ns-3 tutorial

Tom Henderson
University of Washington
and
Mathieu Lacage
INRIA, Planete

Workshop on ns-3
March 2009

Workshop on ns-3 schedule

09h00-10h30:	Tutorial
10h30-11h00:	Coffee break
11h00-12h30:	Tutorial
12h30-14h00:	Lunch
14h00-16h00:	Focus on Wifi
16h00-16h30:	Coffee break
16h30-18h00:	Short talks

Focus on ns-3 Wifi

- **Authors:** Ruben Merz, Cigdem Sengul, and Mustafa Al-Bado
- **Title:** Accurate Physical Layer Modeling for Realistic Wireless Network Simulation

- **Authors:** Timo Bingmann and Jens Mittag
- **Title:** An overview of PHY-layer models in ns-3

- **Author:** Mirko Banchi
- **Title:** Realization of 802.11n and 802.11e models

- **Author:** Kirill V. Andreev
- **Title:** Realization of the draft standard for Mesh Networking (IEEE802.11s)

- **Author:** Guangyu Pei and Tom Henderson
- **Title:** 802.11b PHY model and validation

Short talks (miscellaneous)

- **Authors:** Ramon Bauza, Miguel Sepulcre, and Javier Gozalvez
- **Title:** ns-3 scalability constraints in heterogeneous wireless simulations: iTETRIS a case study

- **Authors:** Francisco Carmona, Juan Carlos Moreno, Ana Cabello, Francisco Lobo, and David Mora
- **Title:** ns-3 Script Generator

- **Authors:** Providence Salumu Munga and Hakima Chaouchi
- **Title:** An ns-3-based IEEE 802.21 MIH Module

- **Author:** Mohamed Amine Ismail
- **Title:** A Mobile WiMAX Module for ns-3

Goals of this tutorial

- Learn about the ns-3 project and its goals
- Understand the software architecture, conventions, and basic usage of ns-3
- Read and modify an example ns-3 script
- Learn how you might extend ns-3 to conduct your own research
- Provide feedback to the ns-3 development team

Assumptions

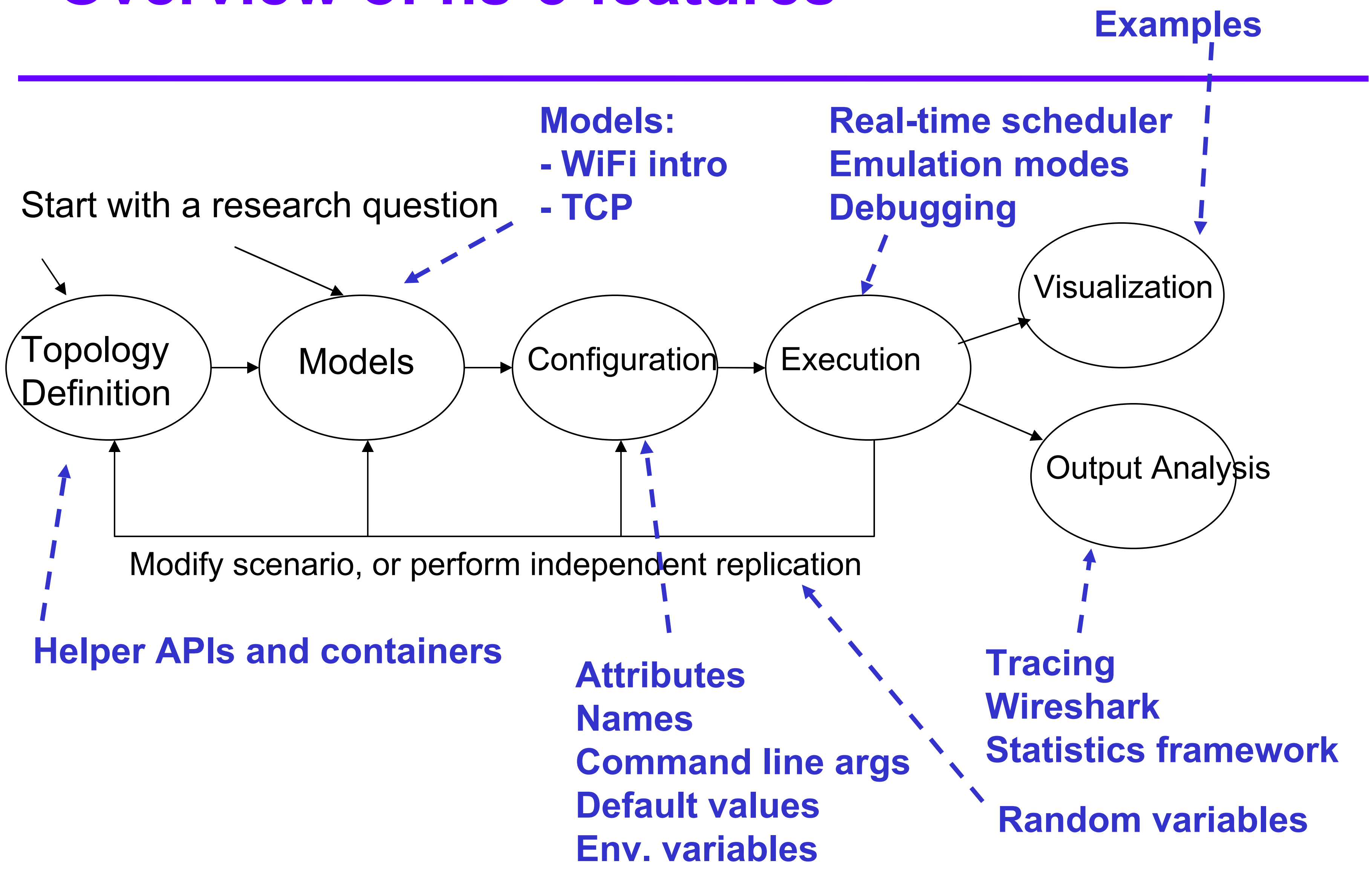
Some familiarity with:

- C++ and Python programming language
- TCP/IP
- Unix Network Programming (e.g., sockets)
- Discrete-event network simulation

Outline

1. Overview of ns-3 features
2. End-to-end perspective of the system
3. Extending ns-3
4. Advanced topics (time permitting)

Overview of ns-3 features



Introductory Software Overview

Basics

- ns-3 is written in C++
- Bindings in Python
- ns-3 uses the waf build system
  - i.e., instead of `./configure;make`, type `./waf`
- simulation programs are C++ executables or python scripts

Simulation basics

- Simulation time moves discretely 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

Scheduling events

```
/* -*- 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 ();
}
```

from samples/
main-simulation.cc

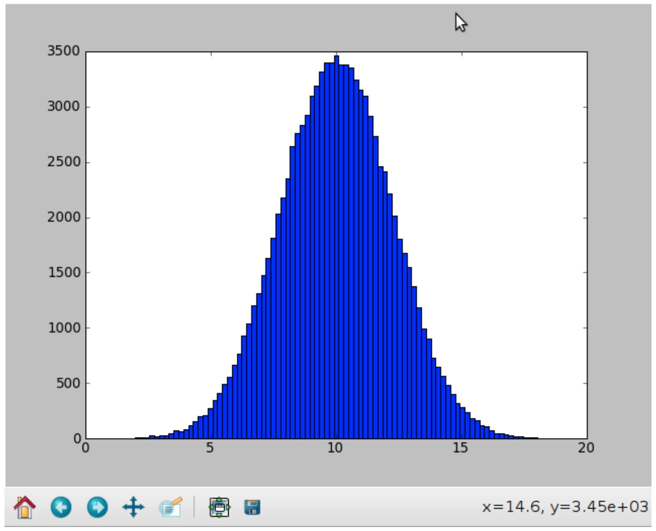
Introductory demo

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,
  - …

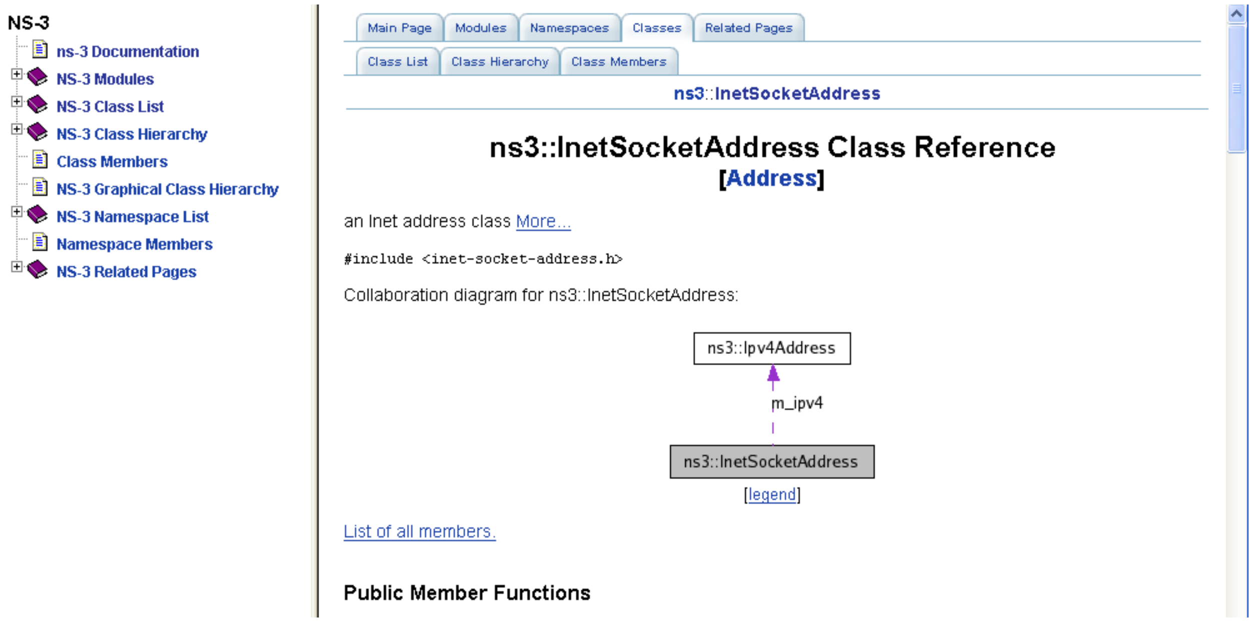
```
import pylab
import ns3
                                     (μ)  (σ)
rng = ns3.NormalVariable(10.0, 5.0)
x = [rng.GetValue() for t in range(100000)]

pylab.hist(x, 100)
pylab.show()
```



APIs

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



ns3::InetSocketAddress Class Reference
[Address]

an Inet address class More...

```
#include <inet-socket-address.h>
```

Collaboration diagram for ns3::InetSocketAddress:

List of all members.

