns3-helper.tr");
CsmaHelper::EnableAsciiAll(ascii);
CsmaHelper::EnablePcapAll("wns3-helper");
YansWifiPhyHelper::EnablePcapAll("wsn3-helper");

GtkConfigStore config;
config.Configure();
```
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’ll talk about the other two features later
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::NscStack<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::WifiPhy::TxGain", 
               DoubleValue (1.0));
  ```

- Syntax also supports string values:
  
  ```
  Config::Set ("WifiPhy::TxGain", StringValue ("1.0");
  ```

```
```
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/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.
- MaxSrc: 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 specific object:
  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
ns-3 has a new tracing model

ns-3 solution: decouple trace sources from trace sinks

Benefit: Customizable trace sinks
ns-3 tracing

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

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

```cpp
std::ofstream ascii;
ascii.open("wns3-helper.tr");
CsmaHelper::EnableAsciiAll(ascii);
CsmaHelper::EnablePcapAll("wns3-helper");
YansWifiPhyHelper::EnablePcapAll("wsn3-helper");
```
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

```c
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

    Config::Connect("/NodeList/.../Source",
                   MakeCallback (&ConfigTest::ChangeNotification, this));
Callback Objects

- ns-3 Callback class implements *function objects*
  - Type safe callbacks, manipulated by value
  - Used for example in *sockets* and *tracing*

- Example

```cpp
double MyFunc (int x, float y) {
    return double (x + y) / 2;
}

[...]
```

```cpp
Callback<double, int, float> cb1;
```

```cpp
cb1 = MakeCallback (MyFunc);
```

```cpp
double result = cb1 (2,3); // result receives 2.5
```
Callback Objects

Class MyClass {
public:
    double MyMethod (int x, float y) {
        return double (x + y) / 2;
    }
};
[...] 
Callback<double, int, float> cb1;
MyClass myobj;
cb1 = MakeCallback(&MyClass::MyMethod, &myobj);
double result = cb1 (2, 3); // result receives 2.5
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
Validation

• 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
Regressions

• ns-3-dev is checked nightly on multiple platforms
  – Linux gcc-4.x, Linux gcc-3.4, i386 and x86_64, OS X
    i386 and ppc
• ./test.py will run regression tests
Improving performance

- Debug vs optimized builds
  - `./waf -d debug configure`
  - `./waf -d debug optimized`

- Build ns-3 with static libraries
  - Patch is in works

- Use different compilers (icc)
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
Quick and Easy Example

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

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

Figure 3. Mixed topology, distributed
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

Nodes per campus = 538
2 Campus Networks

Figure 5. Execution time with 2 campus networks

Figure 6. Speedup with 2 LPs
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 wireless simulation

Popular feature request
Wireless technology is everywhere
Wireless simulation is complex

Introduces new issues
Partitioning (We have mobility!)
Small propagation delay, small lookahead

Very large number of events
Sample Topology

Figure 11. Wireless network topology
Geographic Partitioning

Figure 12. Wireless network topology, partitioned
Node-based Partitioning

Figure 13. Wireless network topology, partitioned
Lookahead

Typical wireless scenarios present small lookahead due to node distances and the speed of light.

Small lookahead is detrimental to distributed simulation performance.

Possible optimizations:
- Protocol lookahead
- Event lookahead
Wireless Simulation Events

Wireless simulations require a large number of events

Increased inter-LP communication (bad)

Event Reduction

Decreases overhead

However, must ensure simulation fidelity
Event Reduction Techniques

Set a propagation limit
  - Carrier Sensing Threshold (too inaccurate?)
  - Popular distance limit

Lazy Updates
  - Leverage protocol mechanics and simulator knowledge
    - Ex: Lazy MAC state update

Event Bundling
  - Send fewer events but deliver the same information
    - Ex: LP-Rx event
Initial Development Plans

Geographic and node-based partitioning

Simple lookahead
  Assume minimal lookahead

Event Reduction
  Use carrier sensing threshold for propagation limit
  Use event bundling
Distributed Wireless Summary

People want distributed wireless
Implementing distributed wireless simulation should be easy
Optimizing distributed wireless simulation is hard
The good news is a great amount of research and previous implementations give us direction for optimization
Topologies and Visualization
Overview

ns-3 not directly funding visualizers and configurators

no “official” tool; expect multiple to be developed

Not integrated (directly) with ns-3

Ns-3 creates “animation” file, visualizers use this as input and create the animation.

netanim, pyviz, and nam for ns-3
Pyviz
Pyviz
NetVis
NetVis / XML Interface

<anim lp = "0">
<topology minX = "1000" minY = "2382.3" maxX = "10900" maxY = "8617.7">
<node lp = "0" id = "0" locX = "4000" locY = "5500"
    image = "Satellite.png" imageScale = "10.0"/>
<node lp = "0" id = "1" locX = "7000" locY = "5500"
    image = "Satellite.png" imageScale = "5.0"/>
<link fromLp = "0" fromId = "0" toLp = "0" toId = "1"/>
</topology>
<packet fromLp = "0" fromId = "11" fbTx = "0.66483" lbTx = "0.665264">
<rx toLp = "0" toId = "1" fbRx = "0.66583" lbRx = "0.666264"/>
</packet>
<wpacket fromLp = "0" fromId = "49" fbTx = "0.549245" lbTx = "0.549497"
    range = "250">
<rx toLp = "0" toId = "16" fbRx = "0.549497" lbRx = "0.549749"/>
<rx toLp = "0" toId = "18" fbRx = "0.549497" lbRx = "0.549749"/>
<rx toLp = "0" toId = "29" fbRx = "0.549497" lbRx = "0.549749"/>
<rx toLp = "0" toId = "32" fbRx = "0.549498" lbRx = "0.549749"/>
<rx toLp = "0" toId = "36" fbRx = "0.549498" lbRx = "0.549749"/>
<rx toLp = "0" toId = "37" fbRx = "0.549497" lbRx = "0.549749"/>
</wpacket>
</anim>
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
```

```
s-3
```

```
ghost node
```

```
TapBridge
```

```
WiFi
```

```
/dev/tunX
```

```
Container
```

```
tapY
```

```
ghost node
```

```
Wifi
```

```
/dev/tunY
```

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

• Support for use of Rutgers WINLAB ORBIT radio grid

ns-3

HOWTO use ns-3 directly on the ORBIT testbed hardware

We provide a runtime emulation package that allows us to connect ns-3 to real networks on real machines. Typically the real network will be a testbed of some kind. ORBIT is a two-tier laboratory emulator/field trial network project of WINLAB (Wireless Information Network Laboratory) at Rutgers. This wireless network emulator provides a large two-dimensional grid of 400 902.11 radio nodes as well as a number of smaller "sandbox" testbeds to allow one to test without rewiring the main grid. This HOWTO shows how ns-3 scripts can be used to drive these radio nodes.

We assume that you have some experience with the ORBIT system. If you are new to ORBIT, please take a look at http://www.orbit-lab.org/ and go through the "Basic Tutorial" and the "Tutorials on controlling the testbed nodes" at a minimum. We will assume throughout this HOWTO that you have registered for an ORBIT account and have made a reservation on the ORBIT Scheduler for a testbed. This HOWTO assumes that you are on the sandbox one (b1) testbed.

HOWTO use ns-3 directly on the ORBIT testbed hardware

We provide a node image on the ORBIT system that includes everything you need to get an ns-3 environment up and running on your testbed nodes. This includes the GNU toolchain, a copy of a precompiled ns-3.3 repository, emacs editor, etc. The first step is to get the environment up on the nodes in your testbed. In ORBIT terminology, we need to "image the nodes."
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