
ns-3 Training

Session 1: Monday 8:30am

**ns-3 Annual Meeting
May 2014**

Logistics

- Introductions
- Wireless access
- Wiki page:
 - <http://www.nsnam.org/wiki/Training2014>

ns-3 training goals

- Learn about the project overall, and where to get additional help
- Understand the architecture and design goals of the software
- Learn how to write new code for the simulator
- Learn about selected topics in more detail
- Answer your questions

ns-3 training agenda

- Monday: Overview of ns-3
 - Session 1: Overview
 - Session 2: Core, basic models, I/O
 - Session 3: Tracing
 - Session 4: Writing new code
- Tuesday: Special topics
 - Session 1: WiFi and mobile network simulations
 - Session 2: LTE
 - Session 3: Parallel, distributed simulations
 - Session 4: DCE and Emulation

ns-3 training agenda: Monday morning

- 8:30-10h00: ns-3 overview
 - Overview of software and models
 - Running and understanding an existing example
 - Basic animation and visualization (netanim, flow monitor)
 - Integrating other tools and libraries
- 10:00-10:15: 15-minute coffee break
- 10:15-12:00: ns-3 core
 - object model
 - scheduler
 - callbacks
 - etc.

ns-3 training agenda: Monday afternoon

- 1:30-3:00: Tracing and data collection
 - Tracing subsystem in depth
 - Data collection framework
- 3:00-3:15: 15-minute coffee break
- 3:15-5:00: Writing new software, Q&A
 - Writing new examples and models

ns-3 project goals

Develop an extensible simulation environment for networking research

- 1) a tool aligned with the experimentation needs of modern networking research
- 2) a tool that elevates the technical rigor of network simulation practice
- 3) an open-source project that encourages community contribution, peer review, and long-term maintenance and validation of the software

How the project operates

- Project provides three annual software releases
- Users interact on mailing lists and using Bugzilla bug tracker
- Code may be proposed for merge
 - Code reviews occur on a Google site
- Maintainers (one for each module) fix or delegate bugs, participate in reviews
- Project has been conducting annual workshop and developer meeting around SIMUTools through 2013
 - Some additional meetings on ad hoc basis
- Google Summer of Code (March-August) five of the past six summers

Acknowledgment of support



Information Sciences Institute

Goals of the consortium

- The NS-3 Consortium is a collection of organizations cooperating to support and develop the ns-3 software.
- It operates in support of the open source project
 - by providing a point of contact between industrial members and ns-3 developers,
 - by sponsoring events in support of ns-3 such as users' days and workshops,
 - by guaranteeing maintenance support for ns-3's core, and
 - by supporting administrative activities necessary to conduct a large open source project.

What is ns-3?

