ns-3 tutorial

Presenter: Tom Henderson
University of Washington

Simutools Conference
March, 2008

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)

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

Outline

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

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++/OTcl)
- *ns* has a companion network animator called *nam*
  - 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)
- testbed integration is a priority
- modular, documented core

ns-3 models

	Existing core ns-2 capability	Existing ns-3
Applications	ping, vat, telnet, FTP, multicast FTP, HTTP, probabilistic and trace-driven traffic generators, webcache	OnOffApplication, asynchronous sockets API, packet sockets
Transport layer	TCP (many variants), UDP, SCTP, XCP, TFRC, RAP, RTP Multicast: PGM, SRM, RLM, PLM	UDP, TCP
Network layer	Unicast: IP, MobileIP, generic dist. vector and link state, IPinIP, source routing, Nixvector Multicast: SRM, generic centralized MANET: AODV, DSR, DSDV, TORA, IMEP	Unicast: IPv4, global static routing Multicast: static routing MANET: OLSR
Link layer	ARP, HDLC, GAF, MPLS, LDP, Diffserv Queueing: DropTail, RED, RIO, WFQ, SRR, Semantic Packet Queue, REM, Priority, VQ MACs: CSMA, 802.11b, 802.15.4 (WPAN), satellite Aloha	PointToPoint, CSMA, 802.11 MAC low and high and rate control algorithms
Physical layer	TwoWay, Shadowing, OmniAntennas, EnergyModel, Satellite Repeater	802.11a, Friis propagation loss model, log distance propagation loss model, basic wired (loss, delay)
Support	Random number generators, tracing, monitors, mathematical support, test suite, animation (nam), error models	Random number generators, tracing, unit tests, logging, callbacks, mobility visualizer, error models

Project focus has been on the software core, to date

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
  - Walid Dabbous, Mathieu Lacage (software lead)
- **Developers:** Raj Bhattacharjea, Gustavo Carneiro, Craig Dowell, Joseph Kopena, 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 Kopena 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

ns-3 roadmap (2008)

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:

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

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

Questions so far?

Outline

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

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

```
tomh@A3803721 ~/home/ns-3-dev
$ ls
AUTHORS   README          VERSION   examples   samples   tutorial   waf       wscript
LICENSE   RELEASE_NOTES   doc       ns3        src       utils      waf.bat
```

Pause presentation to browse source code

http://www.nsnam.org/tutorials/simutools08/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, CMake or Ant
  - http://code.google.com/p/waf/

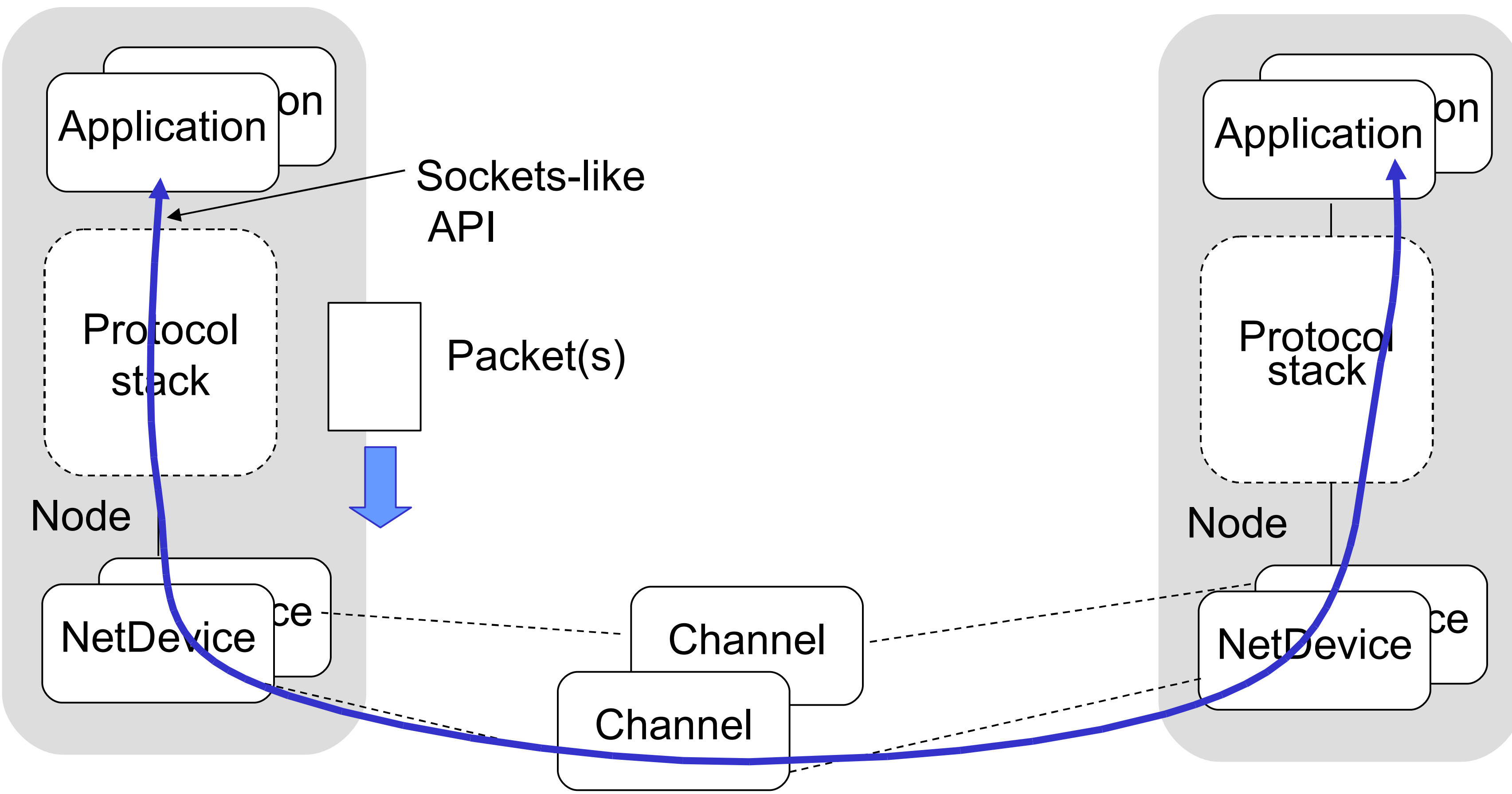
Pause presentation to build with waf

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)