Public Member Functions

the waf build system

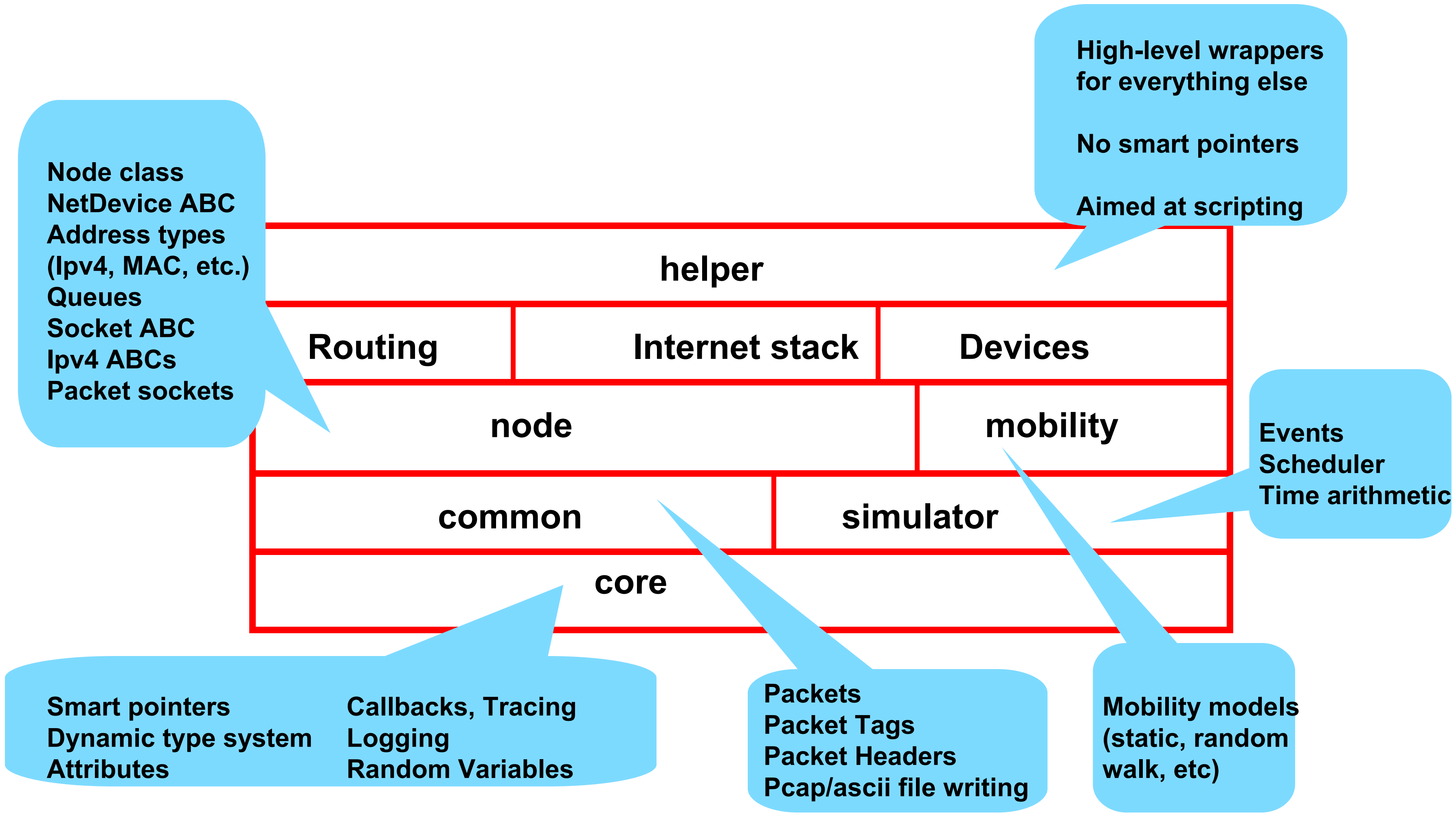
- Waf is a Python-based framework for configuring, compiling and installing applications.
  - It is a replacement for other tools such as Autotools, Scons, CMake or Ant
  - http://code.google.com/p/waf/

waf key concepts

- For those familiar with autotools:
- configure -> ./waf -d [optimized|debug] configure
- make -> ./waf
- make test -> ./waf check (run unit tests)

- Can run programs through a special waf shell; e.g.
  - `./waf --run simple-point-to-point`
  - (this gets the library paths right for you)

A software organization view



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
```

Building from within ns-3-dev

```
cd ns-3-dev
./waf distclean (similar to make distclean)
./waf configure
or ./waf -d optimized configure
./waf
```

- Helpful options:
  - -j# where # is number of cores
  - ./waf --help shows you other options

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

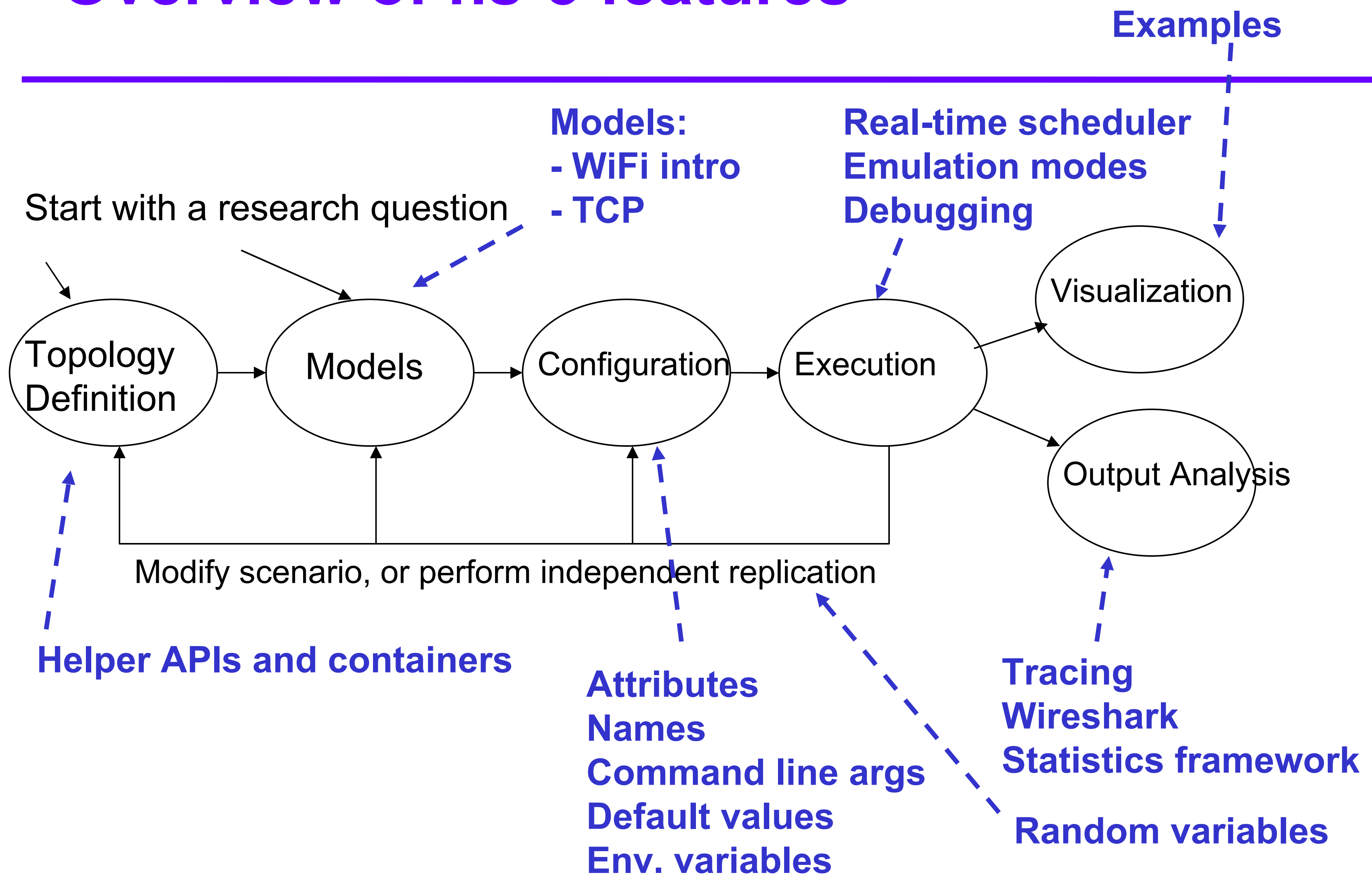
```
./waf --run examples/csma-bridge.cc
./waf --pyrun examples/csma-bridge.py
```

Getting started: Windows

- Install build tools
  - Cygwin (g++, wget)
  - Python (http://www.python.org)
- Download
  - wget http://www.nsnam.org/releases/ns-3.3.tar.bz2
- Build
  - ./waf configure
  - ./waf check (runs unit tests)
- (rest of instructions similar to Linux)

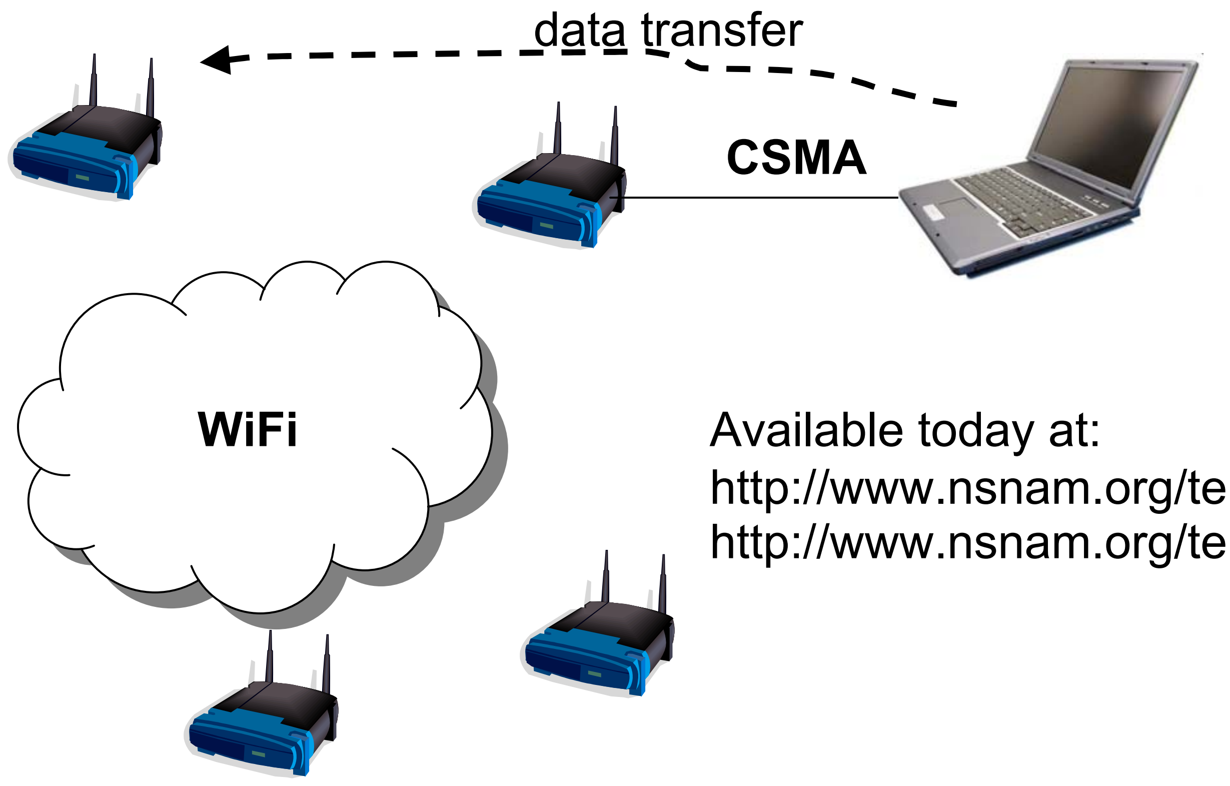
ns-3 features

Overview of ns-3 features



Sample program

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



Available today at:
http://www.nsnam.org/temp/wns3-helper.cc
http://www.nsnam.org/temp/wns3-lowlevel.cc

Review of sample program

```
#include <iostream>
#include <fstream>
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/core-module.h"
#include "ns3/helper-module.h"
#include "ns3/global-route-manager.h"
#include "ns3/contrib-module.h"

using namespace ns3;

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

Review of sample program (cont.)