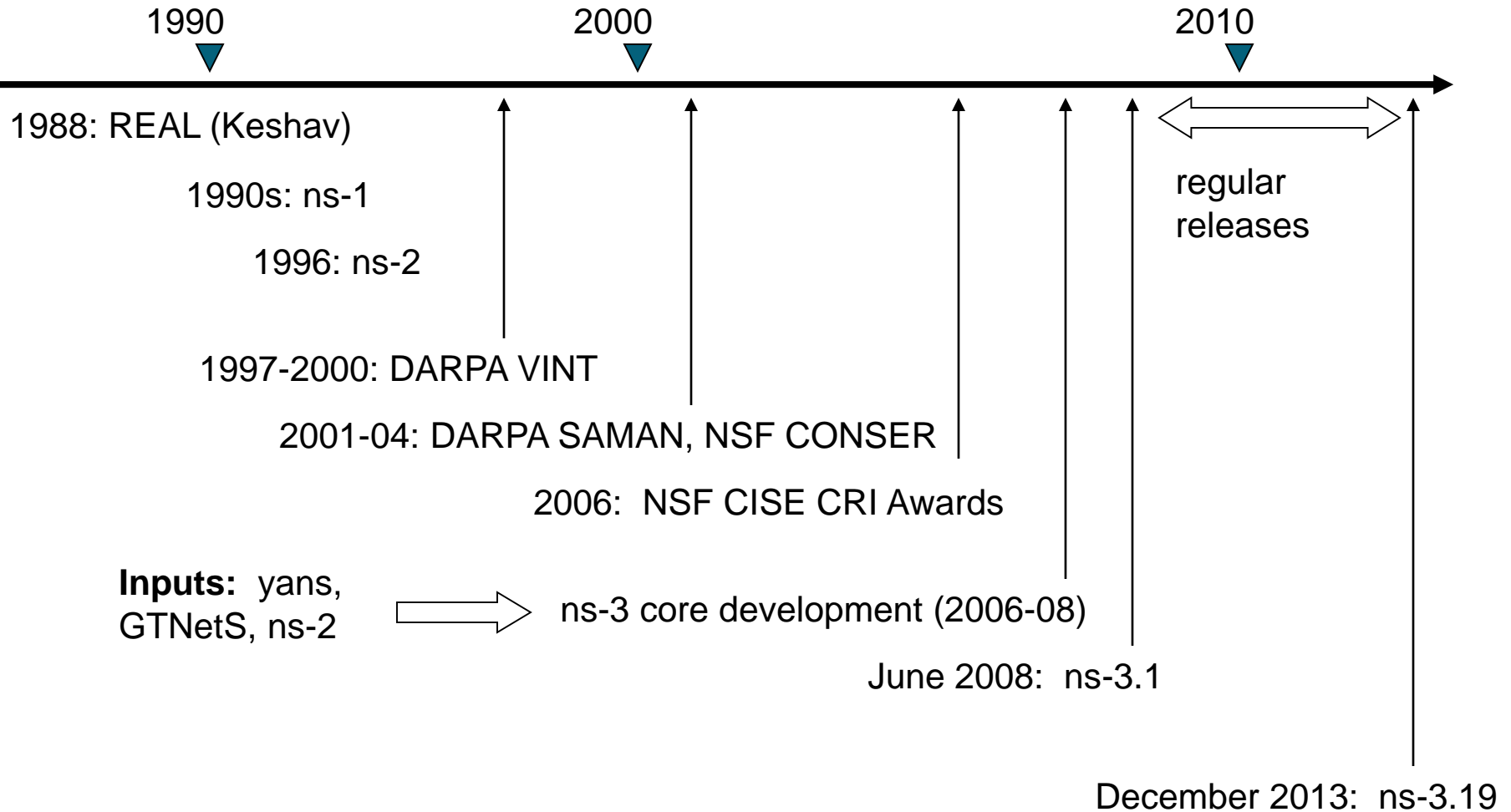
Discrete-event simulation basics

- Simulation time moves in discrete jumps 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

Preliminaries

- ns-3 is written in C++, with bindings available for Python
 - simulation programs are C++ executables or Python programs
 - ~300,000 lines of mostly C++ (estimate based on cloc source code analysis)
- ns-3 is a GNU GPLv2-licensed project
- ns-3 is mainly supported for Linux, OS X, and FreeBSD
 - Windows Visual Studio port available
- ns-3 is not backwards-compatible with ns-2

ns timeline



What have people done with ns-3?

- ~300 publications (March 2013)
– search of 'ns-3 simulator' on IEEE and ACM digital libraries

1014

IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 20, NO. 4, DECEMBER 2012

FSR: Formal Analysis and Implementation Toolkit for Safe Interdomain Routing

Andao Wang, Limin Jia, Member, IEEE, Wenchao Zhou, Ying Ren, Boon Thau Loo, Member, IEEE, Senior Member, IEEE, Vivek Nigam, Andre Secevor, and Carolyn Talcott

Abstract—Interdomain routing stitches the disparate parts of the Internet together, making protocol stability a critical issue to both researchers and practitioners. Yet, researchers create safety proofs and counterexamples by hand and build simulators and prototypes to explore protocol dynamics. Similarly, network operators analyze their router configurations manually or using long-term tools. In this paper, we present a comprehensive toolkit for analyzing and implementing routing policies, ranging from high-level guidelines to specific router configurations. Our *Formally Safe Routing (FSR)* toolkit performs all of these tasks from the same algorithmic representation of routing policy. We show that routing algorithms have a natural translation to both router constraints and the formal safety analysis (FSR) and declarative programs (its generated distributed implementations). Our extensive experiments with realistic topologies and policies show how FSR can detect problems in an autonomous system's (AS) BGP configuration, prove sufficient conditions for Border Gateway Protocol (BGP) safety, and empirically evaluate convergence time.

Index Terms—Communication technology, declarative networking, formal analysis, routing algorithms.

I. INTRODUCTION

THE INTERNET'S global routing system does not necessarily collapse, depending on how the Border Gateway Protocol (BGP) properties of individual networks are configured. Since protocol oscillations cause serious performance degradation and router overload, researchers devote significant attention to BGP stability (or "safety"). Abstract formal models of BGP [12]–[15], [36] allow researchers to explore how local policies affect BGP stability and identify policy configurations that, if universally adopted by ISPs, ensure global

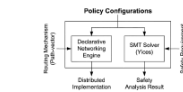


Fig. 1. FSR architecture

safety [4], [10]–[11], [33]. While our understanding of BGP safety has improved dramatically in the past decade, each research study still proceeds independently—usually creating proofs and counterexamples, and sometimes building simulators or prototypes to study protocol overhead and transient behavior during convergence. To aid the design, analysis, and evaluation of safe interdomain routing, we propose the *Formally Safe Routing (FSR)* toolkit. FSR serves two important communities: For researchers, FSR automates important parts of the design process and provides a common framework for describing, evaluating, and comparing new safety guidelines. For network operators, FSR automates the analysis of internal router (eBGP) and border gateway (eBGP) configurations for safety violations. For both communities, FSR automatically generates realistic protocol implementations to evaluate real network configurations (e.g., to study convergence time) prior to actual deployment. The ideas underlying FSR also apply research in routing algorithms [11], [36] with recent advances in declarative networking [22] to produce provably correct implementations of safe interdomain routing.

Given policy configurations as input, FSR produces an analysis of safety properties and a distributed protocol implementation, as shown in Fig. 1. FSR has three main underlying technologies:

- **Policy configuration as algebra:** Our extensions to routing algorithms [13], [36] allow researchers and network operators to express policy configurations in an abstract algebraic form. These configurations can be analyzed from high-level policy guidelines (e.g., proposed constraints that a researcher wants to study) or a completely specified policy instance (e.g., an ASBGP configuration or a multi-autonomous-system (AS) network that an operator wants to analyze). Router configurations can be automatically translated into the algebraic representation, easing the adoption of FSR.
- **Safety analysis:** To automatically analyze the policy configuration, FSR reduces the convergence proof to a

