ns–3, The Network Simulator: WNS2 Tutorial

October 23, 2008
The ns–3 project

1. Funded in 2006 by the U.S. National Science Foundation
2. Four year effort.
3. Principal Investigators
   - Tom Henderson, University of Washington
   - George Riley, Georgia Institute of Technology
   - Sumit Roy, University of Washington
   - Sally Floyd, ICIR (through 2006-07)
4. Full–time personnel
   - Mathieu Lacage (funded by Planete group, INRIA, France)
   - Raj Bhattacharjea - Georgia Tech
   - Craig Dowell - University of Washington
5. Other major contributors
   - Gustavo Carneiro (mostly the python bindings, OLSR)
   - Joe Kopena
   - Sam Jansen
   - Google Summer of Code funded three students (Florian Westphal, Hagen Pﬁefer, and Liu Jian)
Overview

Introduction to *ns–3*

Design goals and Motivation

Simulated Entities
  Nodes
  Applications
  Packets
  Protocols
  NetDevices
  Channels
  The Simulator
  Attributes

Getting Started
  Obtaining *ns–3*
  Compiling/Building *ns–3*

First Example

Simulation Scripts
  Running Scripts
  Logging
  Helpers
  Tracing

Second Example

More Advanced Features
  Command-line arguments

Third Example

Support Features
  Smart Pointers
  Random Variables
  Object Aggregation

Examples Session

Conclusion
ns–3 is a network simulator for research simulation purposes.

- Discrete event simulator
- Written in C++ and released under the terms of the GNU GPLv2
- User scripts are written in C++, or via Python bindings provided to C++ APIs
- Simulator entities are analogous to the real world (sockets, applications, channels, etc.)
- Modular simulator core
- Re–designed from scratch (only a spiritual successor to ns2)
Design goals and Motivation

- Easy extensibility
- Understandability for programmers
- Open source and free to modify and study
- Multi-platform (*NIX systems, including Windows via cygwin)
- Ease of contributing, open development community.
- Real world design philosophy/concepts
  - Network stacks, sockets, and network interfaces are true to real implementations
  - Real code integration (real linux stacks, real sockets application *binaries*)
  - Emulation mode - inject real packets into real networks
Simulated Entities

**Existing core ns-2 capability**
- ping, vat, telnet, FTP, multicast FTP, HTTP, probabilistic and trace-driven traffic generators, webcache

**Transport layer**
- TCP (many variants), UDP, SCTP, XCP, TFRC, RAP, RTP
- Multicast: PGM, SRM, RLM, PLM

**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

**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

**Physical layer**
- TwoWay, Shadowing, OmniAntennas, EnergyModel, Satellite Repeater

**Support**
- Random number generators, tracing, monitors, mathematical support, test suite, animation (nam), error models

**Existing ns-3**
- OnOffApplication, asynchronous sockets API, packet sockets

**UDP, TCP**
- Unicast: IPv4, global static routing
- Multicast: static routing
- MANET: OLSR

**PointToPoint, CSMA, 802.11 MAC low and high and rate control algorithms**

**802.11a, Friis propagation loss model, log distance propagation loss model, basic wired (loss, delay)**

**Random number generators, tracing, unit tests, logging, callbacks, mobility visualizer, error models**
Simulated Entities

▶ Nodes
   Entities which form the network, connected by channels

▶ Applications
   Typically the final data sending/receiving entities, exist on nodes

▶ Packets
   What is sent across networks

▶ Protocols
   Manage connectivity
   A sockets like API exists for Applications to send and receive data through protocols

▶ NetDevices
   Interfaces protocol stack with channels/physical transmission medium

▶ Channels
   Transmission Medium
Simulated Entities

Diagram showing the interaction between
- Application
- Protocol stack
- NetDevice
- Node

Communications include:
- Sockets-like API
- Packet(s)
Nodes

- Abstraction of the "boxes" in your network
- Basically function as a collection of applications, protocol stacks, and net devices
- In the internet, these are the end hosts, routers, servers, etc.
- In MANETs, these are the mobile hosts, in WSNs, sensors, in cell networks, phones, etc.
Applications

- These generate and consume traffic in the networks
- Simple sending applications such as bulk data transfer, constant bit rate transfer, random on-off transfer, and echo are supported presently out of the box
- Simple consuming application which sinks Packets is also currently supported
- Researchers will develop application models as their work will require
Packets

- Actual network traffic
- "Packet" refers to this data at all layers of the protocol stack
- Very smart byte buffers; all packet data is internally represented as a serialized string of bytes. Contrast with other simulators which use a list of header objects.
- Header object concept retained, but headers are serialized transparently into the Packet when added.
- Optimized with the copy-on-write (COW) technique; copying/passing packets as parameters isn’t as memory/time consuming.
Packets