The basic model



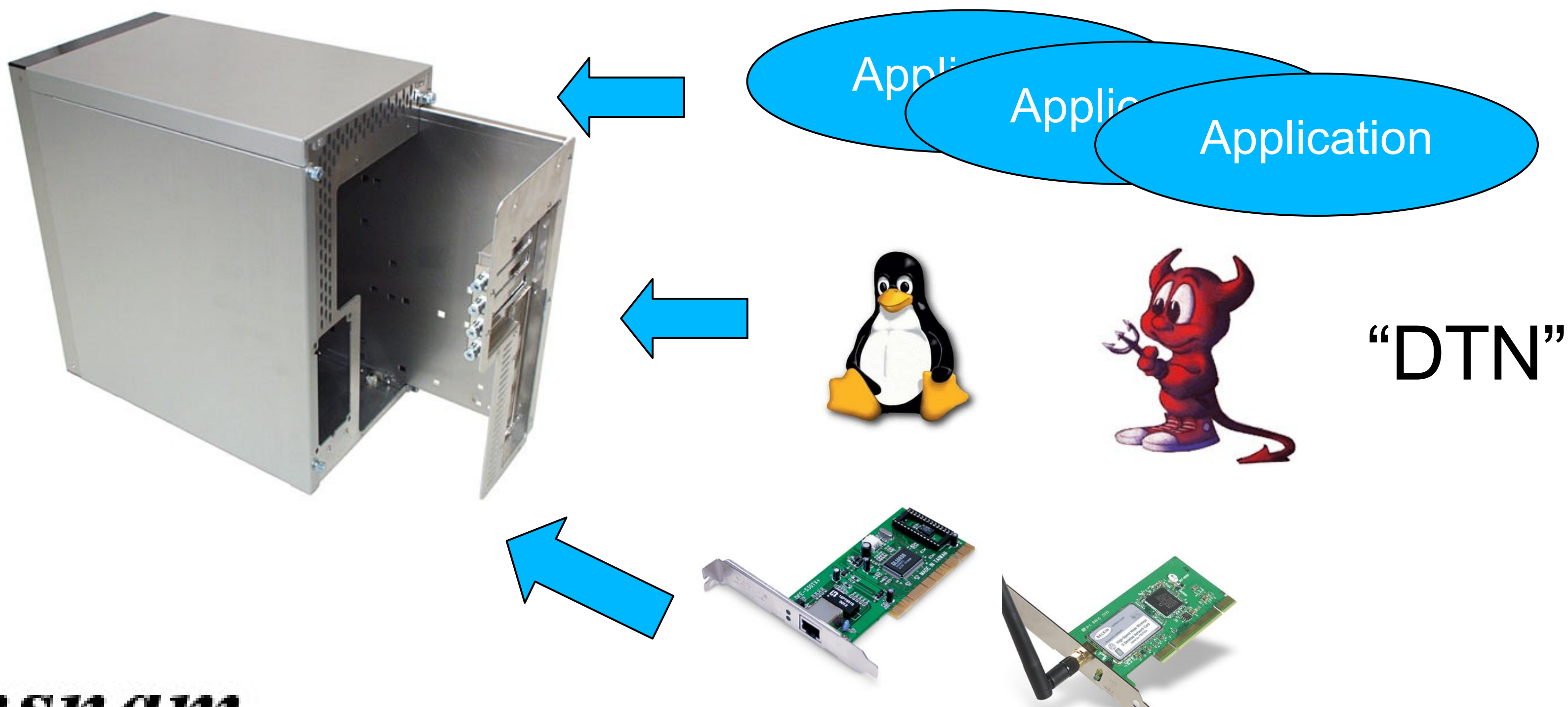
Fundamentals

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

Nodes contain Applications, “stacks”, and NetDevices

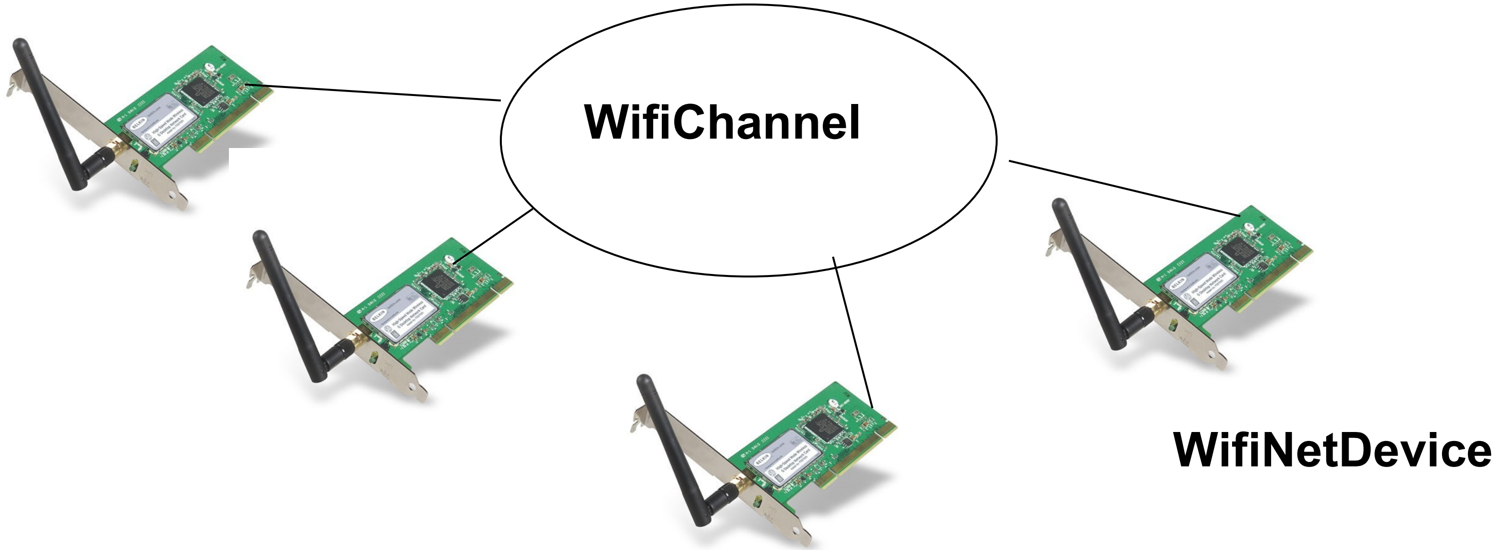
Node basics

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

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 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

```
class UdpHeader : public Header
{
public:
  void SetDestination (uint16_t port);
  ...
  void Serialize (Buffer::Iterator start) const;
  uint32_t Deserialize (Buffer::Iterator start);