Wireless New (2011) 17:1775–1784
DOI 10.1007/s11256-011-0377-0

Message delivery in heterogeneous networks prone to episodic connectivity

Rao Naved Bin Rai · Thierry Taret · Kati Otterbach

Published online: 17 August 2011
© Springer Science+Business Media, LLC 2011

Abstract We present an efficient message delivery framework, called *MedHa*, which enables communication in an Internet connecting heterogeneous networks that is prone to disruptions in connectivity. *MedHa* is complementary to the IETF's Bundle Architecture: besides its ability to store messages for unavailable destinations, *MedHa* can bridge the connectivity gap between infrastructure-based and multi-hop infrastructure-less networks. It benefits from network heterogeneity (e.g., nodes supporting more than one network and nodes having diverse resources) to improve message delivery. For example, in IEEE 802.11 networks, participating nodes may use both infrastructure and ad-hoc nodes to deliver data to otherwise unavailable destinations. It also employs opportunistic routing to support nodes with episodic connectivity. One of *MedHa*'s key features is that any *MedHa* node can rely data to any destination and can act as a gateway to make two networks interoperate or to connect to the external network. The network is able to store data destined to temporarily unavailable nodes at the time of their expiry. This time period depends upon current storage availability as well as quality-of-service needs (e.g., delivery delay bounds) imposed by the application. We showcase

MedHa's ability to operate in environments consisting of a diverse set of interconnected networks and evaluate its performance through extensive simulations using a variety of scenarios with realistic synthetic and real mobility traces. Our results show significant improvement in average delivery ratio and a significant decrease in average delivery delay in the face of episodic connectivity. We also demonstrate that *MedHa* supports different levels of quality-of-service through traffic differentiation and message prioritization.

Keywords Disruption tolerance · Episodic connectivity · Heterogeneous networks · Node relaying · Store-carry-and-forward · DTN routing

1 Introduction

It is envisioned that the Internet of the future will be highly heterogeneous not only due to the wide variety of end devices it encompasses, but also in terms of the underlying network it comprises. Figure 1 illustrates networks that range from wired and wireless backbones (e.g., community wireless mesh networks) to wireless infrastructure-based and ad-hoc networks (e.g., MANETs). On the other hand, current and emerging applications, such as emergency response, environmental monitoring, smart environments (e.g., smart offices, homes, museums, etc.), and vehicular networks, among others, imply frequent and relatively long-lived disruptions in connectivity. The resulting disruption- or delay-tolerant networks (DTNs) will likely become an important component of future Internet-networks. Seamless interoperability among heterogeneous networks is a challenging problem as these networks may have very different characteristics. Node diversity may also

Augmenting Data Center Networks with Multi-Gigabit Wireless Links

Daniel Halperin · Srikanth Kandula · Jitendra Padhye · Paramvir Bahl · and David Wetherall · Microsoft Research · and University of Washington

Abstract—The 60-GHz technology that is now emerging has the potential to provide dense and extremely fast connectivity at low cost. In this paper, we explore its use to relieve bottlenecks in over-subscribed data center (DC) networks. By experimenting with prototype equipment, we show that DC environments are well suited to a deployment of 60-GHz links contrary to concerns about interference and link reliability. Using directional antennas, many wireless links can run concurrently at multi-Gbps rates on top-of-rack (ToR) switches. The wire DC network can be used to offload several common wireless problems. By analyzing production traces of DC traffic for four real applications, we show that adding a small amount of network capacity in the form of wireless *flyways* to the wire DC network can improve performance. However, to be of significant value, we find that one hop indirect routing is needed. Informed by our 60-GHz experiments and DC traffic analysis, we present a design that uses DC traffic levels to select and add flyways to the wire DC network. Trace-driven evaluations show that network limited DC applications with predictable traffic overhead running on a 1:2 over-subscribed network can be sped up by 45% in 50% of the cases, with just one wireless device per ToR switch. With one device, in 40% of the cases, the performance is identical to that of a non-over-subscribed network.

Traditional, wired DC networks are time-structured and over-subscribed to keep costs down [15]. For example, a typical DC rack comprises 40 machines connected to a top-of-the-rack (ToR) switch with 1 Gbps links. The ToR is connected to an aggregation switch (to network with other racks) with 10-Gbps links. Thus, the link from the ToR to the aggregation switch can be over-subscribed with a ratio of 1:4. However, each over-subscribed link is a potential hotspot that hinders some DC application. Recent research backs this problem by combining many more links and switches with variants of multipath routing so that the core of the network is no longer over-subscribed [1, 8, 9]. Of course, this benefit comes with large material cost and implementation complexity [17]. Some designs require many wire ties that cabling becomes a challenge [11, 16] and most require "ToR ToR" [12] upgrades to the entire infrastructure.

In prior work [15], we argued instead for a more modest addition of links to relieve hotspots and boost application performance. The links, called *flyways*, add extra capacity to the link used to alleviate hotspots. When the traffic matrix is sparse (i.e., only a few ToR switches are hot), a small number of flyways can significantly improve performance, without the cost of building a fully non-over-subscribed network.

The basic design of a DC network with 60-GHz flyways is as follows. The bare wire network is provisioned for the average case and can be over-subscribed. Each top-of-the-rack (ToR) switch is equipped with one or more 60-GHz wireless devices, with electrically steerable directional antennas. A central controller monitors DC traffic patterns, and switches the beams of the wireless devices to set up flyways between ToR switches that provide added bandwidth as needed.