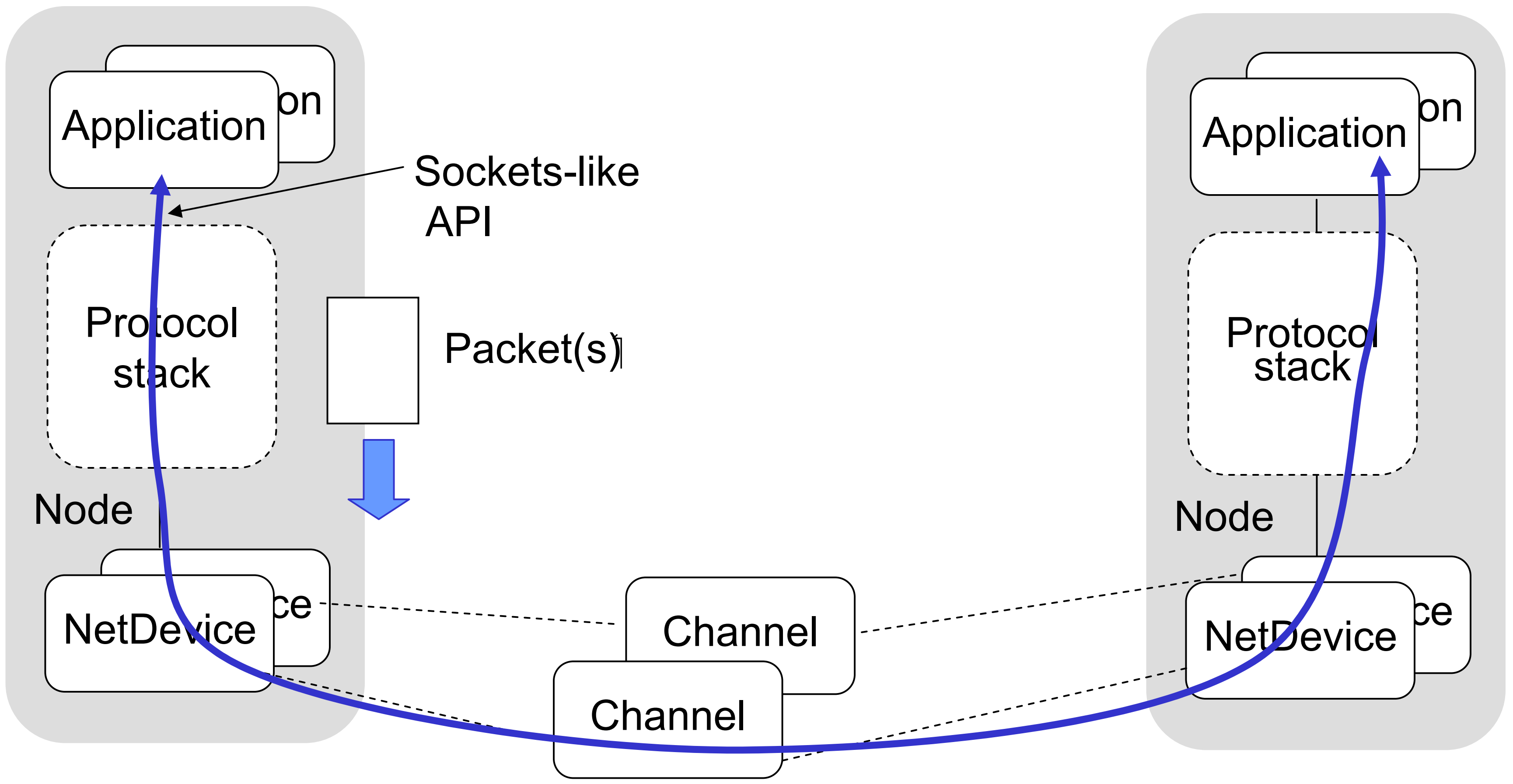
```
int main (int argc, char *argv[])
{
  CommandLine cmd;
  cmd.Parse (argc, argv);


  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);
```

Topology Configuration

The basic model



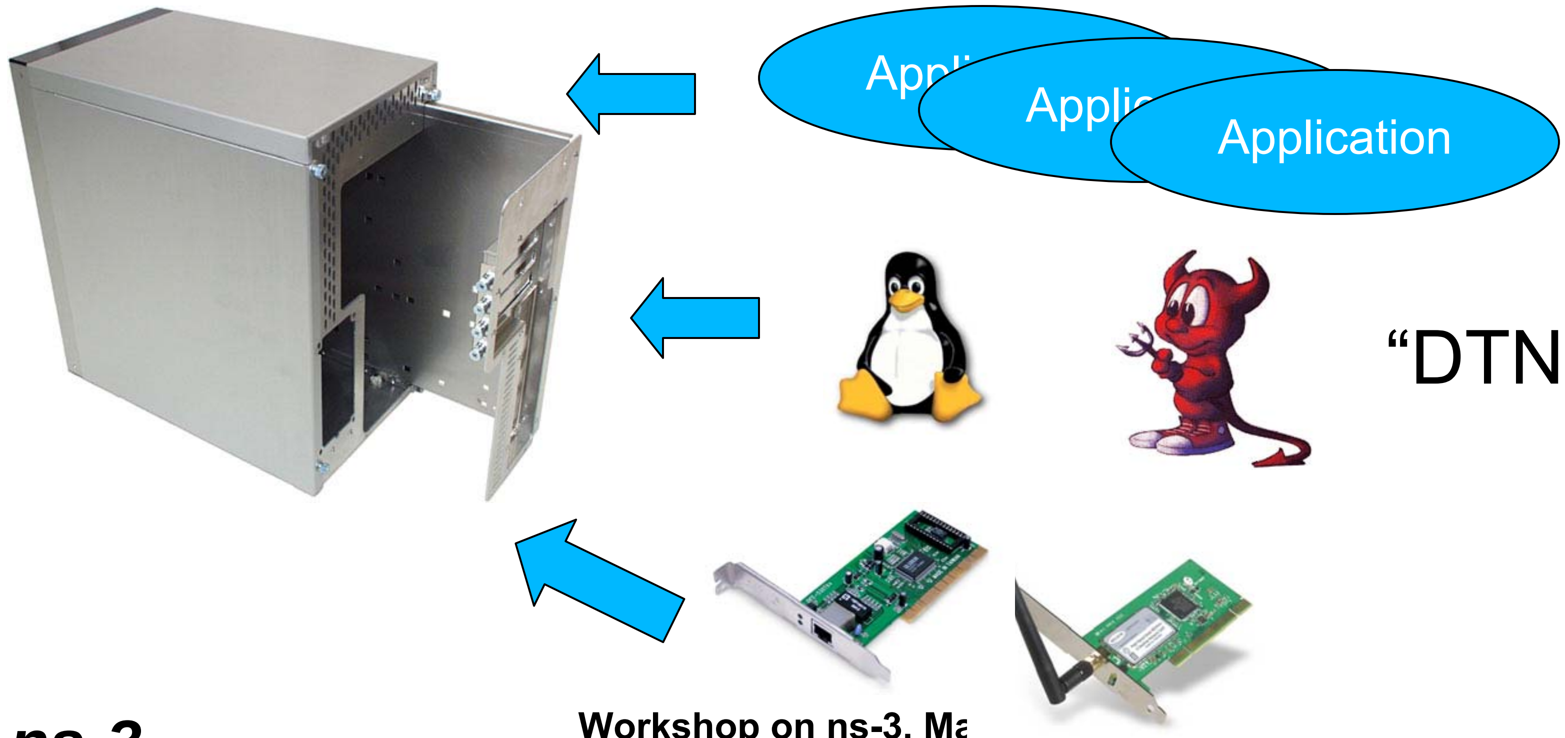
Fundamentals

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

Nodes contain Applications, “stacks”, and NetDevices

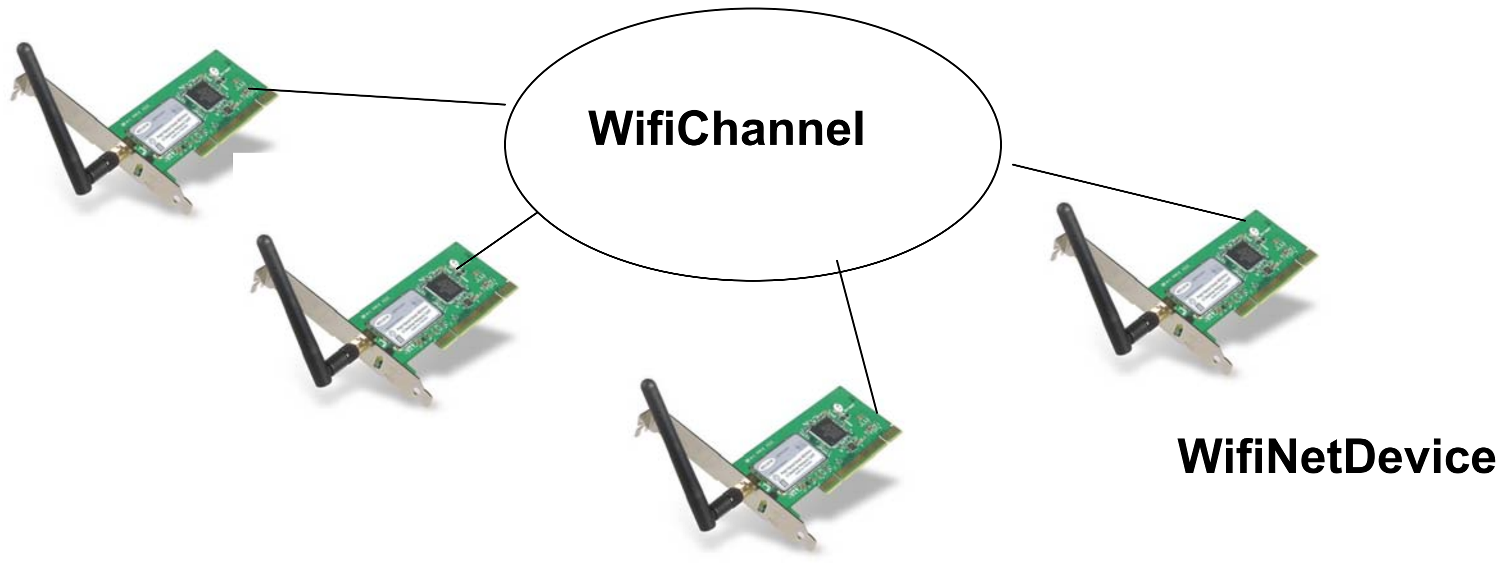
Node basics

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



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 models currently
  - IPv6 models are scheduled for ns-3.5/ns-3.6 timeframe
- Uses an interface design pattern to support multiple implementations

Other basic models in ns-3

- Devices
  - wifi, csma, point-to-point, bridge
- Error models and queues
- Applications
  - echo servers, traffic generator
- Mobility models

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)