- Overall benefit: packet internal representation is the same as the real world. Means easy support of e.g. fragmentation, PCAP traces, emulation mode.
- Optionally has a non-serialized "metadata" which keeps track of headers and trailers for easy printing.
- Supports "tag" objects which can be used for e.g. flow ID, cross-layer info, delay/jitter calculation, etc. Contrast with hacking extra fields into ns2 headers.
Protocols

- Sit between Applications and NetDevices, broker connections, medium access, addressing, routing, etc
- Full internet-stack supported, with IPv4, v6 on the way, TCP, UDP, ARP
- Also uses Network Simulation Cradle technology to allow the use of *unmodified* Linux kernel network stack code, with similar support for BSD on the way
- Global static precomputed routing available for wired type of topologies, OLSR for wireless
NetDevices

- Actual interface between the protocol stack and the Channel
- Models e.g. the ethernet card of your PC
- *ns–3* inherently supports multiple interfaces per node, of different types
- Currently types include 802.11, ethernet-like CSMA, serial-like point-to-point
- Also some bridging code which allows traffic to flow across devices types on a node, e.g. wired ⇔ wireless
Channels

- Model for the transmission medium
- Typically have a data capacity, transmission delay, loss characteristics, etc.
- Connects two or more NetDevices together such that Packets can be transmitted and received.
- Typically implemented as a list of connected NetDevices, with APIs for sending and receiving on the medium.
- \textit{ns–3} has models for both point-to-point and multipoint channels (simple serial channel, ethernet-like CSMA, and 802.11).
- NetDevice types are tied to the Channel types, i.e. wifi devices must be on wifi channel.
The Simulator

- Simulation time moves discretely from event to event
- Simulation schedules events to occur at specific times, e.g. "Schedule the receipt of this packet after some delay".
- Scheduler priority queue; events are ordered by time of execution
- Events invoke a function; implemented using callbacks
- Simulator::Run() method starts processing events from the queue one by one
- Simulation is over when event queue is empty, or at a scheduled stop event at user specified time
Most network simulators (ns–2, GTNetS) use sub–classing and virtual functions to schedule events.

Define base class **Handler**

Virtual function **void Handle(void* Params)**;

Specify handler object when scheduling events.

This has several problems.

- Opaque pointer for parameters (requires typecast)
- Only one handler function per object.
- Can’t schedule static or global functions.
**ns–3 Event Handlers**

- **ns–3** uses a *templated member function pointer* to achieve type–safe and flexible event handlers.