Other researchers have explored one of two options: optical cables and MEMS switches [7, 30] for increasing flyways. We believe that 60-GHz flyways are an attractive choice because wireless devices simplify DC upgrades, as no wiring changes are needed. Furthermore, 60-GHz technology is likely to become inexpensive as it is commoditized by consumer applications, while optical switches are not. Wireless devices can introduce additional issues as well—for example, with dynamic topology, the network management may become more

Promission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. SIGCOMM '11, August 15–19, 2011, Toronto, Ontario, Canada. Copyright 2011 ACM 978-1-4553-0791-0/11/08...\$10.00.

1062-6860/12/0004-1014\$12.00 © 2012 IEEE

Springer

38

What have people done with ns-3?

- Educational use (from ns-3 wiki)

Using ns-3 in Education

This page is a resource for learning about ns-3 as an educational tool for networking education.

Papers

The [2011 Sigcomm Education workshop](#) had a paper regarding ns-3 use in the classroom:

- [An Open-source and Declarative Approach Towards Teaching Large-scale Networked Systems Programming](#)

Courses using ns-3

The following courses have used ns-3 as courseware or to support projects

- [Georgia Tech. ECE 6110](#) [Dr. George Riley](#), Spring 2013 (also Fall 2011, Fall 2010)
- The University of Kansas [EECS 780](#), [EECS 882](#), and [EECS 983](#) [Dr. James Sterbenz](#), 2010 – 2012
- [UPenn CIS 553/TCOM 512](#) [Dr. Boon Thau Loo](#), Fall 2010
- [Aalto University](#) [Jose Costa-Requena](#) and [Markus Peuhkuri](#), Fall 2011
- [Indian Institute of Technology Bombay](#) [Bhaskaran Raman](#), Autumn 2008
- University of Rijeka
 - [RM2-InfUniRi](#), [Dr. Mario Radovan](#) and [Vedran Miletić](#), Spring 2013, also Spring 2012
 - [RM-RiTeh](#), [Dr. Mladen Tomić](#) and [Vedran Miletić](#), Spring 2013

Other resources

- Lalith Suresh's [Lab Assignments using ns-3](#) page.

Software introduction

- Download the latest release

- `wget http://www.nsnam.org/releases/ns-allinone-3.19.tar.bz2`
 - `tar xjf ns-allinone-3.19.tar.bz2`

- Clone the latest development code

- `hg clone http://code.nsnam.org/ns-3-allinone`

Q. What is "**hg clone**"?

A. Mercurial (<http://www.selenic.com>) is our source code control tool.

Software for ns-3 training

Two versions

1) ns-allinone package

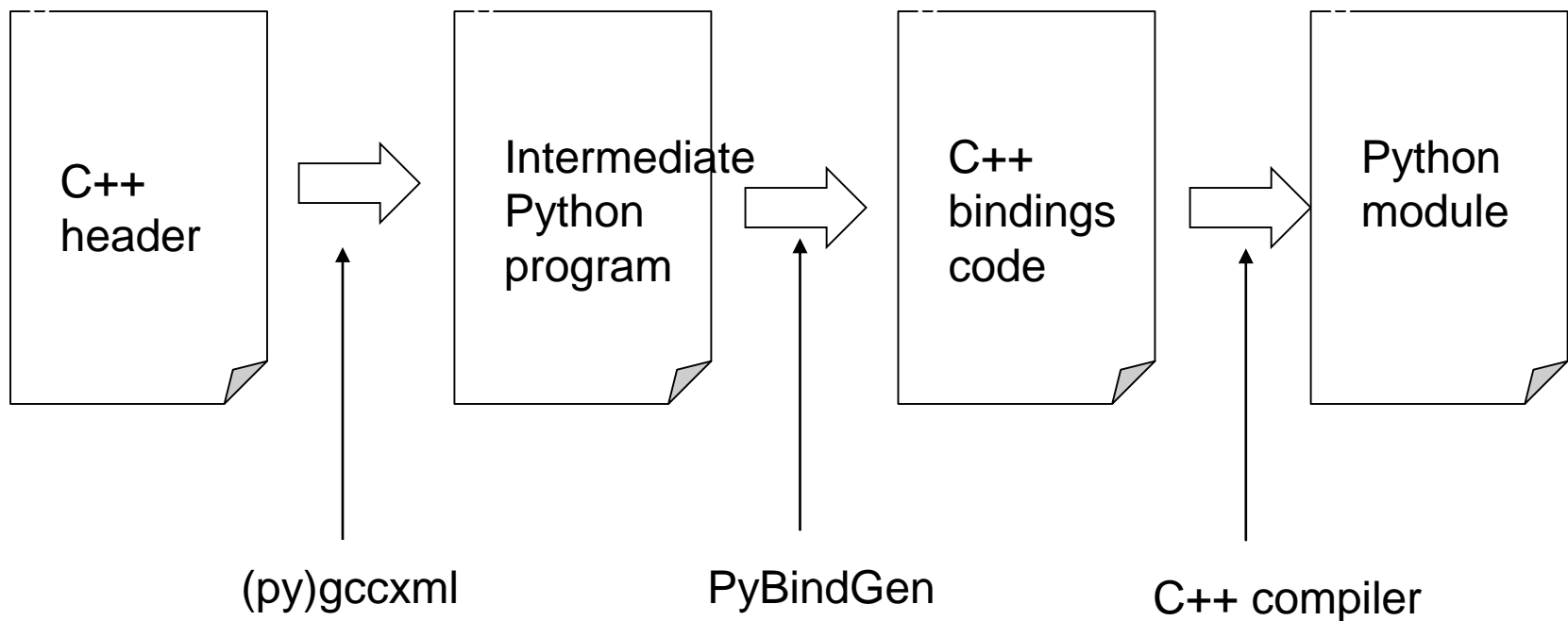
- wget
<https://www.nsnam.org/tutorials/consortium14/ns-allinone-3.20.training.tar.bz2>
- tar xjf ns-allinone-3.20.training.tar.bz2