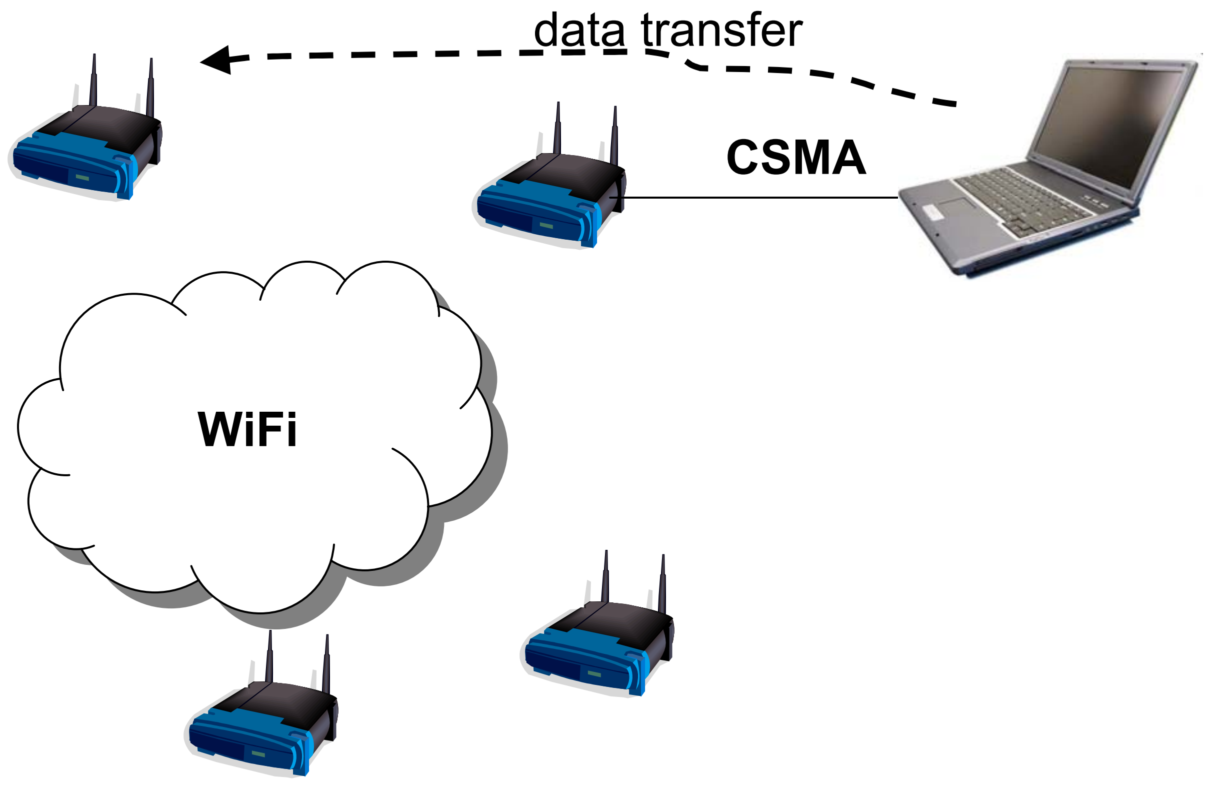
- Is not generic
- Does not try to allow code reuse
- Provides simple 'syntactical sugar' to make simulation scripts look nicer and easier to read for network researchers
- Each function applies a single operation on a "set of same objects"

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



Review of sample program (cont.)

```
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);
```

Wifi

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

Mobility

ns-3 Wifi model

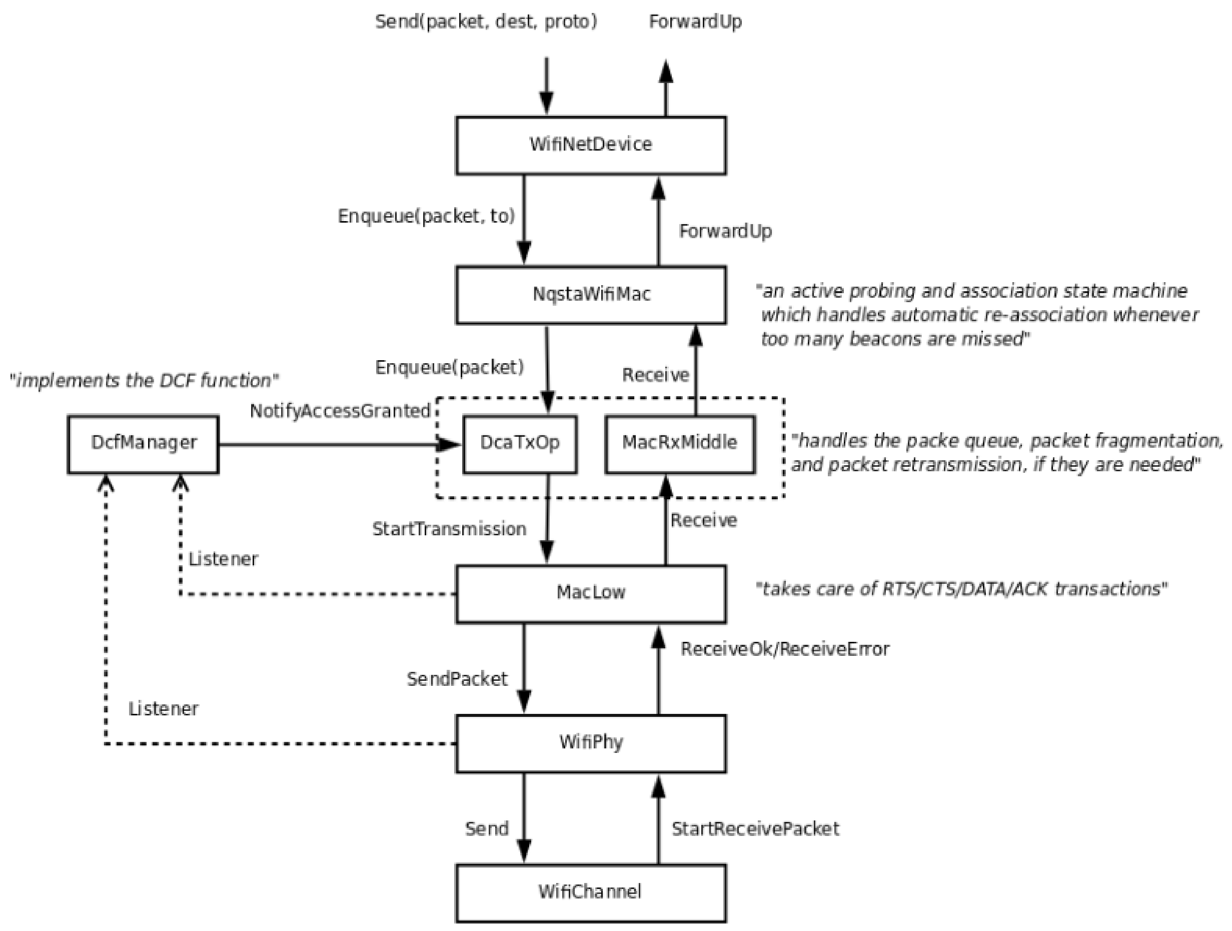
- new model, written from 802.11 specification
- accurate model of the MAC
- DCF, beacon generation, probing, association
- a set of rate control algorithms
- not-so-slow models of the 802.11a PHY

ns-3 Wifi development

Several research groups are maturing the original INRIA model:

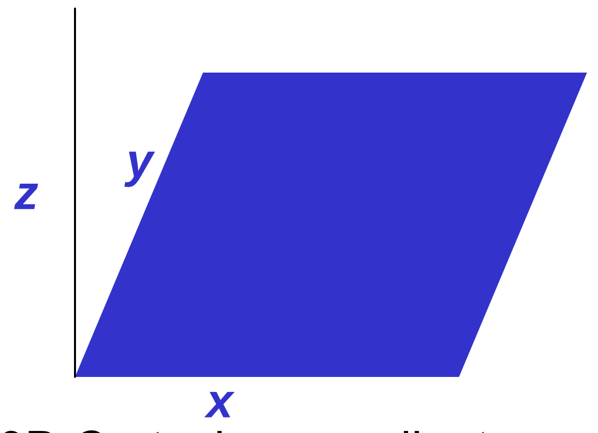
- Karlsruhe Institute of Technology: 802.11 PHY, 802.11e
  - Equalizing PHY models including capture effects, user-definable coding rates (e.g. 5.9 GHz from 802.11p), EDCA QoS extensions of 802.11e, Nakagami/Rayleigh propagation loss model
- University of Florence: 802.11n features
  - Frame Aggregation, Block ACK, HCF (EDCA and support for HCCA),TXOP, HT terminal (also with protection modes), MIMO
- Russian Academy of Sciences: 802.11s
  - a complete model of IEEE802.11s D2.0 Draft Standard
- Deutsche Telekom Laboratories in Berlin: 802.11 PHY
- Boeing: 802.11b channel models, validation
- (and others...)

ns-3 Wifi model (cont.)


Send(packet, dest, proto)
ForwardUp
WifiNetDevice
Enqueue(packet, to)
ForwardUp
NqstaWifiMac
"an active probing and association state machine which handles automatic re-association whenever too many beacons are missed"
Enqueue(packet)
Receive
"implements the DCF function"
NotifyAccessGranted
DcfManager
DcaTxOp
MacRxMiddle
"handles the packe queue, packet fragmentation, and packet retransmission, if they are needed"
StartTransmission
Receive
Listener
MacLow
"takes care of RTS/CTS/DATA/ACK transactions"
SendPacket
ReceiveOk/ReceiveError
Listener
WifiPhy
Send
StartReceivePacket
WifiChannel

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.)

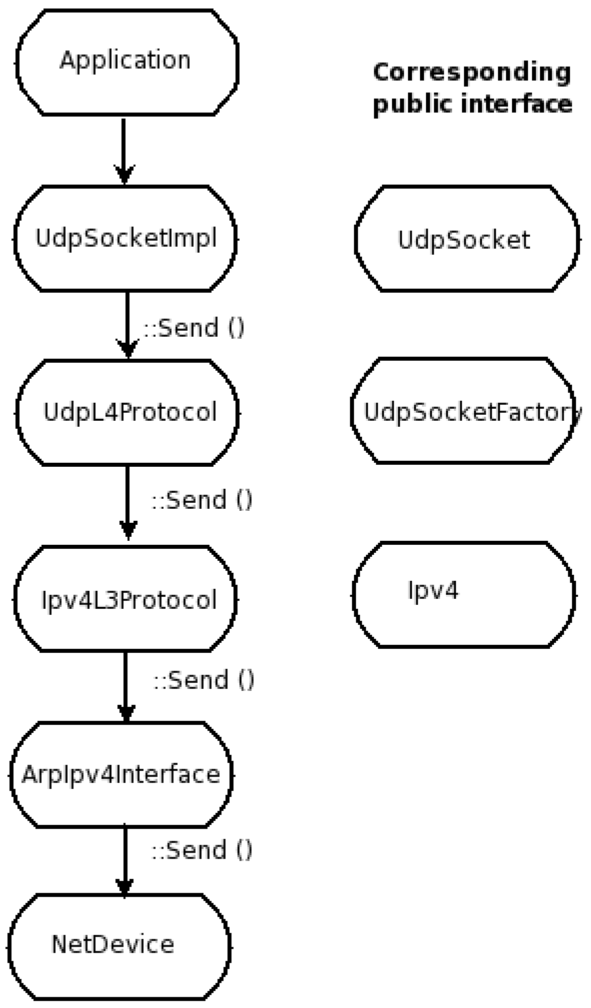
```
Ipv4InterfaceContainer csmaInterfaces;
Ipv4InterfaceContainer wifiInterfaces;
InternetStackHelper internet;
internet.Install (NodeContainer::GetGlobal ());
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
csmaInterfaces = ipv4.Assign (csmaDevices);
ipv4.SetBase ("10.1.2.0", "255.255.255.0");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

GlobalRouteManager::PopulateRoutingTables ();
```

