ns3, The Network Simulator: WNS2 Tutorial
October 23, 2008
Overview

Introduction to ns3
Design goals and Motivation
Simulated Entities
  Nodes
  Applications
  Packets
  Protocols
  NetDevices
  Channels
  The Simulator
  Attributes

Getting Started
  Obtaining ns3
  Compiling/Building ns3

First Example
Simulation Scripts
  Running Scripts
  Logging
  Helpers
  Tracing

Second Example

More Advanced Features
  Command-line arguments
Introduction to \textit{ns3}

\textit{ns3} is a network simulator for research simulation purposes.

- Discrete event simulator
Introduction to *ns3*

*ns3* is a network simulator for research simulation purposes.

- Discrete event simulator
- *Written in C++ and released under the terms of the GNU GPLv2*
Introduction to ns3

ns3 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
Introduction to *ns3*

*ns3* 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.)
Introduction to $ns3$

$ns3$ 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
Introduction to *ns3*

*ns3* 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
- Redesigned from scratch (only a spiritual successor to ns2)
Design goals and Motivation

- Easy extensibility
- Understandibility for programmers.
- Open source and free (as in freedom).
- Multi-platform (*NIX systems, including Windows via cygwin)
- Ease of contributing, open development community.
- Use of real world concepts
  - Real code integration (real linux stacks, real sockets application binaries)
  - Emulation mode - inject real packets into real networks
Design goals and Motivation

- Easy extensibility
Design goals and Motivation

- Easy extensibility
- Understandability for programmers.
Design goals and Motivation

- Easy extensibility
- Understandibility for programmers.
- Open source and free (as in freedom).
Design goals and Motivation

- Easy extensibility
- Understandibility for programmers.
- Open source and free (as in freedom).
- Multi-platform (*NIX systems, including Windows via cygwin)

- Use of real world concepts
- Real code integration (real Linux stacks, real TCP/IP application binaries)
- Emulation mode - inject real packets into real networks
Design goals and Motivation

- Easy extensibility
- Understandibility for programmers.
- Open source and free (as in freedom).
- Multi-platform (*NIX systems, including Windows via cygwin)
- Ease of contributing, open development community.
Design goals and Motivation

- Easy extensibility
- Understandability for programmers.
- Open source and free (as in freedom).
- Multi-platform (*NIX systems, including Windows via cygwin)
- Ease of contributing, open development community.
- Use of real world concepts
Design goals and Motivation

- Easy extensibility
- Understandability for programmers.
- Open source and free (as in freedom).
- Multi-platform (*NIX systems, including Windows via cygwin)
- Ease of contributing, open development community.
- Use of real world concepts
  - Real code integration (real linux stacks, real sockets application *binaries*)
Design goals and Motivation

- Easy extensibility
- Understandability for programmers.
- Open source and free (as in freedom).
- Multi-platform (*NIX systems, including Windows via cygwin)
- Ease of contributing, open development community.
- Use of real world concepts
  - 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**
- 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

- Applications

- Packets

- Protocols

- NetDevices

- Channels
Simulated Entities

- **Nodes**
  Entities which form the network, connected by channels
Simulated Entities

- **Nodes**
  Entities which form the network, connected by channels

- **Applications**
Simulated Entities

- **Nodes**
  Entities which form the network, connected by channels

- **Applications**
  Typically the final data sending/receiving entities, exist on nodes
Simulated Entities

- Nodes
  Entities which form the network, connected by channels

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

- Packets
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
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**
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
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
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**
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
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**
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**
  Medium of transmission
Simulated Entities

Application

Protocol stack

Node

NetDevice

Sockets-like API

Packet(s)

Application

Protocol stack

Node

NetDevice

Channel

Channel
Nodes

▶ Abstraction of the “boxes” in your network
▶ Basically function as a collection of applications, protocol stacks, and netdevices
▶ 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.
Nodes

- Abstraction of the "boxes" in your network
Nodes

- Abstraction of the "boxes" in your network
- Basically function as a collection of applications, protocol stacks, and netdevices
Nodes

- Abstraction of the "boxes" in your network
- Basically function as a collection of applications, protocol stacks, and netdevices
- In the internet, these are the end hosts, routers, servers, etc.
Nodes