2) bake package

- wget
<https://www.nsnam.org/tutorials/consortium14/bake-ns-3.20.training.tar.bz2>
- tar xjf bake-ns-3.20.training.tar.bz2

Python bindings

- ns-3 uses a program called PyBindGen to generate Python bindings for all libraries



APIs

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

The screenshot displays the ns-3 Doxygen API documentation for the `ns3::InetSocketAddress` class. On the left, a sidebar titled "NS-3" contains a tree of navigation links: "ns-3 Documentation", "NS-3 Modules", "NS-3 Class List", "NS-3 Class Hierarchy", "Class Members", "NS-3 Graphical Class Hierarchy", "NS-3 Namespace List", "Namespace Members", and "NS-3 Related Pages". The main content area features a breadcrumb trail: "Main Page", "Modules", "Namespaces", "Classes", and "Related Pages". Below this, tabs for "Class List", "Class Hierarchy", and "Class Members" are visible. The title "ns3::InetSocketAddress" is centered, followed by the heading "ns3::InetSocketAddress Class Reference" and a sub-heading "[Address]". The text "an Inet address class [More...](#)" is present, along with the preprocessor directive `#include <inet-socket-address.h>`. A section titled "Collaboration diagram for ns3::InetSocketAddress:" contains a diagram showing a box for `ns3::InetSocketAddress` at the bottom and a box for `ns3::Ipv4Address` above it. A purple arrow labeled `m_ipv4` points from the `ns3::InetSocketAddress` box to the `ns3::Ipv4Address` box. Below the diagram is a "[legend]" link. At the bottom of the page, there are links for "List of all members." and "Public Member Functions".

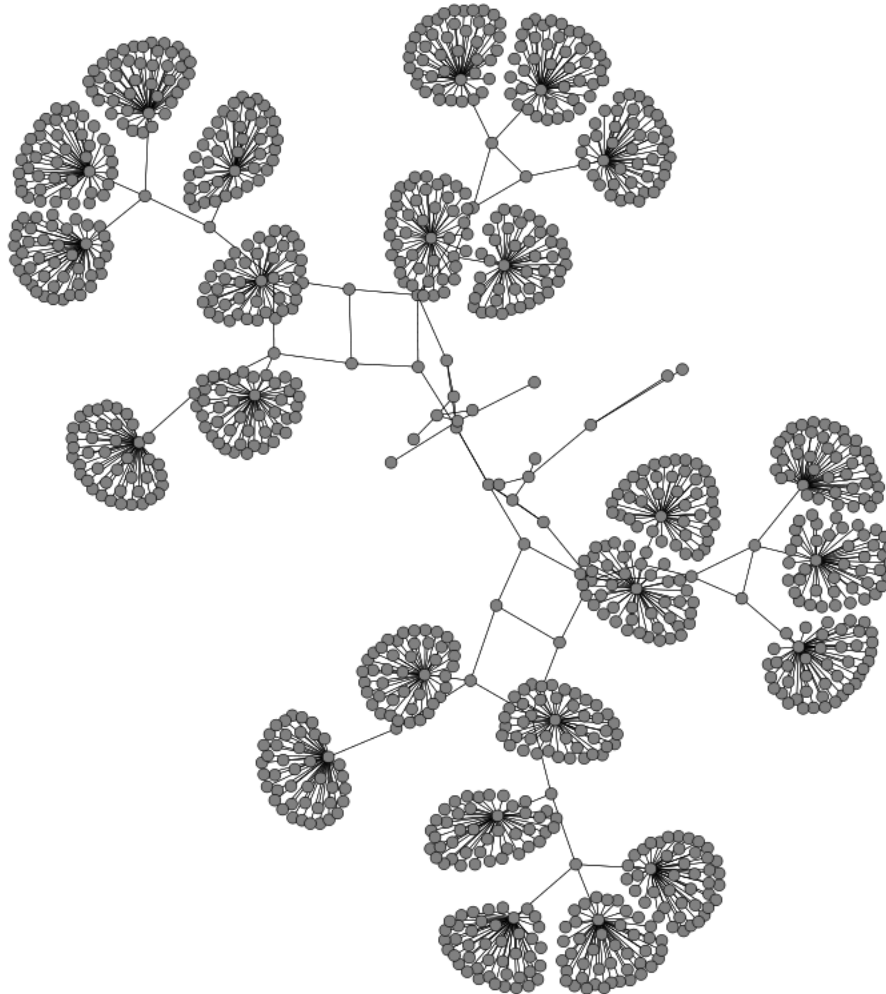
mixed-wireless example

- Placeholder slide
- `./waf --run mixed-wireless`
- `./waf --pyrun examples/wireless/mixed-wireless.py`