private:
  uint16_t m_sourcePort;
  uint16_t m_destinationPort;
  uint16_t m_payloadSize;
  uint16_t m_initialChecksum;
}
```

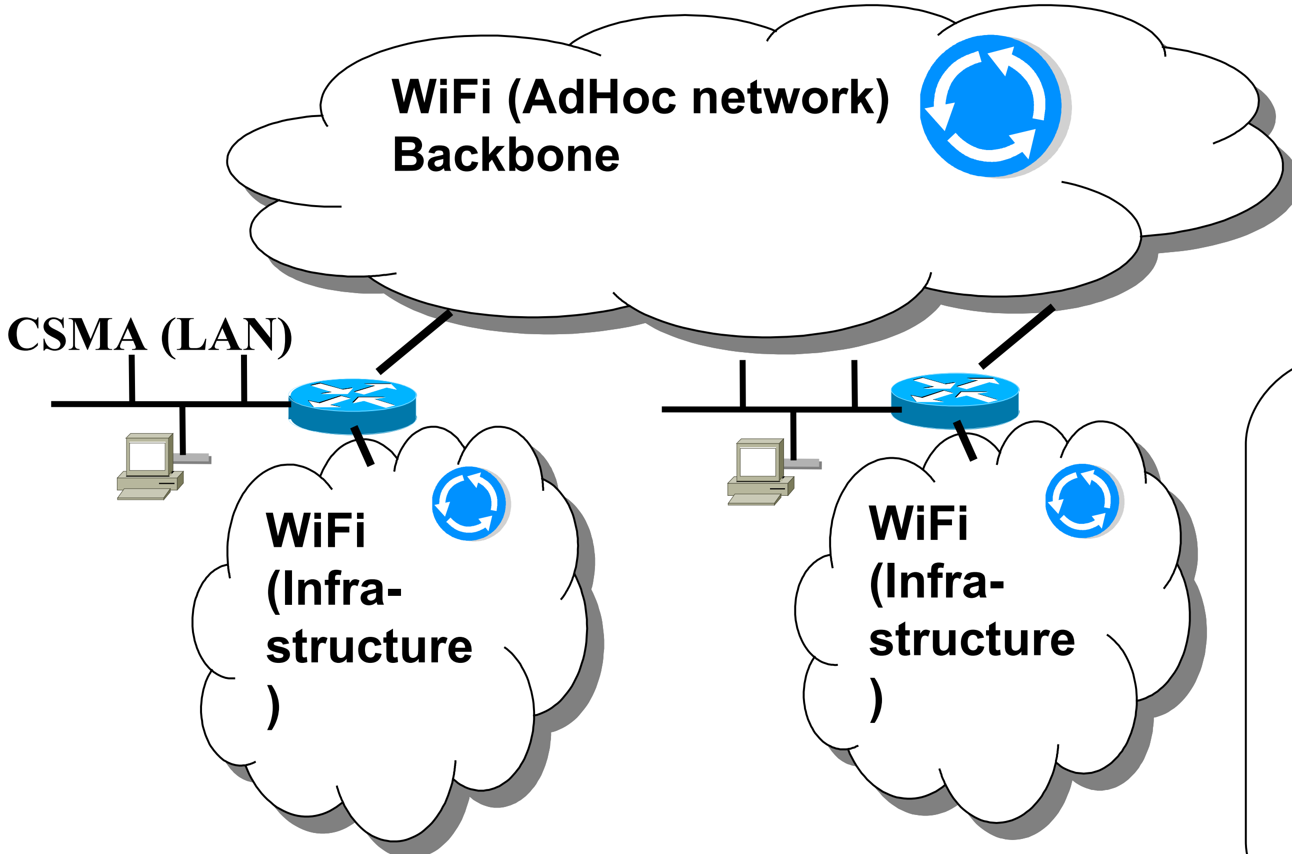
example: UDP header

```
void
UdpHeader::Serialize (Buffer::Iterator start) const
{
  Buffer::Iterator i = start;
  i.WriteHtonU16 (m_sourcePort);
  i.WriteHtonU16 (m_destinationPort);
  i.WriteHtonU16 (m_payloadSize + GetSerializedSize ());
  i.WriteU16 (0);
  if (m_calcChecksum)
    {
     uint16_t checksum = Ipv4ChecksumCalculate (...);
     i.WriteU16 (checksum);
    }
}
```

