ns-3 tutorial

Presenter: Tom Henderson
University of Washington
SimuTools Conference
March, 2008

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

Outline

• Introduction to ns-3
• Reading ns-3 code
• Tweaking ns-3 code
• Extending ns-3 code

Acknowledgments

• Thanks to Mathieu Lacage and Craig Dowell for assembling the tutorial source code and materials
• Thanks to ns-3 development team!
• Tom Henderson is supported by NSF CNS-0551686 (University of Washington)

Assumptions

• Some familiarity with C++ programming language
• Some familiarity with Unix Network Programming (e.g., sockets)
• Some familiarity with discrete-event simulators

What is ns (or ns-2)?

• ns is a discrete-event network simulator for Internet systems
  – protocol design, multiple levels of abstraction
  – written in multiple languages (C++/Tcl)
• ns has a companion network animator called namin
  – hence, has been called the nsnam project

ns-3 is a research-oriented, discrete event simulator
ns-3 features

- open source licensing (GNU GPLv2) and development model
- Python scripts or C++ programs
- alignment with real systems (sockets, device driver interfaces)
- alignment with input/output standards (pcap traces, ns-2 mobility scripts)
- tested integration is a priority
- modular, documented core

ns-3 people

- NSF PIs:
  - Tom Henderson, Sumit Roy (University of Washington), George Riley (Georgia Tech.), Sally Floyd (ICIR)
- Associated Team: INRIA Sophia Antipolis, Planete group
  - Wald Dabbous, Mathieu Lacage (software lead)
- Developers: Raj Bhattacharjea, Gustavo Carneiro, Craig Dowell, Joseph Koprena, Emmanuelle Laprise

ns-3 relationship to ns-2

ns-3 is not an extension of ns-2
- does not have an OTcl API
- C++ wrapped by Python
- synthesis of yans, ns-2, GTNetS simulators, and new software
  - example ns-2 models so far: random variables, error models, OLSR
  - guts of simulator are completely replaced
  - new visualizers are in works

ns-3 status (March 2008)

ns-3 is in a pre-alpha state
- monthly development releases
- APIs being finalized
- emphasis has been on setting the architecture
- new users should expect rough edges
- many opportunities to work on the core models

ns-3 status (March 2008)

What others are already using ns-3 for:
- wifi-based simulations of OLSR and other MANET routing
- MANET routing (SMF and unicast protocols)
- OntoNet: Scalable Knowledge Based Networking* by Joe Koprena and Boon Thau Loo (UPenn)
**ns-3 roadmap (2008)**

**near term (through June)**
- finalize and release simulation core (April/May)
  - core APIs
- ns-3.1 complete release (June timeframe)
  - add Internet and Device models
  - add validation framework
  - some Higher-level topology/scenario APIs

**planned for later this year**
- emulation modes
- statistics
- support for real code
- additional ns-2 porting/integration
- distributed simulation
- visualization

---

**Resources**

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

Mailing list: nsnam-users@listinfra-developers

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

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


---

**Links to materials**

- Today's code
  - http://www.nsnam.org/tutorials/simutools08/ns-3-tutorial.tar.gz
- Tutorial slides:
  - PPT: http://www.nsnam.org/tutorials/simutools08/ns-3-tutorial.ppt
  - PDF: http://www.nsnam.org/tutorials/simutools08/ns-3-tutorial.pdf

---

**Outline**

- Introduction to ns-3
- Reading ns-3 code
- Tweaking ns-3 code
- Extending ns-3 code

---

**Questions so far?**
Reading ns-3 code

- Browsing the source code
- Conceptual overview
- Script walkthrough

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 (python scripts)

Browse the source

Pause presentation to browse source code
http://www.nsnam.org/tutorials/simutode08/ns-3-tutorial.tar.gz

Doxygen documentation

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

Pause presentation to browse Doxygen
http://www.nsnam.org/doxygen/index.html

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, CMak or Ant
  - http://code.google.com/p/waf/

Pause presentation to build with waf

waf key concepts

- For those familiar with autotools:
  - configure -> ./waf [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)
The basic model

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 architectured for multiple interfaces

Node basics
Two key abstractions are maintained:
1) applications use an (asynchronous, for now) sockets API
2) the boundary between IP and layer 2 mimics the boundary at the device-independent sublayer in Linux
i.e., Linux Packet Sockets

ns-3 Packets
- each network packet contains a byte buffer, a list of tags, and metadata
  - buffer: bit-by-bit (serialized) representation of headers and trailers
  - tags: set of arbitrary, user-provided data structures (e.g., per-packet cross-layer messages, or flow identifiers)
  - metadata: describes types of headers and trailers that have been serialized
* optional – disabled by default
**ns-3 Packets**