PyViz overview

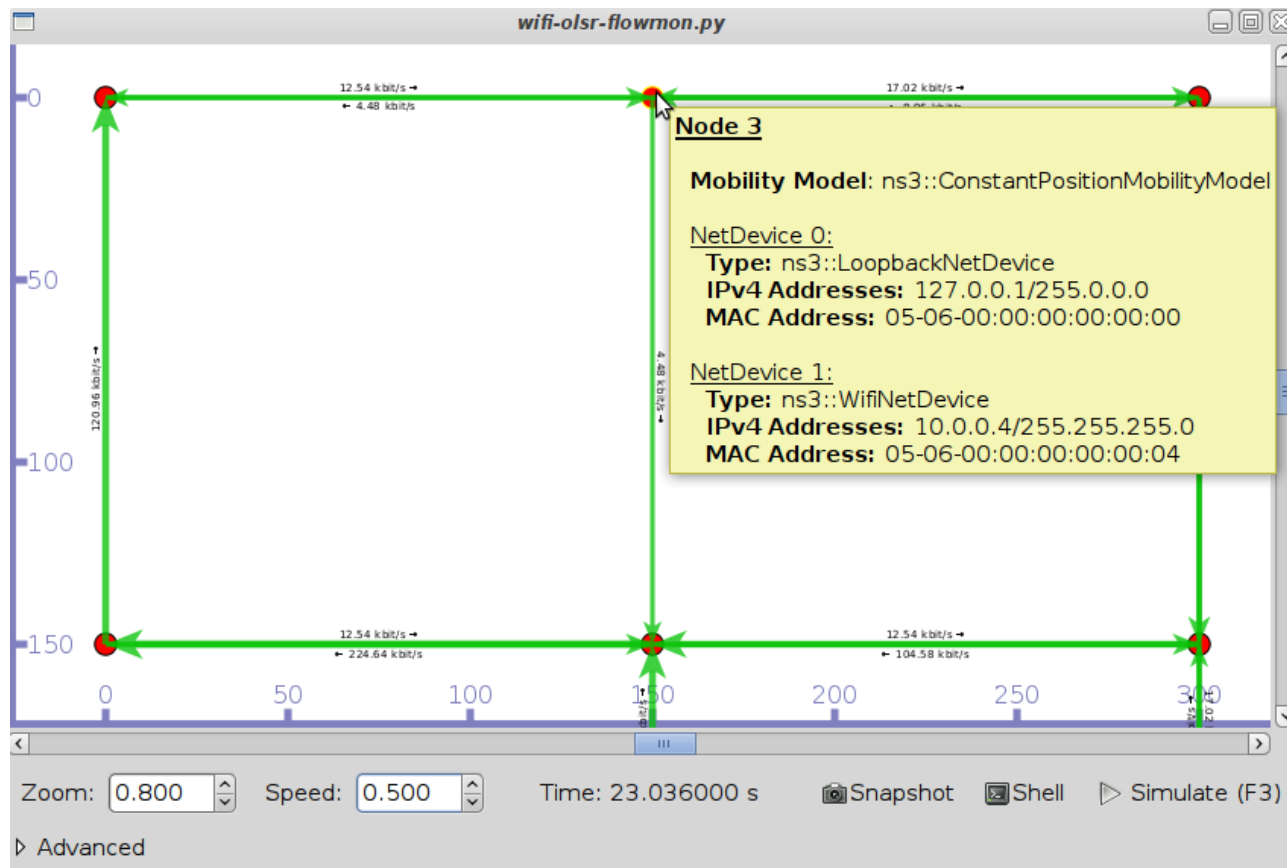
- Developed by Gustavo Carneiro
- Live simulation visualizer (no trace files)
- Useful for debugging
 - mobility model behavior
 - where are packets being dropped?
- Built-in interactive Python console to debug the state of running objects
- Works with Python and C++ programs

Pyviz screenshot (Graphviz layout)



Pyviz and FlowMonitor

- src/flow-monitor/examples/wifi-olsr-flowmon.py



Enabling PyViz in your simulations

- Make sure PyViz is enabled in the build

```
SQLite stats data output      : not enabled (library 'sqlite3' not found)
Tap Bridge                   : enabled
PyViz visualizer              : enabled
Use sudo to set suid bit     : not enabled (option --enable-sudo not selected)
```

- If program supports CommandLine parsing, pass the option
`--SimulatorImplementationType=ns3::VisualSimulatorImpl`
- Alternatively, pass the "--vis" option

FlowMonitor

- Network monitoring framework found in `src/flow-monitor/`
- Goals:
 - detect all flows passing through network
 - stores metrics for analysis such as bitrates, duration, delays, packet sizes, packet loss ratios

G. Carneiro, P. Fortuna, M. Ricardo, "FlowMonitor-- a network monitoring framework for the Network Simulator ns-3," Proceedings of NSTools 2009.

FlowMonitor architecture

- Basic classes
 - FlowMonitor
 - FlowProbe
 - FlowClassifier
 - FlowMonitorHelper
- IPv6 coming in ns-3.20 release