Ipv4 configuration

Routing

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

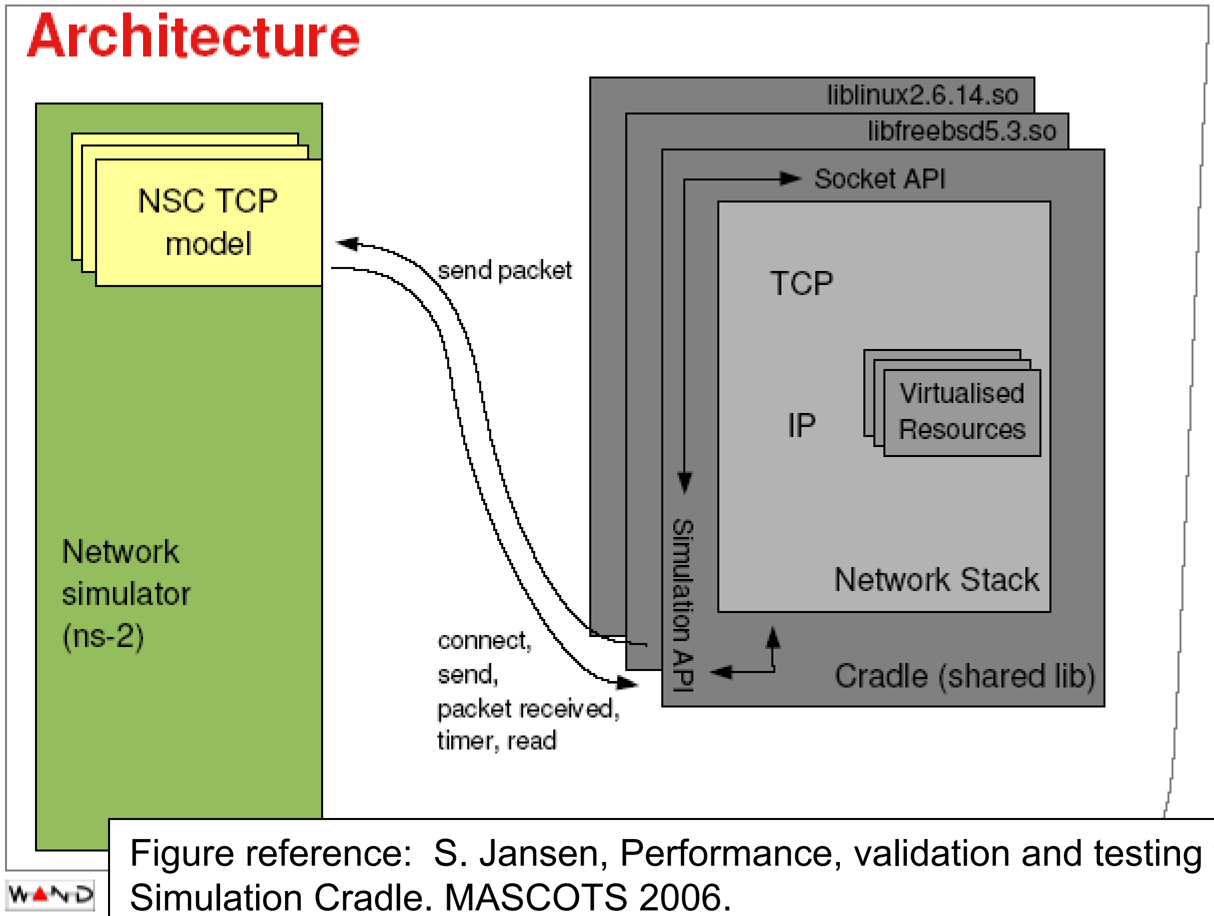
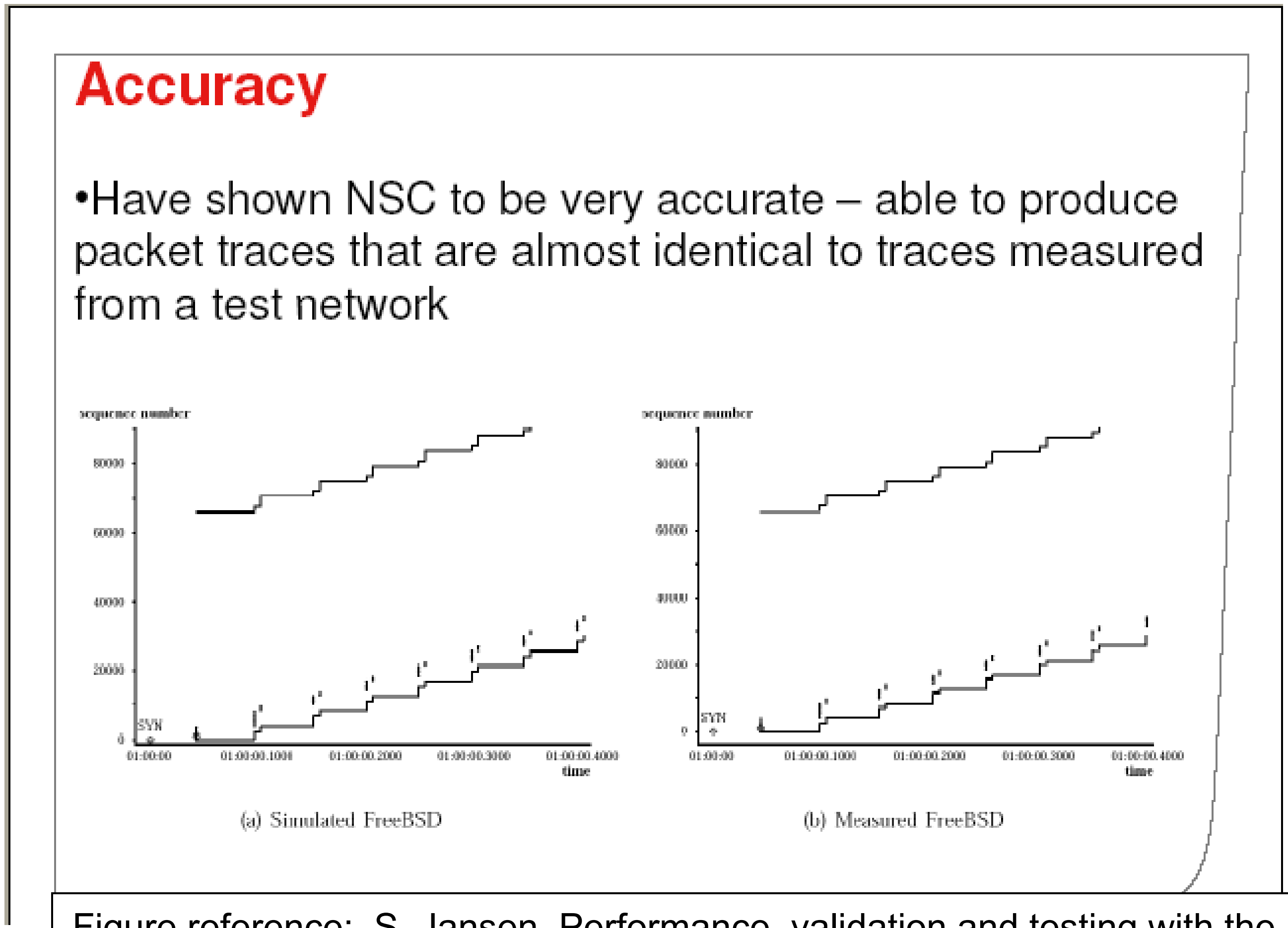


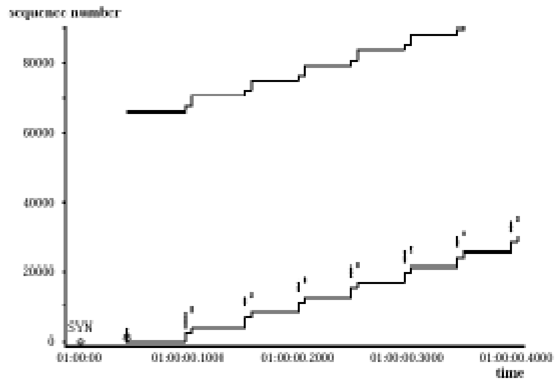
Figure reference: S. Jansen, Performance, validation and testing with the Network Simulation Cradle. MASCOTS 2006.

ns-3 simulation cradle

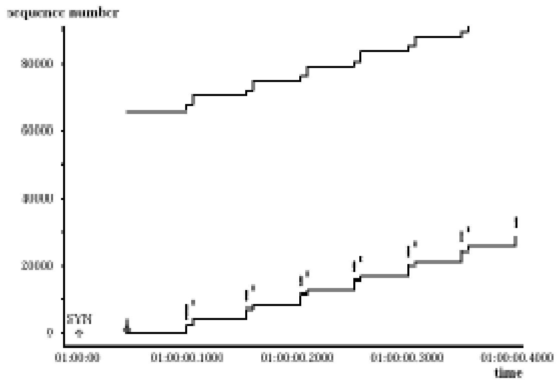


Accuracy

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



(a) Simulated FreeBSD



(b) Measured FreeBSD

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++

IPv4 rework

- The IP-related classes are undergoing rework (in repository ~tomh/ns-3-ip) for ns-3.5 release
  - Multiple IPv4 addresses per interface
  - Delegate IP forwarding logic to an IPv4Routing class
  - Align better with Linux interfaces and system architecture
  - Align with IPv6 work

current ns-3 routing model

classes Ipv4RoutingProtocol, Ipv4Route

- Each routing protocol maintains its own RIB --> no common FIB
- Routing protocols are registered with

```
AddRoutingProtocol (Ptr<> protocol,
  int16_t priority)
```

- Routes are looked up by querying each protocol for a route
  - `Ipv4L3Protocol::Lookup()`

Routing options so far

- Global routing
  - mainly for static topologies
  - point-to-point and CSMA links
- OLSR
  - dynamic routing
  - can handle wired and wireless topologies

Future plans: quagga routing

Support for a synchronous Posix socket API

- each Posix type and function is redefined in the simulator
- processes get their own private stack
  - somewhat like a lightweight virtual machine
- Example use case:
  - compile quagga with -fPIC option
  - load quagga binary with ns-3 Process API
- Benefits:
  - makes porting real world application code much easier
  - makes writing applications easier because the BSD socket API is faithfully followed
- see the “~mathieu/ns-3-simu” code repository

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));
```

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)
{
...
}
```

Review of sample program (cont.)

```
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 ();
```

Attributes

Tracing

Config store

ns-3 attribute system

Problem: Researchers want to know all of the values in effect in 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