- Abstraction of the "boxes" in your network
- Basically function as a collection of applications, protocol stacks, and netdevices
- 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.
Nodes

- Abstraction of the "boxes" in your network
- Basically function as a collection of applications, protocol stacks, and netdevices
- 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 bitrate transfer, random on-off transfer, and echo are supported presently out of the box. A simple consuming application which sinks packets is also currently supported. Researchers will develop application models as their work will require.
Applications

- These generate and consume traffic in the networks

Simple sending applications such as bulk data transfer, constant bitrate transfer, random on-off transfer, and echo are supported presently out of the box. A simple consuming application which sinks packets is also currently supported. Researchers will develop application models as their work will require.
Applications

- These generate and consume traffic in the networks
- Simple sending applications such as bulk data transfer, constant bitrate transfer, random on-off transfer, and echo are supported presently out of the box
Applications

- These generate and consume traffic in the networks
- Simple sending applications such as bulk data transfer, constant bitrate transfer, random on-off transfer, and echo are supported presently out of the box
- Simple consuming application which sinks Packets is also currently supported
Applications

- These generate and consume traffic in the networks
- Simple sending applications such as bulk data transfer, constant bitrate 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

- Actual network traffic
Packets

- Actual network traffic
- "Packet" refers to this data at all layers of the protocol stack
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.
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.
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. 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.
Packets

- Overall benefit: packet internal representation is the same as the real world. Means easy support of e.g. PCAP traces, emulation mode.
Packets

- Overall benefit: packet internal representation is the same as the real world. Means easy support of e.g. PCAP traces, emulation mode.
- Optionally has a non-serialized "metadata" which keeps track of headers and trailers for easy printing.
Packets

- Overall benefit: packet internal representation is the same as the real world. Means easy support of e.g. 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.
Packets

- Overall benefit: packet internal representation is the same as the real world. Means easy support of e.g. 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.
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
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
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
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

The NetDevices module modelizes the actual interface between the protocol stack and the Channel. It inherently supports multiple interfaces per node, of different types, currently including 802.11, ethernet-like CSMA, serial-like point-to-point, and some bridging code which allows traffic to flow across devices types on a node, e.g., wired⇔wireless.
NetDevices

- Actual interface between the protocol stack and the Channel

- Models e.g. the ethernet card of your PC

- ns3 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 device types on a node, e.g. wired⇔wireless
NetDevices

- Actual interface between the protocol stack and the Channel
- Modelizes e.g. the ethernet card of your PC
NetDevices

- Actual interface between the protocol stack and the Channel
- Modelizes e.g. the ethernet card of your PC
- \textit{ns3} inherently supports multiple interfaces per node, of different types
NetDevices

- Actual interface between the protocol stack and the Channel
- Modelizes e.g. the ethernet card of your PC
- *ns3* inherently supports multiple interfaces per node, of different types
- Currently types include 802.11, ethernet-like CSMA, serial-like point-to-point
NetDevices

- Actual interface between the protocol stack and the Channel
- Modelizes e.g. the ethernet card of your PC
- ns3 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
▶ ns3 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
Channels

- Model for the transmission medium
Channels

- Model for the transmission medium
- Typically have a data capacity, transmission delay, loss characteristics, etc.
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.
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
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
- ns3 has models for both point-to-point and multipoint channels (simple serial channel, ethernet-like CSMA, and 802.11)
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
- *ns3* 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.
The Simulator

- Simulation time moves discretely from event to event
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".
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
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
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
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
Getting Started With *ns3*
Obtaining *ns3*
Obtaining *ns3*

- Distributed as source code (no binaries maintained by the *ns3* core developers)
Obtaining *ns3*

- Distributed as source code (no binaries maintained by the *ns3* core developers)
- Via mercurial repository: http://code.nsnam.org/
Obtaining *ns3*

- Distributed as source code (no binaries maintained by the *ns3* core developers)
- Via mercurial repository: http://code.nsnam.org/
  - All previous releases are kept here, as well as experimental and developmental branches
Obtaining *ns3*

- Distributed as source code (no binaries maintained by the *ns3* 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 *ns3*