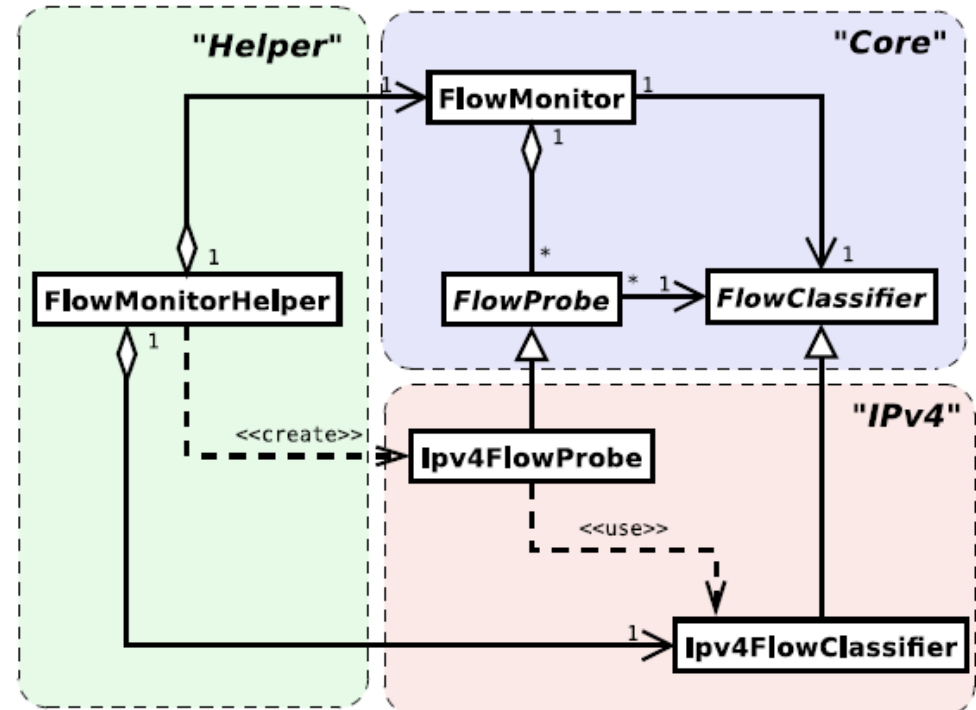
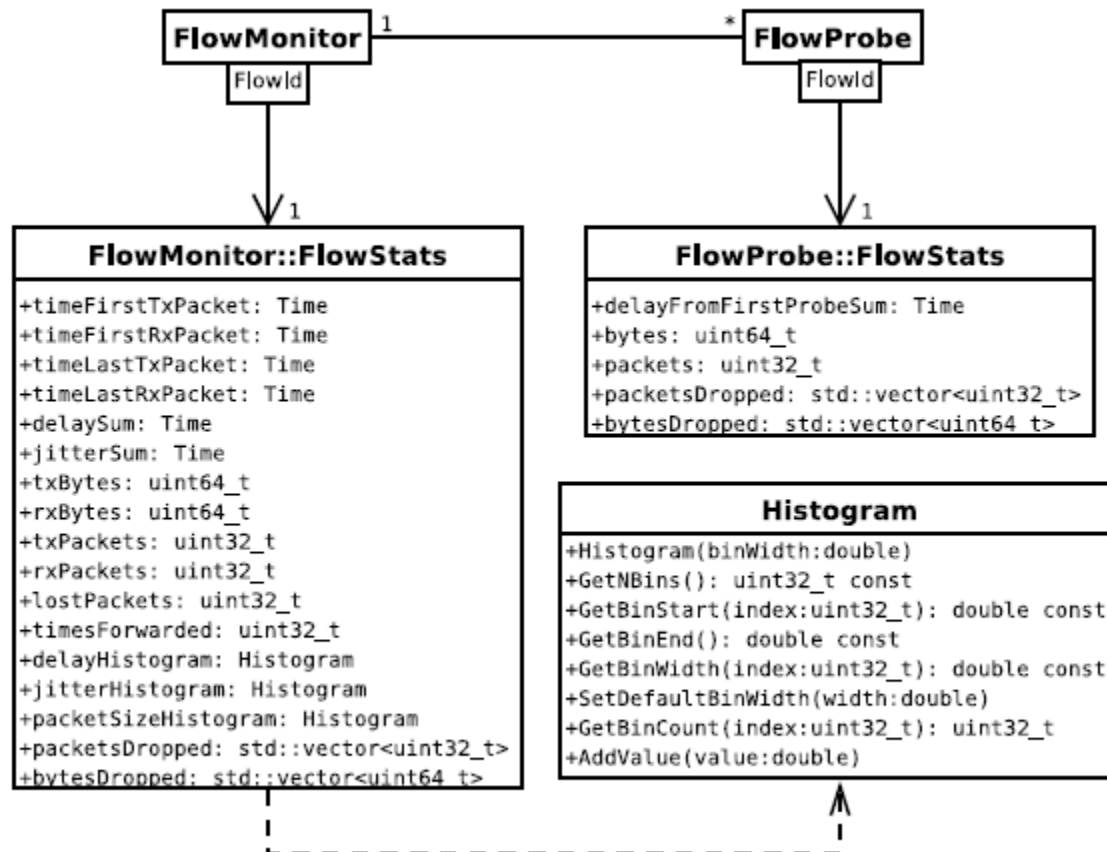


Figure credit: G. Carneiro, P. Fortuna, M. Ricardo, "FlowMonitor-- a network monitoring framework for the Network Simulator ns-3," Proceedings of NSTools 2009.