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



Hierarchical mobility
1) AdHoc movement in backbone
2) Local movement relative to Access Point

Applications: UDP flow,
Routing: OLSR on backbone

Sample script walkthrough

Walk through mixed-wireless.cc

(aside) similar looking code in Python

```
import sys
import ns3 as ns


def main():
## Set up some default values for the simulation.  Use the Bind()
## technique to tell the system what subclass of Queue to use,
## and what the queue limit is
## The below Bind command tells the queue factory which class to
## instantiate, when the queue factory is invoked in the topology code

ns.DefaultValue.Bind("Queue", "DropTailQueue")
ns.DefaultValue.Bind("OnOffApplicationPacketSize", "210")
ns.DefaultValue.Bind("OnOffApplicationDataRate", "448kb/s")

ns.CommandLine.Parse(sys.argv)

## Here, we will explicitly create four nodes.  In more sophisticated
## topologies, we could configure a node factory.

n0 = ns.InternetNode()
n1 = ns.InternetNode()
n2 = ns.InternetNode()
n3 = ns.InternetNode()
...
```

examples/ directory

- examples/ contains other scripts with similar themes

```
$ ls
csma-broadcast.cc                 simple-point-to-point.cc
csma-multicast.cc                 tcp-large-transfer-errors.cc
csma-one-subnet.cc                tcp-large-transfer.cc
csma-packet-socket.cc             tcp-nonlistening-server.cc
mixed-global-routing.cc           tcp-small-transfer-oneloss.cc
simple-alternate-routing.cc       tcp-small-transfer.cc
simple-error-model.cc             udp-echo.cc
simple-global-routing.cc          waf
simple-point-to-point-olsr.cc     wscript
```

Outline

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

ns-3 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-3 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 mathieu/ns-3-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

```
void
WifiChannel::Send (Ptr<WifiPhy> sender, Ptr<const
  Packet> packet, ...)
{
  Ptr<MobilityModel> senderMobility = 0;
  Ptr<MobilityModel> receiverMobility = 0;
  ...
  senderMobility = sender->GetNode ()->
    GetObject<MobilityModel> ();
}
```

class Node does not need to know about 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. 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

The traditional C++ way

```
class Foo
{
public:
  void SetVar1 (uint32_t value);
  uint32_t GetVar1 (void);
  static void SetInitialVar1(uint32_t value);
  void SetVar2 (uint32_t value);
  uint32_t GetVar2 (void);
  static void SetInitialVar2(uint32_t value);
  ...
private:
  uint32_t m_var1;  // document var1
  uint32_t m_var2;  // document var2
  static uint32_t m_initial_var1;
  static uint32_t m_initial_var2;
}

Foo::Foo() : m_var1(Foo::m_initial_var1), m_var2(Foo::m_initial_var2)
{
}
```

to modify an instance of Foo, get the pointer somehow, and use the public accessor functions
to modify the default values, modify the statics
Default values may be available in a separate framework (e.g. ns-2 Bind())

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/[3-5]|[8-11]|1"
  - UdpL4Protocol object instance aggregated to matching nodes:
    "/$UdpL4Protocol"
  - 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::TxGain", Double (1.0));
```

- Syntax also supports string values:

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

Attribute   Value

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::SetAttribute("/NodeList/5/DeviceList/3/Phy
  /TxGain", Double(1.0));
Double d =
  Config::GetAttribute("/NodeList/5/NetDevice/3/Phy
  /TxGain");
```

- 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> rxNode = 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

```
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.:

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

How is all this implemented (overview)

```
class Foo P public Object
{
public:
  static TypeId GetTypeId (void);
private:
  uint32_t m_var1;  // document var1
  uint32_t m_var2;  // document var2
}

Foo::Foo() {
}

TypeId Foo::GetTypeId (void)
{
  static TypeId tid = TypeId("Foo");
  .SetParent (<ooParent>
  .SetGroupName ("FooDefaults")
  .AddConstructor<Foo> ();
  .AddAttribute ("m_var1", "document var1",
    UInteger(3),
    MakeUIntegerAccessor (&Foo::m_var1),
    MakeUIntegerChecker<uint32_t> ())
  .AddAttribute ("m_var2", "", ...)
  return tid;
}
```

A real TypeId example

```
TypeId
RandomWalk2dMobilityModel::GetTypeId (void)
{
  static TypeId tid = TypeId ("RandomWalkMobilityModel")
    .SetParent<MobilityModel> ()
    .SetGroupName ("Mobility")
    .AddConstructor<RandomWalk2dMobilityModel> ()
    .AddAttribute ("bounds",
                   "Bounds of the area to cruise.",
                   Rectangle (0.0, 0.0, 100.0, 100.0),
                   MakeRectangleAccessor (&RandomWalk2dMobilityModel::m_bounds),
                   MakeRectangleChecker ())
    .AddAttribute ("time",
                   "Change current direction and speed after moving for this delay.",
                   Seconds (1.0),
                   MakeTimeAccessor (&RandomWalk2dMobilityModel::m_modeTime),
                   MakeTimeChecker ())
    .AddAttribute ("distance",
                   "Change current direction and speed after moving for this distance.",
                   Seconds (1.0),
                   MakeTimeAccessor (&RandomWalk2dMobilityModel::m_modeTime),
                   MakeTimeChecker ())
```

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:

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

Statements you should understand now