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

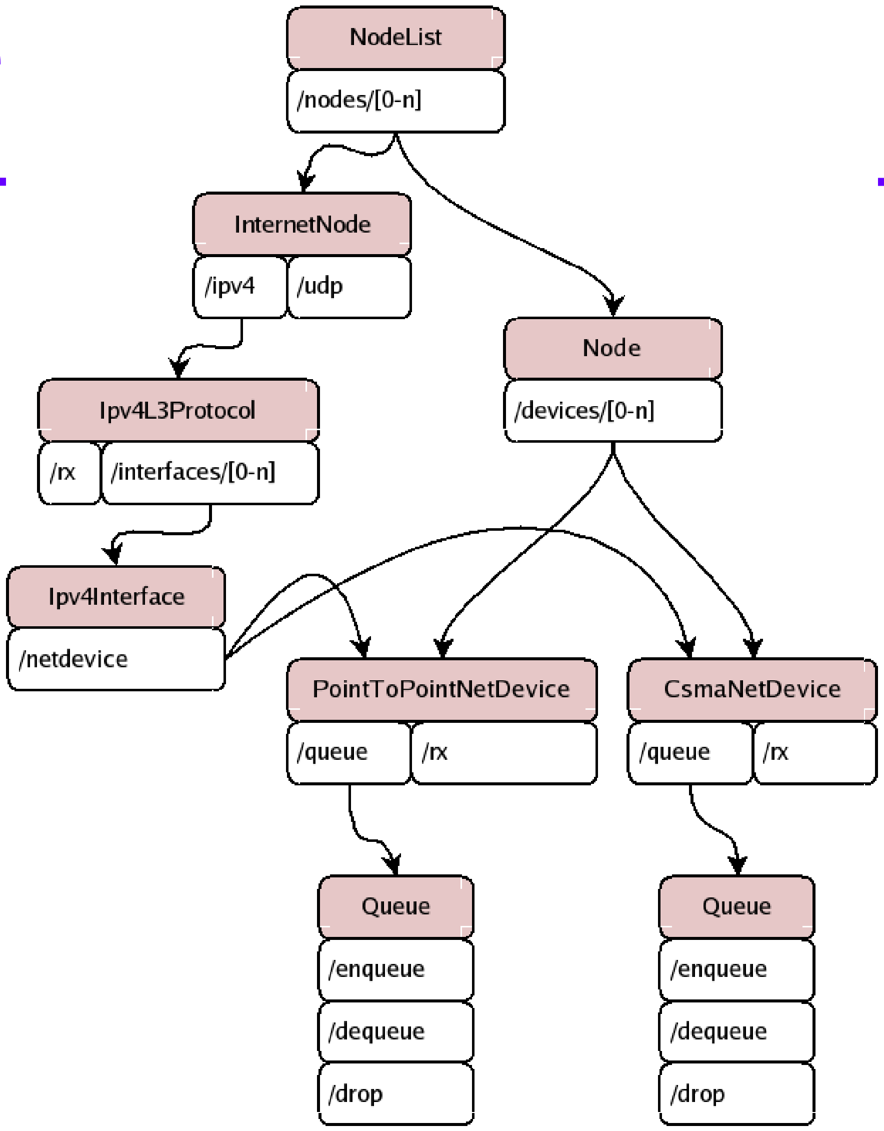
- The traditional C++ way:
  - export attributes as part of a class's public API
  - walk pointer chains (and iterators, when needed) to find what you need
  - use static variables for defaults
- The attribute system provides a more convenient API to the user to do these things

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



```
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::WifiPhy::TxGain”,
  DoubleValue (1.0));
```

- Syntax also supports string values:

```
Config::Set (“WifiPhy::TxGain”, StringValue
  (“1.0”));
```

Attribute   Value

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/Ph
  y/TxGain", DoubleValue(1.0));
```

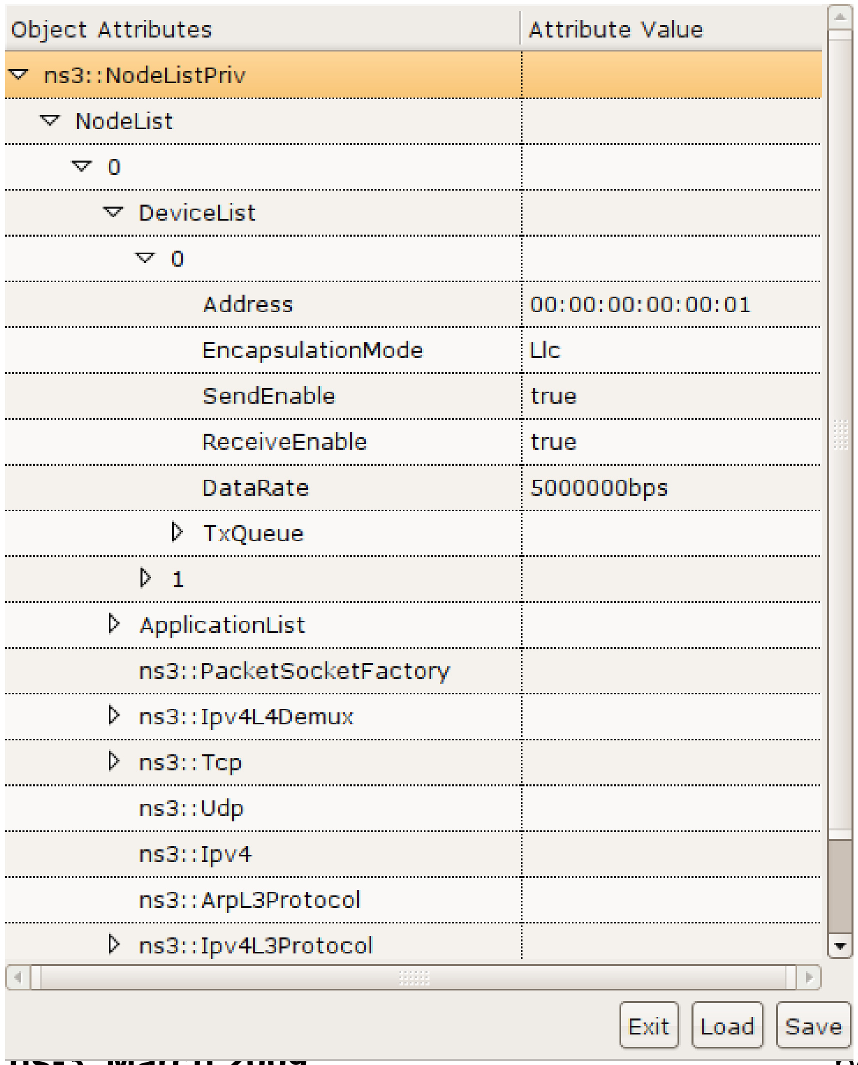
```
DoubleValue d; nodePtr->GetAttribute (
  "/NodeList/5/NetDevice/3/Phy/TxGain", v);
```

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

ns-3 attribute system

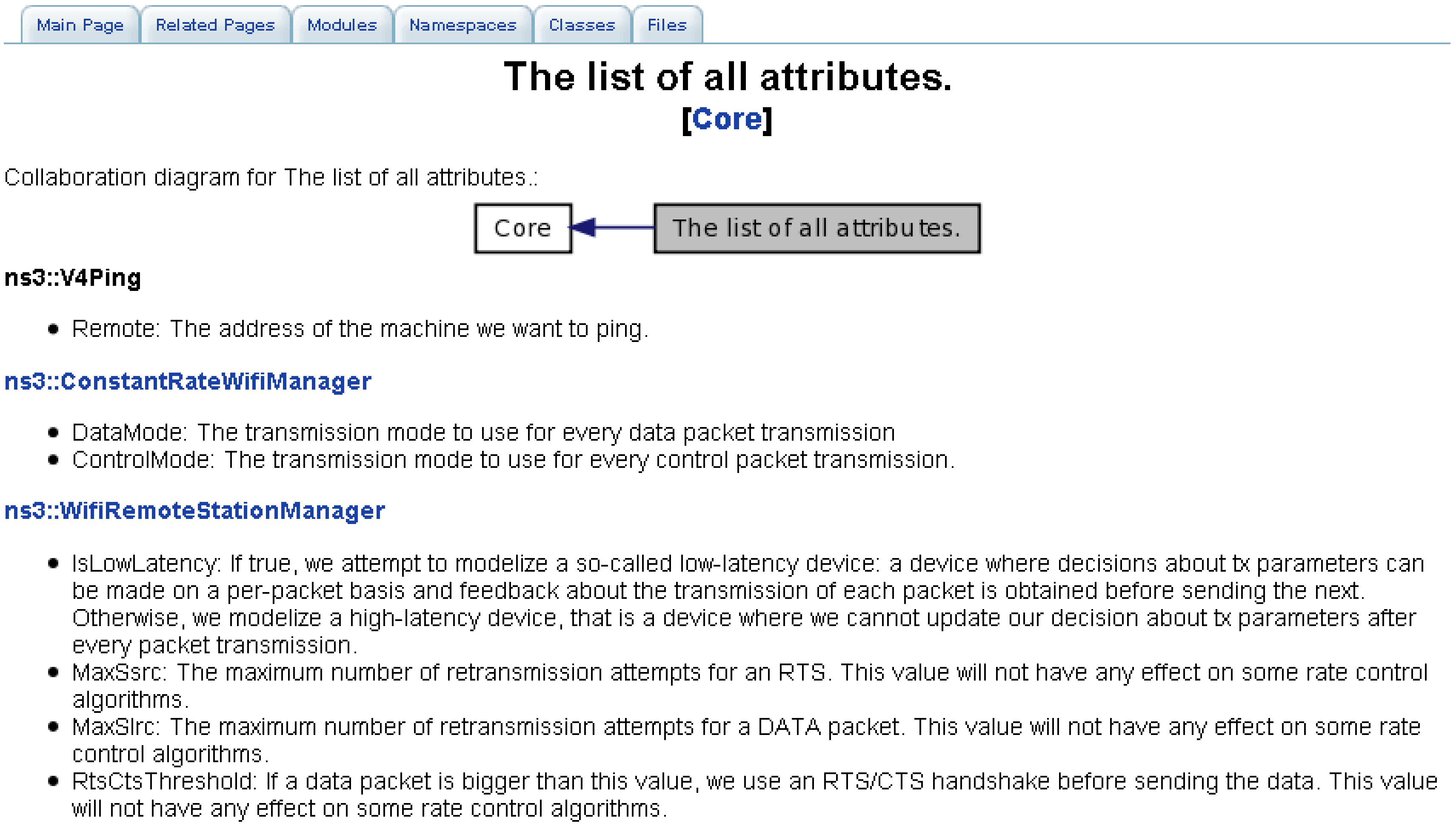
- Object attributes are organized and documented in the Doxygen

- Enables the construction of graphical configuration tools:



Object Attributes	Attribute Value
ns3::NodeListPriv	
NodeList	
0	
DeviceList	
0	
Address	00:00:00:00:00:01
EncapsulationMode	Llc
SendEnable	true
ReceiveEnable	true
DataRate	5000000bps
TxQueue	
1	
ApplicationList	
ns3::PacketSocketFactory	
ns3::Ipv4L4Demux	
ns3::Tcp	
ns3::Udp	
ns3::Ipv4	
ns3::ArpL3Protocol	
ns3::Ipv4L3Protocol	

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:

```
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"));
```

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
```

<u>Problem:</u> Tracing needs vary widely

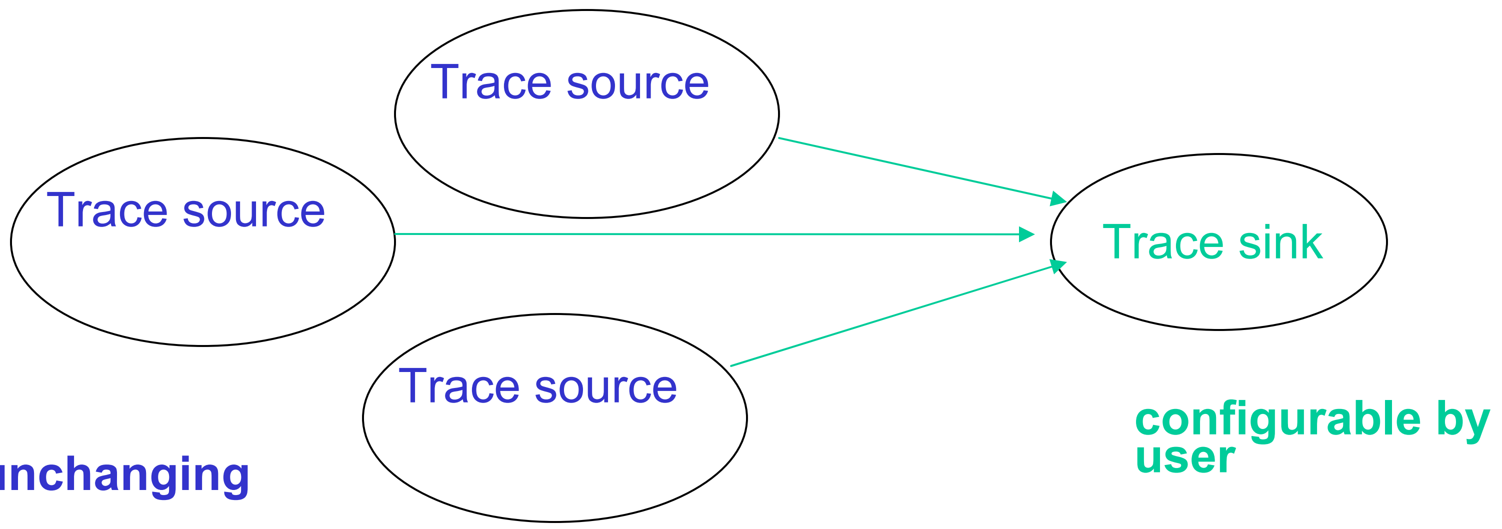
- would like to change tracing output without editing the core
- would like to support multiple outputs