**Prerequisites:**
- GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
- Python 2.4 or newer
- *ns3* 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 *ns3*, 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`
Compiling/Building ns3

- Prerequisites:
  - GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer
  - ns3 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 ns3, 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`
Compiling/Building *ns3*

- **Prerequisites:**
  - GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer
  - *ns3* 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 *ns3*, 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`
Compiling/Building *ns3*

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

- Just run `./waf` in the source directory; this is like `./configure && make`.
- `waf` not only builds `ns3`, 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`.
Compiling/Building *ns3*

- Prerequisites:
  - GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer
- *ns3* 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 *ns3*, 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`
Compiling/Building *ns3*

- Prerequisites:
  - GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer
- *ns3* 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
Compiling/Building *ns3*

- Prerequisites:
  - GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer
- *ns3* 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 *ns3*, 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`
Compiling/Building *ns3*

- **Prerequisites:**
  - GNU compiler toolchain on Linux, Mac OS, or Windows (via Cygwin or MinGW)
  - Python 2.4 or newer
- *ns3* 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 *ns3*, it can be used to run example programs
- Now you are ready for your first example program, found in examples/first.cc
Compiling/Building *ns3*

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

- *ns3* 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 *ns3*, 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 ns3 headers, and link against the ns3 library, bypassing waf for their simulations altogether.
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)

- Advanced users can write their simulations scripts, include the ns3 headers, and link against the ns3 library, bypassing waf for their simulations altogether
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)
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/...
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 ns3 headers, and link against the ns3 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 the “Sent 1024 bytes to 10.1.1.2...” messages
Logging

- Runtime messages indicating debugging info, and soft errors/warnings
Logging

- Runtime messages indicating debugging info, and soft errors/warnings
- Provide understanding the internals of model
Logging

- Runtime messages indicating debugging info, and soft errors/warnings
- Provide understanding the internals of model
- Can be runtime enabled with LogComponentEnable

UDP echo example: logging output the “Sent 1024 bytes to 10.1.1.2...” messages
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
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 the "Sent 1024 bytes to 10.1.1.2..." messages
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
Entities are not directly manipulated by the user one by one.
Helpers

- Entities are not directly manipulated by the user one by one.
- Overlay API which allows manipulating these in groups.
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.
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 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" to generate PCAP output, collect statistics, results, etc. Primary method of data collection for analysis.
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 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.
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
- Can use these "hooks" to generate PCAP output, collect statistics, results, etc.
- Primary method of data collection for analysis
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
- Can use these "hooks" can be used to generate PCAP output, collect statistics, results, etc.
- Primary method of data collection for analysis
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
- Can use these "hooks" to generate PCAP output, collect statistics, results, etc.
- Primary method of data collection for analysis
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
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" to generate PCAP output, collect statistics, results, etc.

Primary method of data collection for analysis
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.
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.
Tracing

- Uses a namespace resembling operating system paths to connect a callback to an event, e.g.

\[
\text{Config::Connect} \left( \text{"/NodeList/*/DeviceList/*/Tx"}, \text{MakeCallback} \left( \&\text{DevTxTrace} \right) \right)
\]

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.
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
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\textsuperscript{th} NetDevice of the i\textsuperscript{th} node

These trace sources are connected to trace sinks, user-provided methods using Config::Connect, e.g.

\[
\text{Config::Connect } (/\text{NodeList/}[*]/\text{DeviceList/}[*]/\text{Rx}, \text{Callback (&DevTxTrace)}); \]

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.
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\textsuperscript{th} NetDevice of the i\textsuperscript{th} node

- These trace sources are connected to trace sinks, user provided methods using Config::Connect, e.g.
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));`
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 \(^{\text{jth}}\) NetDevice of the \(^{\text{i}}\text{th}\) 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

Complete list can be found in the Doxygen documentation
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\textsuperscript{th} NetDevice of the i\textsuperscript{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
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.
- 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.
Attributes

- The tracing namespace is reused for simulation configuration

- 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
Attributes

- The tracing namespace is reused for simulation configuration
- Attributes system allows you to change individual parameters on many simulations entities

More info 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
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

More info 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
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
- ":/nsns::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
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
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

More info 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
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
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.
Command-line arguments