- to add a new header, subclass from Header, and write your Serialize() and Deserialize() methods
- how bits get written to/from the Buffer
- Similar for Packet Tags
- Packet Buffer implements a (transparent) copy-on-write implementation

---

**Example: UDP header**

```cpp
class UDpHeader : public Header {
    public:
        void SetDestination (uint16_t port);
        ...
        void Serialize (Buffer::Iterator start) const;
    private:
        uint16_t m_sourcePort;
        uint16_t m_destinationPort;
        uint16_t m_payloadSize;
        uint16_t m_initCheckSum;
    }
```

---

**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

---

**Sample script walkthrough**

*Walk through mixed-wireless.cc*
Outline

- Introduction to ns-3
- Reading ns-3 code
- Tweaking ns-3 code
- Extending ns-3 code

ns-logging

- ns-3 has a built-in logging facility to stderr
- Features:
  - can be driven from shell environment variables
  - Multiple log levels like syslog
  - Function and call argument tracing
- Intended for debugging, but can be abused to provide tracing
- we do not guarantee that format is unchanging

ns-logging example

- NS_LOG_UNCOND();
- NS_LOG environment variable
- per-source-file logging
- log levels
- example scripts

attributes and tracing

- Next, we would like to talk about attributes
  (default values, settable and gettable values) and tracing
- To understand this, we’ll introduce the ns-3 Object system
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

Disclaimer: This is not all main-line ns-3 code—parts are in a proposal in the mathlunch-5-param repository

Object aggregation use case

- You can aggregate objects to one another at run-time
  - Avoids the need to modify a base class to provide pointers to all possible connected objects
- Object aggregation is planned to be the main way to create new Node types (rather than subclassing Node)

Object aggregation example

```c
void
WifChannel::Send (Ptr<WifPhy> sender, Ptr<Packet>
packet, ...)
{
  Ptr<MobilityModel> senderMobility = Cs
  Ptr<MobilityModel> receiveMobility = Ss
  ...
  senderMobility = sender->GetNode () ->
  GetObject<MobilityModel>() ;
}
```

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. TopSocket::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
The traditional C++ way

```
// C++ code

// 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. WifiPhy::TxGain

// A Config class allows users to manipulate the attributes
```

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. WifiPhy::TxGain
- A Config class allows users to manipulate the attributes

Navigating the attributes using paths

- Examples:
  - Nodes with NodeIds 1, 3, 4, 5, 8, 9, 10, 11:
    "NodeList/[0-9][8-11]"
  - Udp4Protocol object instance aggregated to matching nodes:
    "Udp4Protocol"
  - EndPoints which match the SrcPort=1025 specification:
    "EndPoints/SrcPort=1025"

What users will do

- e.g.: Set a default initial value for a variable
- (Note: this replaces DefaultValue::Bind())
- `Config::Set("WifiPhy::ThGain", Double(1.0));`
- Syntax also supports string values:
- `Config::Set("WifiPhy::ThGain", "1.0");`

What users will see

- 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::GetAttribute("NodeList/3/DeviceList/3/Phy/ThGain", Double(1.0));`
- `Config::GetAttribute("NodeList/3/DeviceList/3/Phy/ThGain");`
- Users can get Ptrs to instances also, and Ptrs to trace sources, in the same way

CreateObject<>();

- `CreateObject<>()` is a wrapper for operator new.
- `ns3::Object` objects must be created on the heap using `CreateObject<>()` (which returns a smart pointer); e.g.
- `Ptr<Node> mNode = CreateObject<Node>();`
Create<>():

- What is Create<>()?
- Create<> provides some smart pointer help for objects that use ns3::Ptr<> but that do not inherit from Object.
- Principally, class ns3::Packet

```cpp
Ptr<Packet> p = Create<Packet>(data, size);
```

Non-default constructors

- The attribute system allows you to also pass them through the CreateObject<> constructor.
- This provides a generic non-default constructor for users (any combination of parameters), e.g.:

```cpp
Ptr<WifiPhy> phy = CreateObject<WifiPhy>()
    "Tunable", Double(1.0));
```

How is all this implemented (overview)

Also part of Object: smart pointers

- ns-3 uses reference-counting smart pointers at its APIs to limit memory leaks
  - Or "pass by value" or "pass by reference to const" where appropriate
- A "smart pointer" behaves like a normal pointer (syntax) but does not lose memory when reference count goes to zero
- Use them like built-in pointers:

```cpp
Ptr<MyClass> p = CreateObject<MyClass>();
p->method();
```