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

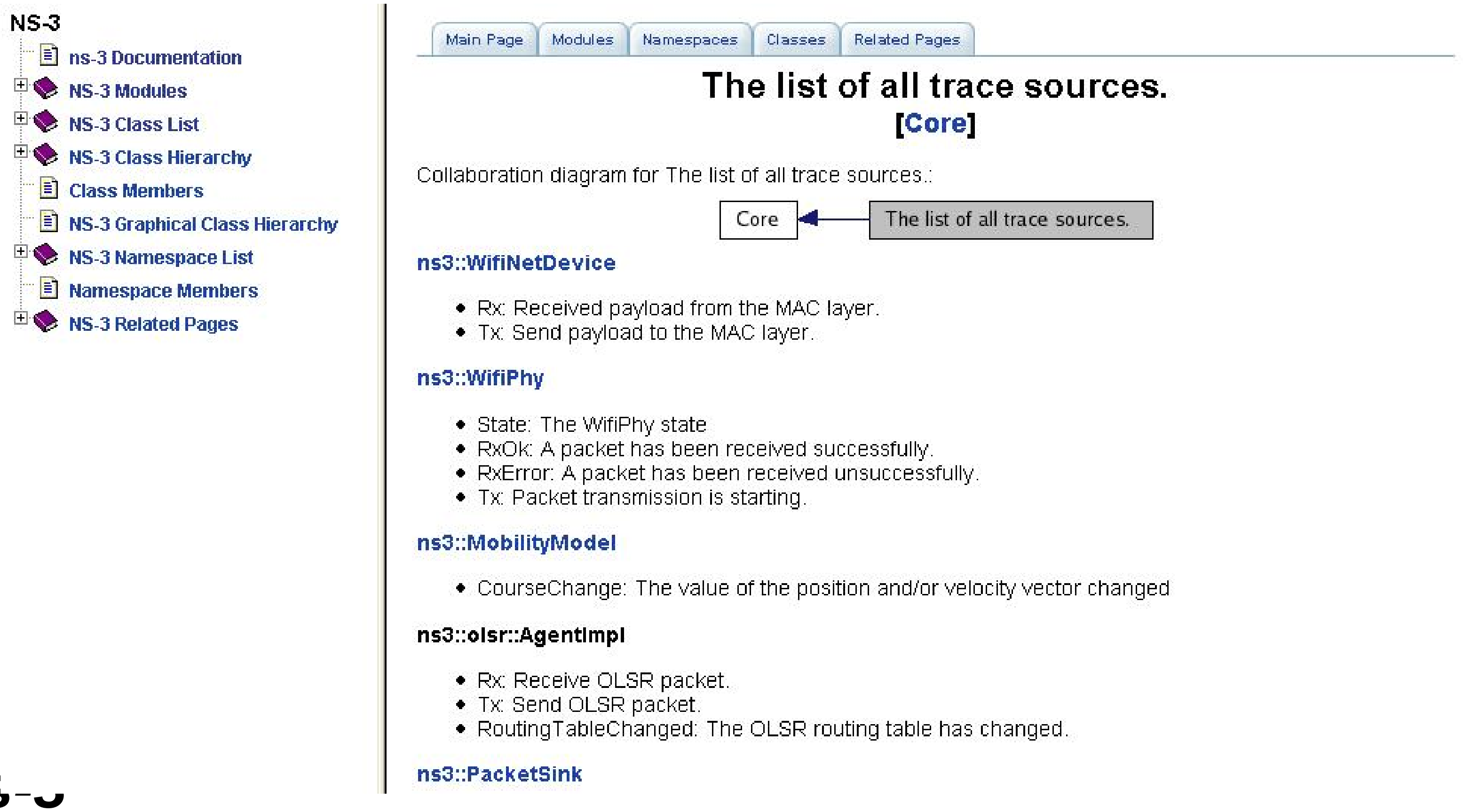
ns-3 solution: decouple trace sources from trace sinks



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



NS-3
- ns-3 Documentation
- NS-3 Modules
- NS-3 Class List
- NS-3 Class Hierarchy
- Class Members
- NS-3 Graphical Class Hierarchy
- NS-3 Namespace List
- Namespace Members
- NS-3 Related Pages

Main Page | Modules | Namespaces | Classes | Related Pages

The list of all trace sources.
[Core]

Collaboration diagram for 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
- RxOk: A packet has been received successfully.
- RxError: A packet has been received unsuccessfully.
- Tx: 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

```
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

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

Regular expression editing

Hook in a different trace sink

Asciitrace: under the hood

```
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

```
double MyFunc (int x, float y) {
  return double (x + y) / 2;
}
[...]
Callback<double, int, float> cb1;
cbl1 = MakeCallback (MyFunc);
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
```

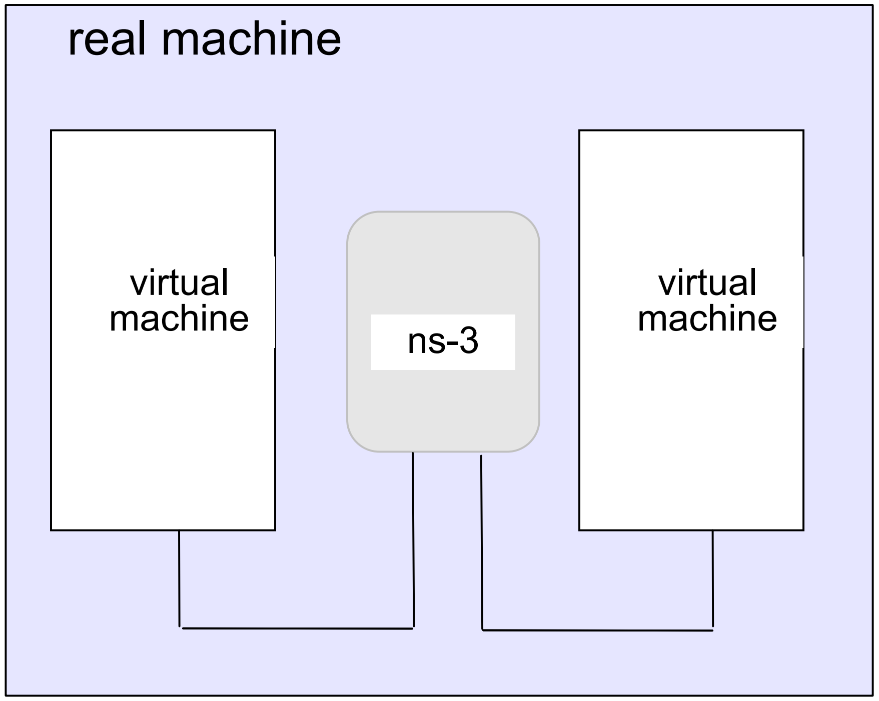
Emulation support

Support moving between simulation and testbeds or live systems

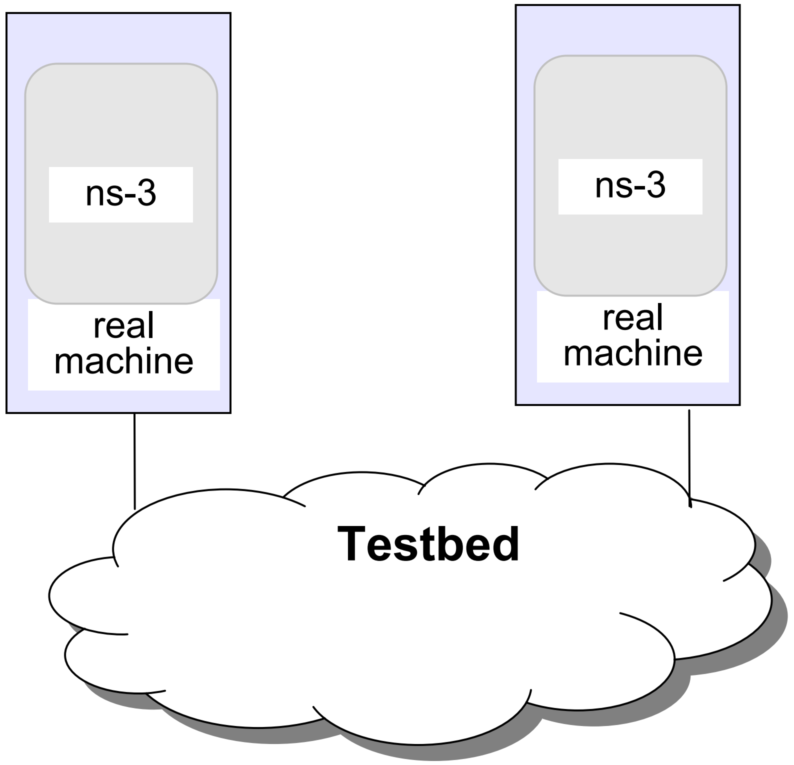
- A real-time scheduler, and support for two modes of emulation

```
GlobalValue::Bind (“SimulatorImplementationType”,
  StringValue (“ns3::RealTimeSimulatorImpl”));
```

ns-3 emulation modes



1) ns-3 interconnects real or virtual machines

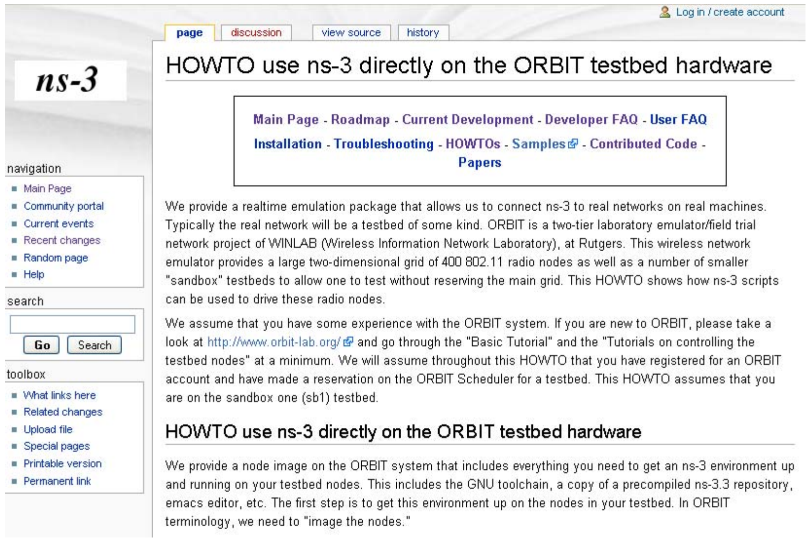


2) testbeds interconnect ns-3 stacks

Various hybrids of the above are possible

Example: ORBIT and ns-3

- Support for use of Rutgers WINLAB ORBIT radio grid



HOWTO use ns-3 directly on the ORBIT testbed hardware

Main Page - Roadmap - Current Development - Developer FAQ - User FAQ
Installation - Troubleshooting - HOWTOs - Samples - Contributed Code - Papers

We provide a realtime 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 400 802.11 radio nodes as well as a number of smaller "sandbox" testbeds to allow one to test without reserving 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 (sb1) testbed.