  - For example:
    ```cpp
template <typename MEM, typename OBJ, 
typename T1, 
typename T2, 
typename T3>
static EventId Schedule (Time const &time, 
MEM mem_ptr, 
OBJ obj, 
T1 a1, T2 a2, T3 a3);
```

  - `mem_ptr` is the function pointer
  - `obj` is the object that has function `mem_ptr`
  - `a1`, `a2`, and `a3` are all variables *of any type*.
  - A similar `Schedule` function is defined for non–member functions for handlers (global or static).
ns-3 also makes heavy use of callbacks.

A callback is a function pointer, but with type-safe parameters.

Typically defined with a typedef

define Callback<void,

Ptr<NetDevice>.,

Ptr<const Packet>,

uint16_t,

const Address &,

const Address &,

NetDevice::PacketType> ProtocolHandler;

The first argument is the return type of the callback function (void in this case).

Remaining arguments are the types of the parameters to the callback function.

A variable of type ProtocolHandler (in this example) can later be called as if it were a function:

ProtocolHandler myProtoHandler = MakeCallback(.... //省略

myProtoHandler (device, packet, protocol, from, to, p...
Making a Callback Variable

- Once the callback type is defined, it can (optionally) be bound to a function using `MakeCallback`;
- `myProtoHandler = MakeCallback(&Ipv4L3Protocol::Receive, this);
- The parameter list on the specified function (`Receive` *MUST* match the parameter list specified in the `Callback` `typedef` exactly. If not, a compile error occurs.
- When the callback is invoked, as on the previous slide, the specified function (`Receive`) on the specified object (`this`) is called with the specified parameter values.

Advantages of callbacks:
- Any object can register and receive callbacks from any other object.
- There are no class hierarchy requirements.

Disadvantages of callbacks:
- The flow of the function calls can be hard to follow.
Smart Pointers

- *ns–3* makes heavy use of *Smart Pointers*
- Smart pointers automatically free the memory used by the pointer *only when the last reference is destroyed*.
- Thus, if a function returns a smart pointer, the programmer need not take any specific action to free the memory.
- Rather, you just let the pointer variable go out of scope.
- Alternately, you can just assign a value of zero to the pointer, which forces a decrement of the reference counter.
- Smart pointers are created with `CreateObject`.
  ```cpp
  Ptr<UdpSocketImpl> socket = CreateObject<UdpSocketImpl>();
  ```
- Smart Pointers are returned by many of the *ns–3* functions. For example, in `application.h`:
  ```cpp
  Ptr<Node> GetNode () const;
  ```
- The returned pointer is simply allowed to go out of scope to decrement the reference count.
Getting Started With ns–3
Obtaining *ns–3*

- Distributed as source code (no binaries maintained by the *ns–3* core developers)
- Via mercurial repository: http://code.nsnam.org/
  - All previous releases are kept here, as well as experimental and developmental branches
- Tarball source releases http://www.nsnam.org
Compiling/Building *ns–3*

- **Prerequisites:**
  - GNU compiler tool chain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer

- *ns–3* uses the waf build system based on Python (instead of the GNU autotools configure, make, etc)

- Just run `./waf` in the source directory; this is like `./configure && make`

- waf not only builds *ns–3*, it can be used to run example programs

- Now you are ready for your first example program, found in `examples/first.cc`

- Copy this file into the scratch directory and run with the command `./waf --run scratch/first`
Example Code: first.cc
Running Examples and Scripts

- All of the *.cc files in the examples directory can be run with ./waf --run ...
- In addition, you can drop simulation scripts into the scratch directory, and they will be built automatically (what we did with first.cc)
- Run things from the scratch directory with ./waf --run scratch/...
- Advanced users can write their simulations scripts, include the ns–3 headers, and link against the ns–3 library, bypassing waf for their simulations altogether
Logging

- Runtime messages indicating debugging info, and soft errors/warnings
- Provide understanding the internals of model
- Can be runtime enabled with LogComponentEnable
- Should not necessarily be used to trace the simulation, there is separate tracing functionality
- UDP echo example: logging output: "Sent 1024 bytes to 10.1.1.2..."
Helpers

- Entities are not directly manipulated by the user one by one.
- Overlay API which allows manipulating these in groups.
- Allows for quick scenario / topology construction.
- Saw in first.cc
Tracing

- Tracing system allows simulation writers access to exactly the interesting simulation info
- Lowest level functionality: a user specified callback is invoked when a particular event occurs
  - NetDevice receives a frame
  - TCP congestion window changes
  - Random sending application sends a packet
  - A Queue drops a packet
  - etc.
- Can use these "hooks" can be used to generate PCAP output, collect statistics, results, etc.
- Primary method of data collection for analysis
Tracing

- Uses a namespace resembling operating system paths to connect a callback to an event, e.g.
  - `/NodeList/[i]/DeviceList/[j]/$ns3::WifiNetDevice/Rx`
  - Refers to Rx event on the $j^{th}$ NetDevice of the $i^{th}$ node

- These trace sources are connected to trace sinks, user provided methods using `Config::Connect`, e.g.
  - `Config::Connect("/NodeList/*/DeviceList/*/Tx", MakeCallback (&DevTxTrace));`
  - This gets the user provided function `DevTxTrace` a notification whenever any node, any NetDevices transmits a packet
  - User provided function can log this to file, output to screen, calculate some statistics, etc.

- Complete list can be found in the Doxygen documentation
Attributes

- The tracing namespace is reused for simulation configuration
- Attributes system allows you to change individual parameters on many simulations entities
- Also allows setting of global defaults for these parameters
  - Tweak the SIFS, DIFS, PIFS intervals for a specific NetDevice’s MAC
  - Set the initial congestion window size for all created TCP instances
  - etc.
- Uses the Config::Set and Config::SetDefault APIs