A real Typedef example

Statements you should understand now

```cpp
Ptr<ns3::Channel> ch;
ns3::Object myObject;
```
Tracing model

- Tracing is a structured form of simulation output
- tracing format should be relatively static across simulator releases
- Example (from ns-2)
  + 1.84373 0 2 obe 210 -------- 0 0.0 3.1 225 610
  - 1.84373 0 2 obe 210 -------- 0 0.0 3.1 225 610
  ± 1.84373 0 2 obe 210 -------- 1 3.0 1.0 195 680
  + 1.8456 0 2 tcp 1000 -------- 2 0.1 3.2 302 611
- Needs vary widely

Crude tracing

```cpp
#include <iostream>
...
int main ()
{
...
  std::cout << "The value of x is " << x << std::endl;
...
}
```

slightly less crude

```cpp
#include <iostream>
...
int main ()
{
...
  NS_LOG_UNCOND ("The value of x is " << x);
...
}
```

Simple ns-3 tracing

- these are wrapper functions/classes
- see examples/mixed-wireless.cc
```cpp
#include "ns3/ns3.h"

AcciTrace acciTrace ("mixed-wireless.tr");
acciTrace.TraceAllQueues ();
acciTrace.TraceAllNetDeviceNs ();
```

Simple ns-3 tracing (pcap version)

- these are wrapper functions/classes
- see examples/mixed-wireless.cc
```cpp
#include "ns3/ns3.h"

PcapTrace pcaptrace ("mixed-wireless.pcap");
pcaptrace.TraceAll ();
```

ns-3 tracing model (revisit)

- Fundamental #1: decouple trace sources from trace sinks
- Fundamental #2: prefer standard trace outputs for built-in traces

Diagram:
- Trace source
- Trace sink
- Trace source
- Trace source
- Trace sink
- Trace sink
- Trace sink
- Trace source
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

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

```
// Also configure user topology traces: each interface will be traced
// and can be read by the 'toplogy-' named proc - "-d" option to
// display topology correctly
progTrace 구성("topology-trace.pgm");
```

Mid-level of tracing

```
mod
  progSource("source1", "source2");
```

Lowest-level of tracing

```
progSource("source1", "source2");
Hook in a different trace sink
```

AsciiTrace: under the hood

```
mod
  ProcRate("source1", "source2");
```

Low-level: Add trace sources to the tracing namespace
Statistics  
- Avoid large trace files  
- Full statistics support planned for later in 2008  
- Reuse tracing framework  
- One similar approach: ns-2-measure project  
  - Static “Stat” object that collects samples of variables  
  - based on explicit function calls inserted into the code  
  - Graphical front end, and framework for replicating simulation runs

Design patterns for topology scripts

Design approaches
- Use simple helper functions with attributes  
- Use reusable “frameworks”

Note: This area of our API is under discussion; feedback wanted

Helper Objects

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

setup backbone

- NodeContainer backbone;  
  backbone.Create ();  
- MobilityHelper mobility;  
  mobility.SetPositionAllocator ("RandomPositionAllocator", "Min", 
  "Max", 0.0);  
  mobility.SetMobilityModel ("RandomWaypointMobilityModel");  
- MobilityLayer backbone;  
- WTHelper wifi;  
  wifi.SetEncryption ("WEP64");  
- wifi.SetChannel ("1", "2", "3", "4", "5", "6", "7");  
- wifi.SetDefaultDeviceManager ("ConstantStaDeviceManager", 
  "RandomWaypointMobilityModel", "SetPositionAllocator");  
- wifi.SetDefaultIfaceConfig ("inet", 
  "Config" ("inet", "device", "ifname", "br0"));
- NetDeviceContainer backbone = wifi.Build (backbone, channel);
setup wifi subnets

for (i = 0; i < 32; i++)
    NodeContiner stuff;
    stuff.xname (i);
    stuff.yname (i);
    stuff.zname (i);
    stuff.wifi_snatch (i);
    stuff.wifi_speed (i);
    stuff.wifi_power (i);
   _stuff.wifi_isgood (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
    stuff.wifi_power (i);
Frameworks

• Idea: Can we write the same flow of operations once, but delegate them to a Manager?
• The Manager implements the functions
• The functions are virtual
• Users wishing to specialize them can override them as needed

Outline

• Introduction to ns-3
• Reading ns-3 code
• Tweaking ns-3 code
• Extending ns-3 code

How do simulator objects fit together?