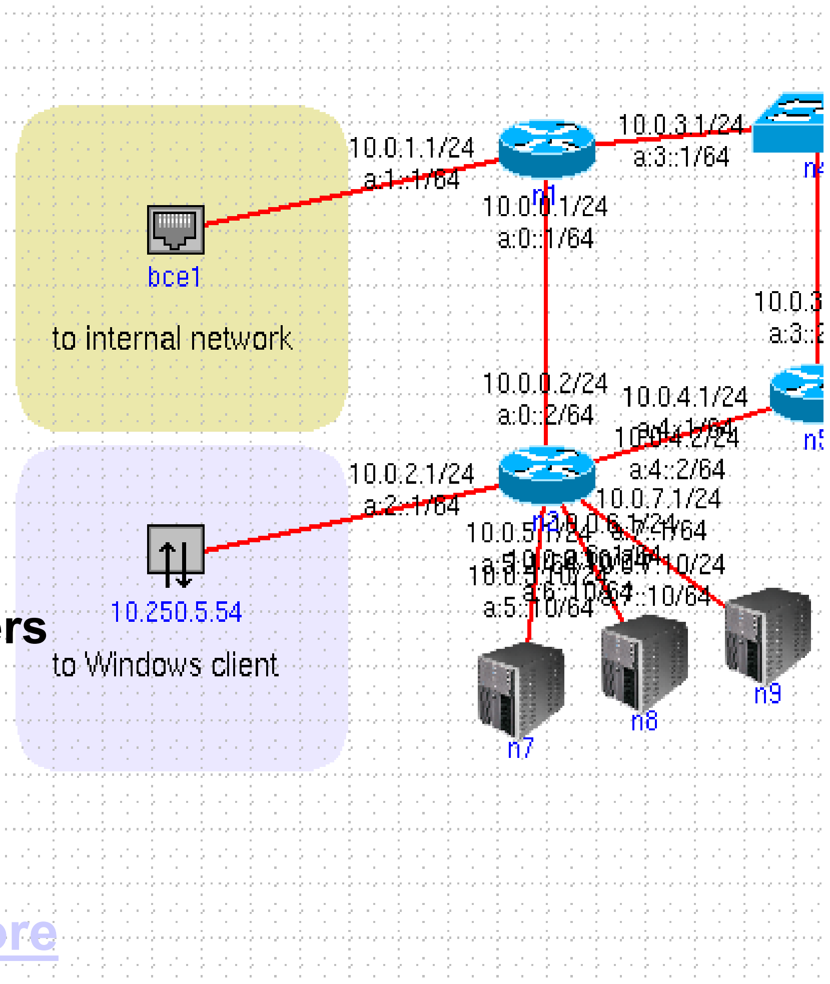
HOWTO use ns-3 directly on the ORBIT testbed hardware

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 this environment up on the nodes in your testbed. In ORBIT terminology, we need to "image the nodes."

example: CORE and ns-3

Scalable Network Emulator

- Network lab “in a box”
  - **Efficient and scalable**
  - **Easy-to-use GUI canvas**
- Kernel-level networking efficiency
  - **Reference passing packet sending**
- Runs real binary code
  - **No need to modify applications**
- Connects with real networks
  - **Hardware-in-the-loop**
  - **Distributed - runs on multiple servers**
  - **Virtual nodes process real packets**
- Fork of the IMUNES project
  - **University of Zagreb**
- Open Source
  - **http://cs.itd.nrl.navy.mil/work/core**



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

Visualization

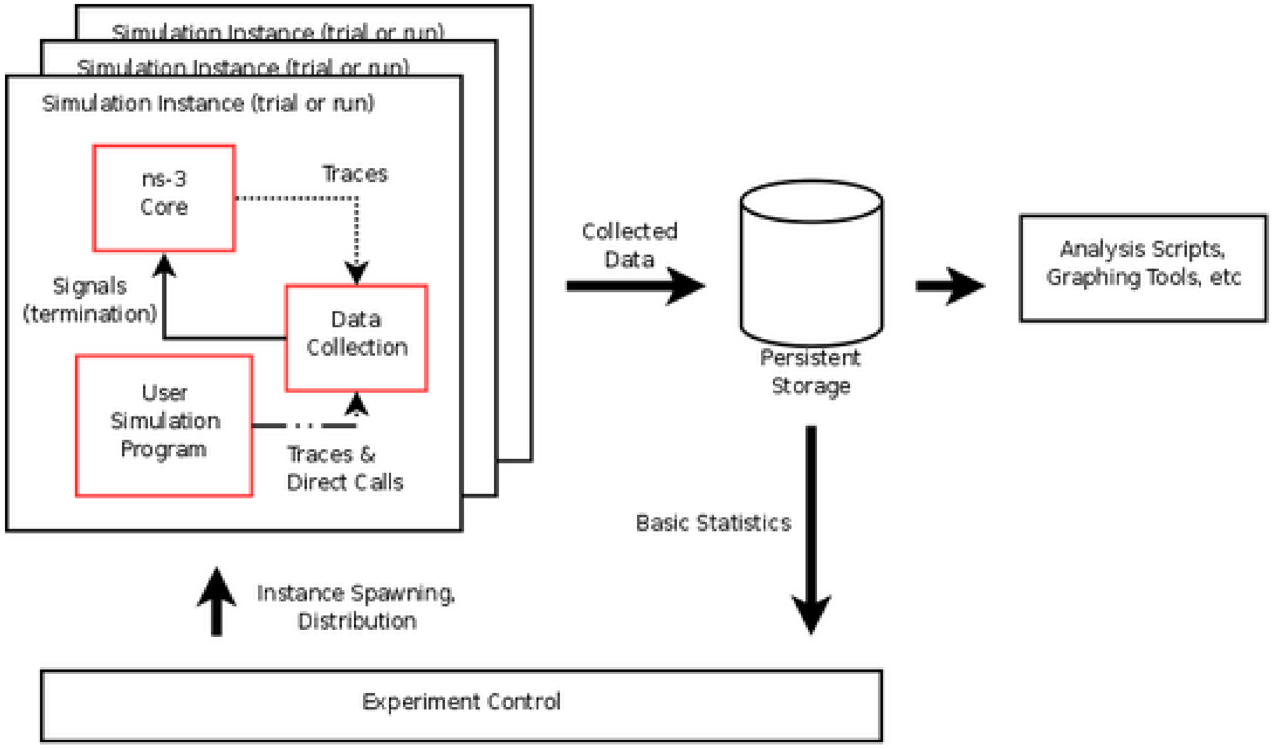
- Various projects in work to build animators and visualizers for ns-3
  - May provide a simulation implementation that allows for GUI interaction with the scheduler (e.g., pause)
- Examples:
  - Gustavo Carneiro pyviz (demoed earlier)
  - George Riley's NetAnim (demo to follow)
  - Hagen Paul Pfeifer's OpenGL animator
  - Colorado School of Mines iNSpect tool
  - Eugene Dedu, awk scripts for ns-3 and nam

Statistics framework

- Tracing system supports a statistical and data management framework
  - currently a contributed module
  - `src/contrib/stats; examples/stats`
- Features:
  - manage multiple independent runs of a scenario
  - marshal data into several output formats
    - including databases, with per-run metadata
  - hook into ns-3 trace sources
  - statistics objects can interact with simulator at run-time
    - e.g. stop simulation when counter reaches a value

statistics framework (cont.)

- Details at:
  - http://www.nsnam.org/wiki/index.php/Statistical_Framework_for_Network_Simulation


Simulation Instance (trial or run)
Simulation Instance (trial or run)
Simulation Instance (trial or run)
ns-3 Core
Traces
Signals (termination)
Data Collection
User Simulation Program
Traces & Direct Calls
Collected Data
Persistent Storage
Analysis Scripts, Graphing Tools, etc
Basic Statistics
Instance Spawning, Distribution
Experiment Control

Data Collection objects

- DataCollector
  - Provides framework for data collection
- DataCalculator
  - Connected to ns-3 trace sources via different techniques
- DataOutputInterface
  - Defines the output interface for the processed data

DataCollector

```
// Create a DataCollector object to hold information about this
 run.
DataCollector data;
data.DescribeRun(experiment,
                 strategy,
                 input,
                 runID);

// Add any information we wish to record about this run.
data.AddMetadata("author", "tjkopena");
```

DataCalculator

```
// This ... creates a counter to track how many frames
// are received.  Instead of our own glue function, this uses a
// method of an adapter class to connect a counter directly to the
// trace signal generated by the WiFi MAC.
Ptr<PacketCounterCalculator> totalRx =
  CreateObject<PacketCounterCalculator>();
totalRx->SetKey("wifi-rx-frames");
Config::Connect("/NodeList/1/DeviceList/*/$ns3::WifiNetDevice/Rx",
                MakeCallback(&PacketCounterCalculator::FrameUpdate,
                                  totalRx));
data.AddDataCalculator(totalRx);
```

- Other DataCalculators
  - PacketCounter
  - MinMaxAvgTotal
  - TimeMinMaxAvgTotal

DataOutputInterface

```
Simulation::Run ();
Simulation::Destroy ();
//---------------------------------------------------------------
//-- Generate statistics output.
//----------------------------------------------

// Pick an output writer based in the requested format.
Ptr<DataOutputInterface> output = 0;
if (format == "omnet") {
  NS_LOG_INFO("Creating omnet formatted data output.");
  output = CreateObject<OmnetDataOutput>();
} else if (format == "db") {
  #ifdef STATS_HAS_SQLITE3
    NS_LOG_INFO("Creating sqlite formatted data output.");
    output = CreateObject<SqliteDataOutput>();
  #endif
} else {
  NS_LOG_ERROR("Unknown output format " << format);
}

// Finally, have that writer interrogate the DataCollector and save
// the results.
if (output != 0)
  output->Output(data);
```

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 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

new in ns-3.4

- ns-3 simulations use a fixed seed and run number by default
  - default was random seeding prior to 3.4
- a class SeedManager used to edit seeds and run numbers

```
SeedManager::SetSeed (3); // Changes seed from default of 1 to 3
SeedManager::SetRun (7); // Changes run number from default of 1 to 7
// Now, create random variables
UniformVariable x(0,10);
ExponentialVariable y(2902);
...
```

Flexibility in changing these values

- Use NS_GLOBAL_VALUE environment variable

```
NS_GLOBAL_VALUE="RngRun=3" ./waf --run program-name
```

- Pass command-line argument

```
./waf --command-template="%s --RngRun=3" --run program-name
```

- Another way (outside of waf)

```
./build/optimized/scratch/program-name --RngRun=3
```

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
    - Need to be ***event-based*** rather than ***trace-based***
  - validation of models on testbeds
  - reuse of code
  - documented scripts and repositories
    - discussion topic for later today

Regressions

- ns-3-dev is checked nightly on multiple platforms
  - Linux gcc-4.x, Linux gcc-3.4, i386 and x86_64, OS X ppc
- ./waf --regression will run regression tests
  - a python script in regression/test directory will typically compare trace output with known good traces

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)

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

Acknowledgments

Thanks to:

- Gustavo Carneiro for tutorial content
- the core development team and research project leads
  - Raj Bhattacharjea, Gustavo Carneiro, Walid Dabbous, Craig Dowell, Joe Kopena, Mathieu Lacage (software lead), George Riley, Sumit Roy
- 2008 Google Summer of Code mentors and students
- many code authors and testers
- the ns-2 PIs and developers for creating ns-2 and for supporting ns-3 activities
- USC ISI for hosting project mailing lists