```
Ptr<Ipv4AddressAllocator> ipAddrs = CreateObject<Ipv4AddressAllocator> ();
```

C++ Smart Pointer

ns3::Object

```
Config::SetDefault ("OnOffApplication::DataRate", String("448kb/s"));

Config::SetDefault ("/NodeList/*/DeviceList/*/Phy/TxGain", Double(10.0));
```

Attribute namespace

Tracing model

- Tracing is a structured form of simulation output
  - tracing format should be relatively static across simulator releases
- 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
```

- Needs vary widely

Crude tracing

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

slightly less crude

```
#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

```
#include "ns3/ascii-trace.h"



 AsciiTrace asciitrace ("mixed-wireless.tr");
 asciitrace.TraceAllQueues ();
 asciitrace.TraceAllNetDeviceRx ();
```

Simple ns-3 tracing (pcap version)

- these are wrapper functions/classes
- see examples/mixed-wireless.cc

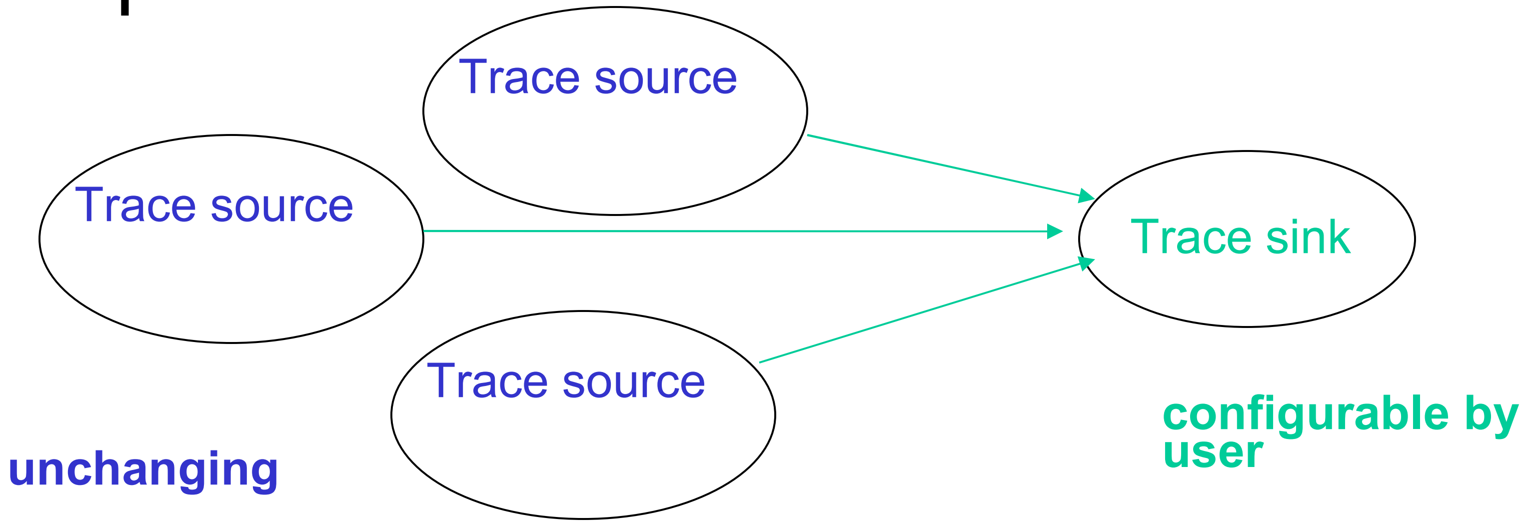
```
#include "ns3/pcap-trace.h"



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

ns-3 tracing model (revisit)