• ns-3 objects are C++ objects
  • can be subclassed
• ns-3 Objects support aggregation

Aside: C++ templates

• templates allow a programmer to write one version of code that is applicable over multiple types
• templates are declared, defined and used
• Declaration:
  • template <typename T> T Add (T first, T second);
  • T Add (T first, T second);
  • might eventually become
  • int Add (int first, int second);

Aside: C++ templates

• Definition:
  template <typename T>
  T Add (T first, T second) {
    return first + second;
  }
• Usage:
  int x, y, z;
  z = Add<int> (x, y);
Classes may also be templated

- **Declaration:**
  ```cpp
template <typename T>
  class MYClass;

  void Push (T data);
  T Pop (void);
  ```

- **Definition:**
  ```cpp
template <typename T> void MYFunction1::Push (T data)
  ```

- **Usage:**
  ```cpp
  MYStructure->stack.
  start.Push (x);
  y = stack.Pop ();
  ```

### Scheduler and callbacks

- Let's look at samples/main-simulator.cc
- Schedules a single event, then exits

```cpp
let main (int argc, char *argv[])
```  
```cpp
  parseModel;
  simulator->schedule (seconds (0.0), example_function, sendet);  
  simulator->run ();  
  simulator->stop ();  
```  

### ns-3 callbacks

- Class template Callback<> implements the functor design pattern
- Callbacks are like function pointers, but more type-safe
- `static double Gnome (double a, double b) {}`
  ```cpp
  Callback<double, double, double> one;
  ```
- Bind a function with a matching signature to a callback
- `one = MakeCallback (Gnome);`
- `double returnone = one (10.0, 20.0);`

### Path of a packet (send)

1. An application sends a packet, is added to the send queue
2. A packet is transferred to a module
3. A packet is transferred to a module
4. A packet is transferred to a module
5. A packet is transferred to a module
6. A packet is transferred to a module

### Path of a packet (receive)

1. A packet is received

### current ns-3 routing model

- Classes `ipv4RoutingProtocol, ipv4Route`
- Each routing protocol maintains its own RIB `->` no common FIB
- Routing protocols are registered with
  ```cpp
  AddRoutingProtocol (Ftro* protocol, int16_t priority)
  ```
- Routes are looked up by querying each protocol for a route
  ```cpp
  - ipv4L3Protocol::Lookup()
  ```
Writing new ns-3 models

1) Define your requirements
   - reusability
   - dependencies
   - functionality
2) API review
   - Provide sample header file for API review
   - gather feedback from the ns-developers list

3) Create a non-functional skeleton
   - review coding style
   - decide which compilation unit it resides in
   - add to waf
   - build with body ifdefed out
   - copyright and headers
   - initial doxygen

4) Build a skeleton
   - header include guards
   - namespace ns3
   - constructor, empty function prototypes
   - key variables
   - Object/Typed code
   - write small test program
   - start a unit test

5) Build core functions and unit tests
   - use of logging, and asserts
6) Plumb into other modules, if needed
7) Post for review on developers list
8) Resolve comments and merge

Porting from ns-2

- Objects can be ported from ns-2 (or other simulators)
- Make sure licensing is compatible
- Example:
  - ns-2: queue/ermode/queue.c
  - ns-3: src/common/error-model.cc

Validation

- Can you trust ns-3 simulations?
  - Can you trust any simulation?
  - Cirus is on the researcher to verify results
- ns-3 strategies:
  - Open source benefits
  - Validation of models on testbeds
  - Reuse of code
  - Unit tests
  - Event driven validation tests
Walk through examples (time permitting)

- Beyond simple simulation scenarios
- Add a new type of MAC+PHY:
  - subclass NetDevice and a Channel
  - Add new types of transport layers:
  - subclass Node and Socket
  - subclass IP, class to implement per node level forwarding table and IP
  - interface configuration
  - for example, the Linux TCP stack could be easily integrated into a new type of node, LinuxNode with a LinuxTapSocket
- Add a new type of traffic generation and analysis:
  - subclass Application

ns-3 goals for emulation

1) ns-3 interconnects virtual machines
2) testbeds interconnected ns-3 stacks

Summary

- ns-3 is an emerging simulator to replace ns-2
- Consider ns-3 if you are interested in:
  - Open source and collaboration
  - More faithful representations of real computers and the Internet
  - Integration with testbeds
  - A powerful low-level API
  - Python scripting
- ns-3 needs you!

Proposed Google Summer of Code projects

- Performance Evaluation and Optimization
- Linux Kernel Network Stack Integration
- Parallel Simulations
- GUI Development
- Real World Code Integration

Resources

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

Mailing list:
nsnam@cs.uiuc.edu/mailman/listinfo/developers

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