- "ns3::WifiRemoteStationManager::RtsCtsThreshold"; used to set RTS/CTS behavior in Wifi
- "/NodeList/5/$ns3::MobilityModel/$ns3::RandomWaypointMobilityModel/Speed"; change the random waypoint mobility model on node 5 to use a uniform distribution random number generator to generate the speeds
- More info in the doxygen documentation
Example Code: second.cc
Command-line arguments

- Allows for command line configuration of simulation parameters
- Can add custom command line arguments to modify; in second.cc, we saw nCsma, the number of extra LAN nodes
- The attributes discussed before are all exported through this system as well
- --PrintHelp shows possible parameters
- Example: ./waf --run "scratch/second --PrintHelp"
- Example: ./waf --run "scratch/second --PrintAttributes=ns3::CsmaChannel"
- Example: ./waf --run "scratch/second --ns3::CsmaChannel::DataRate=10000000000bps"
Example Code: third.cc
Smart Pointers

- In third.cc you saw: `Ptr<WifiChannel>`
- This is an *ns–3* smart-pointer
- Behaves just like a pointer, except it manages a reference count
- Each time the smart-pointer is copied or passed around as a parameter, a reference count is incremented on the underlying object
- Each time the smart-pointer is destructed or set to 0, the reference count gets decremented on the underlying object
- When there are no more outstanding references to an object in the system, it gets automatically deleted; *ns–3* method of doing garbage collection, developers and users don’t have to worry about explicit memory management
Smart Pointers

- Typical usage: `Ptr<X> object = CreateObject<X>(...);`
- This maps to the regular C++: `X* object = new X(...);`
- `CreateObject<>` does some magic to make reference counting and attributes work, and you don’t have to worry about calling delete!
- One atypical case is Packets; Packets are created with `Create<Packet>(...);`
- Distinction between `Create` and `CreateObject` is subtle, has to do with Packets lacking attributes
- Most users won’t encounter this, but use `CreateObject<>` for children of `ns3::Object`, `Create<>` for all other smart-pointers
Random Variables

- *ns–3* supports pseudo random numbers from many types of distributions
  - Normal
  - Exponential
  - LogNormal
  - Pareto
  - Triangular
  - Uniform
  - Weibull
  - Constant

- User can also specify any arbitrary cumulative distribution function

- Internally uses L’ecuyer MRG32k3a generator

- Models use these generators: RandomWalk2dMobilityModel, OnOffApplication, etc.

- Usage Example:
  ```cpp
  UniformVariable x(0,100);
  int r = x.GetValue();
  ```
Random Variables, continued

- *ns–3* random number generators (RNGs) produced guaranteed *independent* streams of values, e.g.:
  - UniformVariable x(0,100); UniformVariable y(0,100);
  - The two stream of numbers from repeated calls to x.GetValue() and y.GetValue() will be statistically uncorrelated

- *ns–3* Random Variables default to being seeded by the time of day (or /dev/random if it is available)
- Insures that different runs of a simulation using randomness are different from each other
- If the user wants to avoid this behavior (get replicability) he must set the seed himself
  - RandomVariable::UseGlobalSeed (a,b,c,d,e,f);
Random Variables, continued

- User can then set a "run number" which increments the internal state of the RNG in a predictable way
  - `RandomVariable::SetRunNumber (N);`
- For a fixed seed and given run number, the simulation results should be the same
- By varying the run number with a fixed seed you get different trials of the simulation, can show statistical significance, etc.
- This behavior will change in ns-3.3: the behavior will default to a fixed seed instead of seeding by time of day or `/dev/random` (this is in line with other simulators and a better statistical practice)
- User will have to explicitly set time of day or `/dev/random` seeding
Object Aggregation

- **ns3::Object base class was mentioned earlier**
- Allows children classes reference counting and attributes; also serves a third purpose, object aggregation
- **An ns–3 aggregation is collection of Object subclasses such that there is a maximum of only one Object of each Object subclass**
- Designed to solve the problem of knowing what "capabilities" an object has
  - If a Node has internet (IPv4) capabilities, it should have some TCP/UDP/IP objects aggregated to it that control this functionality
  - Similarly for if a Node has mobility
  - etc.
Object Aggregation

- You can "query" an object to see if it actually has the functionality you are looking for; this is done with `Object::GetObject<X>()`, which returns a `Ptr<X>` smart-pointer
- If the returned pointer is non-null, you have a pointer to the object which controls that functionality
- For those familiar, this model takes inspiration from Microsoft COM, QueryInterface, etc.
Examples Session

Possible Examples (as time permits, in no particular order)

- mixed-wireless
- tcp-large-transfer
- tcp-star-server
- tcp-nsc-zoo
- wifi-adhoc
- wifi-ap
Conclusion and Future Directions

- *ns–3* is still being developed and maintained
- Real world design philosophy allows for new, exciting things; some of these are in ns3.2, some are experimental or planned for future release
  - running unmodified real world network stacks
  - emulation and integration into test-beds
  - binary loaders for real applications
  - parallel/distributed simulation
- The set of *ns–3* models is being expanded, with ICMP and IPv6 soon to be released, WiMax in experimental stage, etc.
Conclusion and Future Directions

- New models could be added by researchers like yourselves!
- \textit{ns–3} community is growing each month, with thousands of researchers downloading the 3.2 release
- Contributor base is growing, with groups providing their code for merging into the main branch
- Conference papers using \textit{ns–3} are trickling in.
- In short: it is an exciting time to be involved with \textit{ns–3}!
- Go try it out and convince your research group to use it!
- Finally, help is available:
  - ns-3-users list: http://groups.google.com/group/ns-3-users/
  - ns-developers mailing list: http://mailman.isi.edu/mailman/listinfo/ns-developers
  - developer chat on IRC: irc.freenode.net, room #ns-3