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



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

- 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

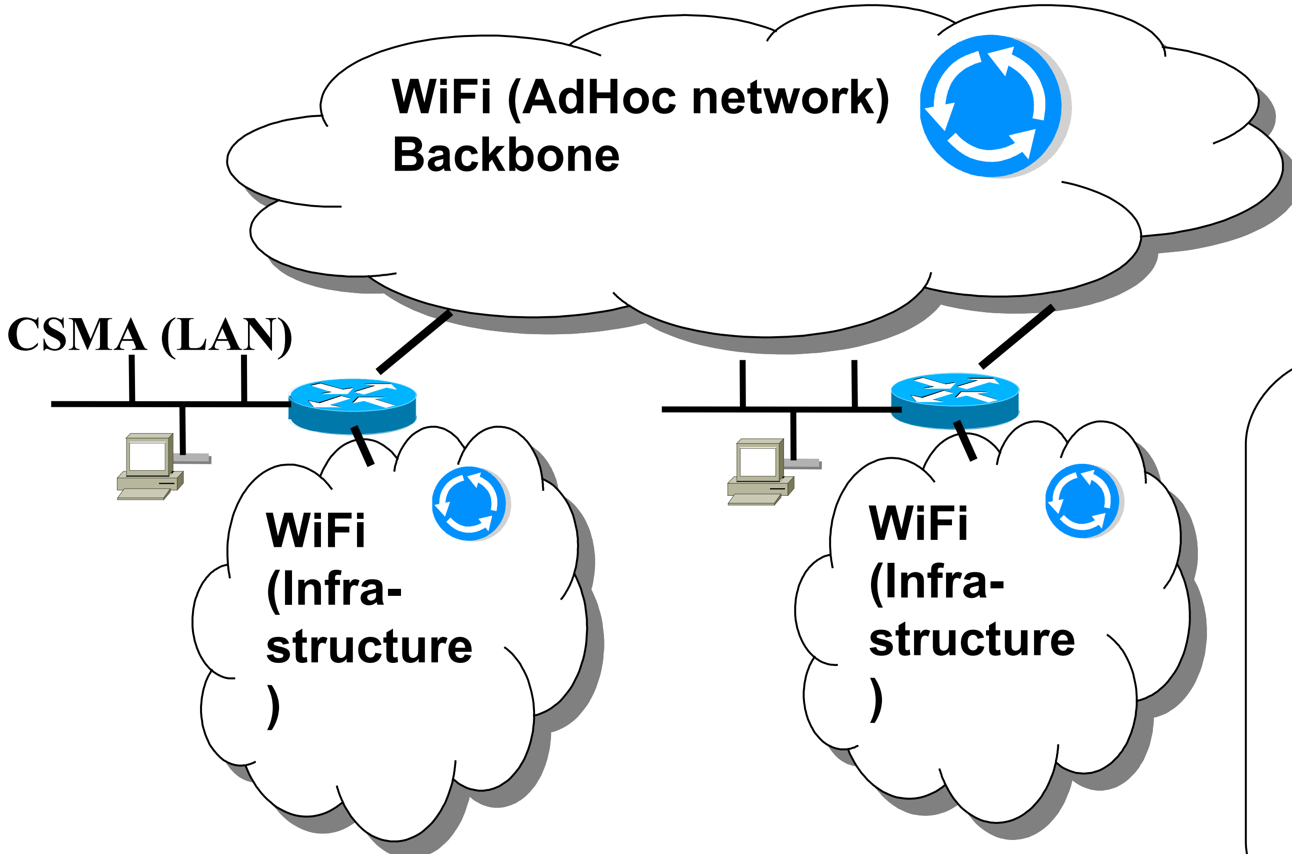
```
Config::Connect ("/NodeList/.../Source",
                 MakeCallback (&ConfigTest::ChangeNotification, this));
```

Statistics

Disclaimer: not yet part of ns-3

- Avoid large trace files
- Full statistics support planned for later in 2008
- Reuse tracing framework
- One similar approach: ns-2-measure project
  - http://info.iet.unipi.it/~cng/ns2measure/
  - 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

Revisit our script



Applications: TCP flow,
Routing: OLSR routing

Hierarchical mobility
1) AdHoc movement in backbone
2) Local movement relative to Access Point

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

The Helper approach

- 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

setup backbone

```
NodeContainer backbone;
backbone.Create (20);
MobilityHelper mobility;
mobility.SetPositionAllocator (“GridPositionAllocator”, “MinX”,
   Double (-100), ...);
mobility.SetMobilityModel (“RandomDirectionMobilityModel”)
mobility.Layout (backbone
WifiHelper wifi;
wifi.SetMac (“AdhocWifiMac”);
wifi.SetPhy (“WifiPhy”, “TxGain”, Double (10));
wifi.SetRemoteStationManager (“ConstantRateWifiManager”, “DataMode”,
   String (“wifia-54mb”))
Ptr<WifiChannel> channel = ...;
NetDeviceContainer backboneDev = wifi.Build (backbone, channel);
```

setup wifi subnets

```
for (uint32_t I = 0; I < 20; i++)
    NodeContainer subnet;
    subnet.Create (29);
    subnets.push_back (subnet);
    mobility.PushReferenceModel (backbone.Get (i));
    mobility.SetMobilityModel (...)
    mobility.SetPositionAllocator (...);
    mobility.Layout (subnet);
    subnet.Add (backbone.Get (i));
    Ptr<WifiChannel> subnetChannel = ...;
    NetDeviceContainer subnetDev =
      wifi.Build (subnet, subnetChannel);
    subnetDevs.push_back (subnetDev);
```

setup ip over backbone and subnets

```
IpNetworkAddressAllocator network;
network.SetMask ("192.168.0..0", "255.255.0.0");
InternetStackHelper ip;
ip.SetAddressAllocator (network.GetNext ());
ip.Setup (backboneDev);
for (uint32_t I = 0; I < 20; i++)
    NetDeviceContainer subnetDev = subnetDevs[i];
    ip.SetAddressAllocator (network.GetNext ());
    ip.Setup (subnetDev);
```

setup olsr on backbone

```
OlsrHelper olsr;
olsr.Enable (backbone);
```

setup traffic sinks everywhere

```
TrafficSinkHelper sink;
// listen on port 1026 for protocol udp
sink.EnableUdp (1026);
sink.Setup (NodeList::Begin (), NodeList::End ());
```

setup trace sources

```
OnOffApplicationHelper source;
source.SetUdpDestination (“168.192.4.10”, 1026);
NodeContainer one = subnets[2].Get ();
source.Setup (one);
```

Frameworks

- Observation: Many of the operations executed by the helper class are repetitively executed, in slightly different ways

```
// Create Nodes
// Add NetDevice and Channel
// Add Protocol Stack
// Add Applications
// Add Mobility
```

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

Frameworks

- This design pattern is called Inversion Of Control
- This provides more reusable scenario/topology scripts than ones based on the Helper classes

walk through mixed-wireless-topology.cc and src/topology/

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

ns-3 models are composed of hooking C++ classes together in the traditional way, and also with object 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 templatized

- Declaration:

```
template <typename T>
class MyStack
{
  void Push (T data);
  T Pop (void);
};
```

- Definition:

```
template <typename T> void MyStack<T>::Push (T data)
{ ... }
```

- Usage:

```
MyStack<int> stack;
stack.Push (x);
y = stack.Pop ();
```

Scheduler and callbacks

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

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

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

  Simulator::Run ();

  Simulator::Destroy ();
}
```

ns-3 callbacks

- Class template Callback<> implements the functor design pattern
- Callbacks are like function pointers, but more type-safe

```
static double CbOne (double a, double b) {}
          ^           ^          ^
          |        ---|    ------|
          |        |       |
Callback<double, double, double> one;
```

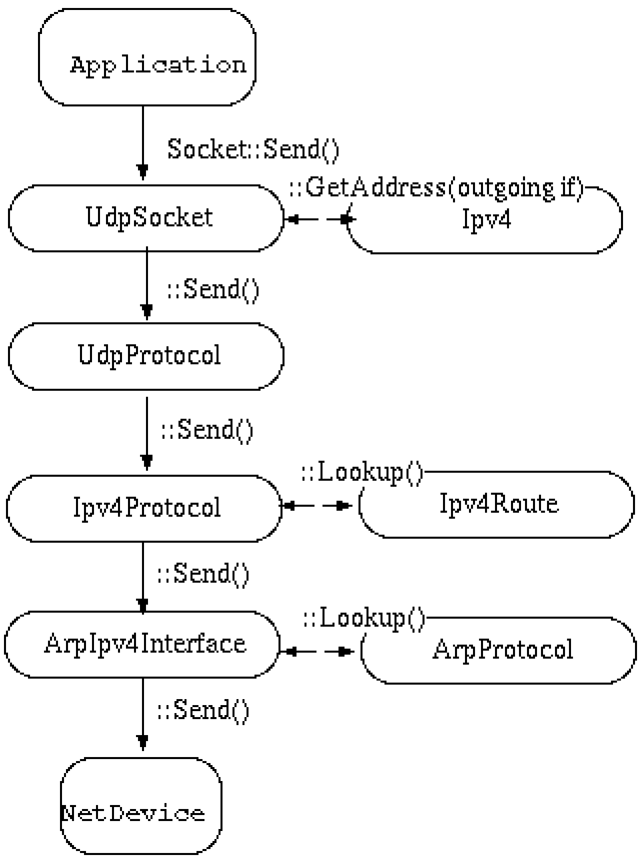
- Bind a function with a matching signature to a callback

```
one = MakeCallback (&CbOne);
double returnOne = one (10.0, 20.0);
```

Path of a packet (send)

Note: This architecture is under additional work

Function/object trace for sending a packet



Step in packet sending process:

1. The Application has previously created a socket (here, a UdpSocket). It calls Socket::Send(). Either real data or dummy data is passed at the API.

2. Socket::Send() forwards to UdpSocket::DoSend() and later to UdpSocket::DoSendTo(). These functions set the proper source and destination addresses, handle socket calls such as bind() and connect() and then the UdpProtocol::Send() function is called. As in a real implementation, the socket must query the Ipv4 layer to find the right source address to match the destination address.

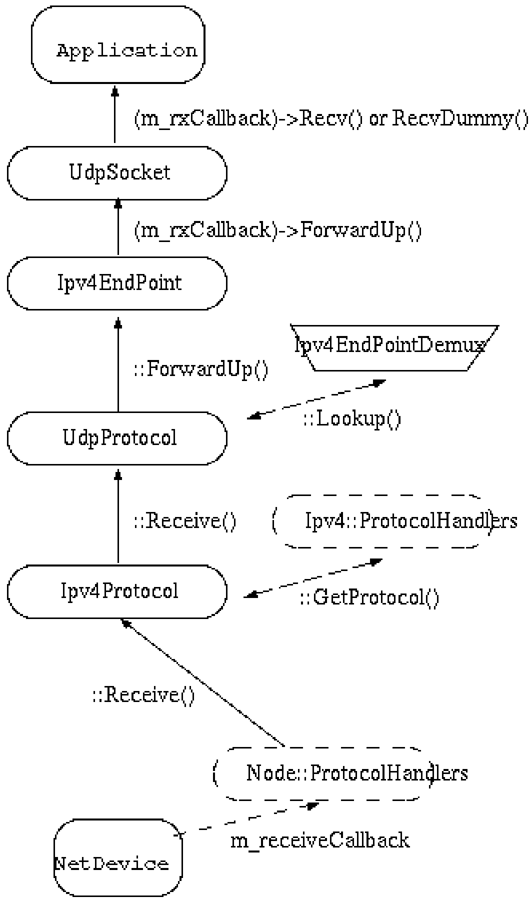
3. UdpProtocol is where the socket-independent protocol logic for UDP is implemented. The Send() method adds the UDP header, initializes the checksum, and sends the packet to the Ipv4 layer. Here, a private API (Ipv4Private) is queried, and the Send() method is called.

4. Ipv4Protocol adds the IP header, looks up a route, and sends the packet to an appropriate Ipv4Interface instance. In this example, the device is one that supports Arp, so the packet is sent to an ArpIpv4Interface object.

5. Ipv4Interface is an abstract base class; here, we depict the ArpIpv4Interface concrete class. This object looks up the MAC address if Arp is supported on this NetDevice technology, and if there is a cache hit, it sends it to the NetDevice, or else it first initiates an Arp request.

Path of a packet (receive)

Function/object trace for receiving a packet



Step in packet receive process:

7. UdpSocket itself calls one of two callbacks to get the data to the application. If the Application is sending fake data, the RecvDummy() callback is called; else, the Recv() callback is called.

6. Ipv4EndPoint has a callback where a Socket object is able to register a receive method. Here, this callback calls to UdpSocket::ForwardUp()

5. UdpProtocol is where the socket-independent protocol logic for UDP is implemented. The Receive() method removes the UDP header and looks up the per-flow context state, which is one or more Ipv4EndPoint objects stored in an Ipv4EndPointDemux (indexed by src addr, src port, dest addr, dest port). It then calls Ipv4EndPoint::ForwardUp() when done.

4. Ipv4Protocol removes the IP header, checks checksum (if implemented), and either Forwards the packet or calls ForwardUp(). ForwardUp() then looks up the IP protocol number in a callback-based demultiplexer (similar to Node::ProtocolHandlers, and calls the registered ::Receive() method.

3. Node::ReceiveFromDevice stores a set of callbacks that are looked up based on protocol number and device. In this case, the lookup will result in Ipv4Protocol::Receive() being called.

2. This is typically the Node::ReceiveFromDevice() function

1. NetDevice calls the function registered at Node::m_receiveCallback

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()`

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

Writing new ns-3 models

3) Create a non-functional skeleton

- review coding style
- decide which compilation unit it resides in
- add to waf
- build with body ifdeffed out
- copyright and headers
- initial doxygen

Writing new ns-3 models

4) Build a skeleton

- header include guards
- namespace ns3
- constructor, empty function prototypes
- key variables
- Object/TypeId code
- write small test program
- start a unit test

Writing new ns-3 models

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/errmodel.{cc,h}
  - ns-3: src/common/error-model.{cc,h}

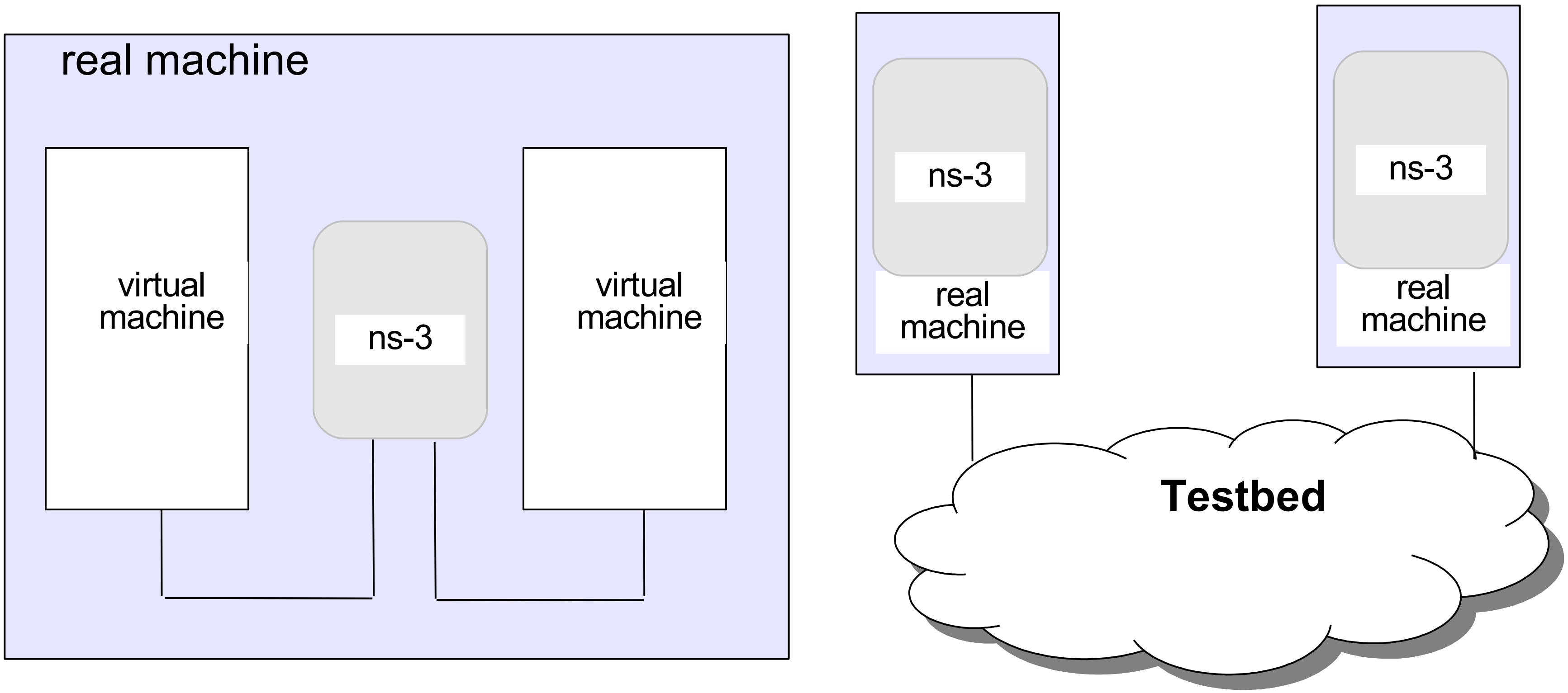
Validation

- Can you trust ns-3 simulations?
  - Can you trust *any* simulation?
  - Onus 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 a NetDevice and a Channel
- Add new types of transport layers:
- subclass Node and Socket
- subclass Ipv4 class to implement per-node Ipv4 forwarding table and Ipv4
- interface configuration
- for example, the Linux TCP stack could be easily integrated into a new type of node, LinuxNode with a LinuxTcpSocket
- Add a new type of traffic generation and analysis:
- subclass Application
- • use Socket API

ns-3 goals for emulation



1) ns-3 interconnects virtual machines

2) testbeds interconnect 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:

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