FlowMonitor statistics

- Statistics gathered



FlowMonitor configuration

- `example/wireless/wifi-hidden-terminal.cc`

```
// 8. Install FlowMonitor on all nodes
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();

// 9. Run simulation for 10 seconds
Simulator::Stop (Seconds (10));
Simulator::Run ();

// 10. Print per flow statistics
monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin (); i != stats.end (); ++i)
{
    // first 2 FlowIds are for ECHO apps, we don't want to display them
    if (i->first > 2)
    {
        Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
        std::cout << "Flow " << i->first - 2 << " (" << t.sourceAddress << " -> " << t.destinationAddress << ")\n";
        std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
        std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
        std::cout << "  Throughput: " << i->second.rxBytes * 8.0 / 10.0 / 1024 / 1024 << " Mbps\n";
    }
}
```

FlowMonitor output

- This program exports statistics to stdout
- Other examples integrate with PyViz

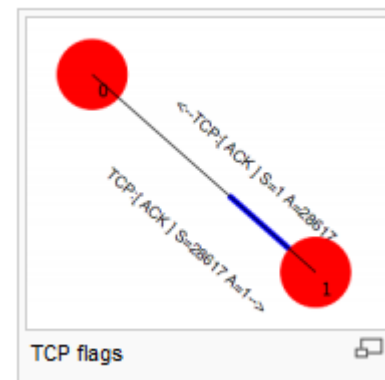
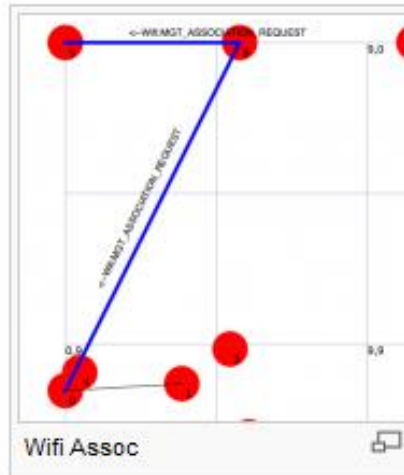
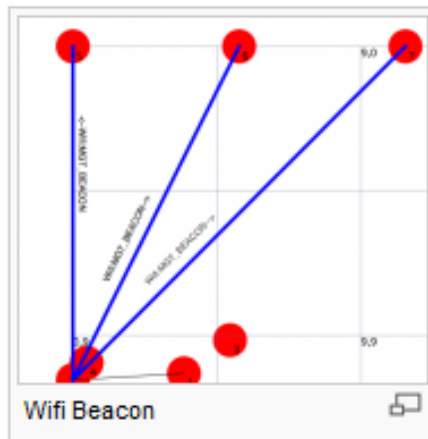
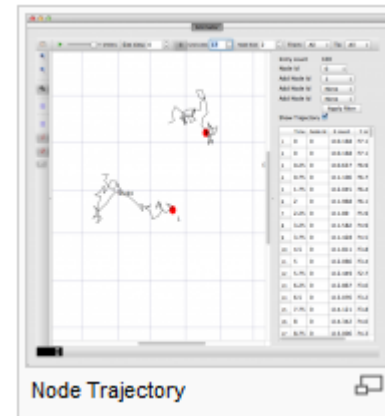
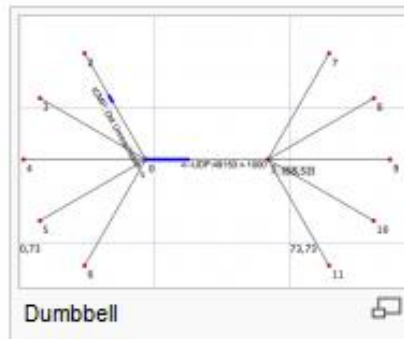
```
Hidden station experiment with RTS/CTS disabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
  Tx Bytes:   3847500
  Rx Bytes:   316464
  Throughput: 0.241443 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Bytes:   3848412
  Rx Bytes:   336756
  Throughput: 0.256924 Mbps
-----
Hidden station experiment with RTS/CTS enabled:
Flow 1 (10.0.0.1 -> 10.0.0.2)
  Tx Bytes:   3847500
  Rx Bytes:   306660
  Throughput: 0.233963 Mbps
Flow 2 (10.0.0.3 -> 10.0.0.2)
  Tx Bytes:   3848412
  Rx Bytes:   274740
  Throughput: 0.20961 Mbps
```

NetAnim

- "NetAnim" by George Riley and John Abraham

No	Time	From Node Id	To Node Id	
1	2.5e-05	0	5	WIFI_MGT_BEACON FromDS: 0 ToDS: 0 DA: 00:00:00:00:00:00
2	2.3e-05	0	6	WIFI_MGT_BEACON FromDS: 0 ToDS: 0 DA: 00:00:00:00:00:00
3	2.5e-05	0	7	WIFI_MGT_BEACON FromDS: 0 ToDS: 0 DA: 00:00:00:00:00:00
4	0.000167033	5	6	WIFI_MGT_ASSOCIATION_REQUEST FromDS: 0 ToDS: 1
5	0.000167033	5	7	WIFI_MGT_ASSOCIATION_REQUEST FromDS: 0 ToDS: 1
6	0.000167033	5	0	WIFI_MGT_ASSOCIATION_REQUEST FromDS: 0 ToDS: 1
7	0.000279066	0	5	WIFI_CTL_ACK RA:00:00:00:00:00:00
8	0.000279066	0	6	WIFI_CTL_ACK RA:00:00:00:00:00:00
9	0.000279066	0	7	WIFI_CTL_ACK RA:00:00:00:00:00:00
10	0.000492193	6	5	WIFI_MGT_ASSOCIATION_REQUEST FromDS: 0 ToDS: 1
11	0.000492193	6	0	WIFI_MGT_ASSOCIATION_REQUEST FromDS: 0 ToDS: 1
12	0.00051414	0	5	WIFI_CTL_ACK RA:00:00:00:00:00:00
13	0.00051414	0	6	WIFI_CTL_ACK RA:00:00:00:00:00:00
14	0.00051414	0	7	WIFI_CTL_ACK RA:00:00:00:00:00:00

Packet Statistics



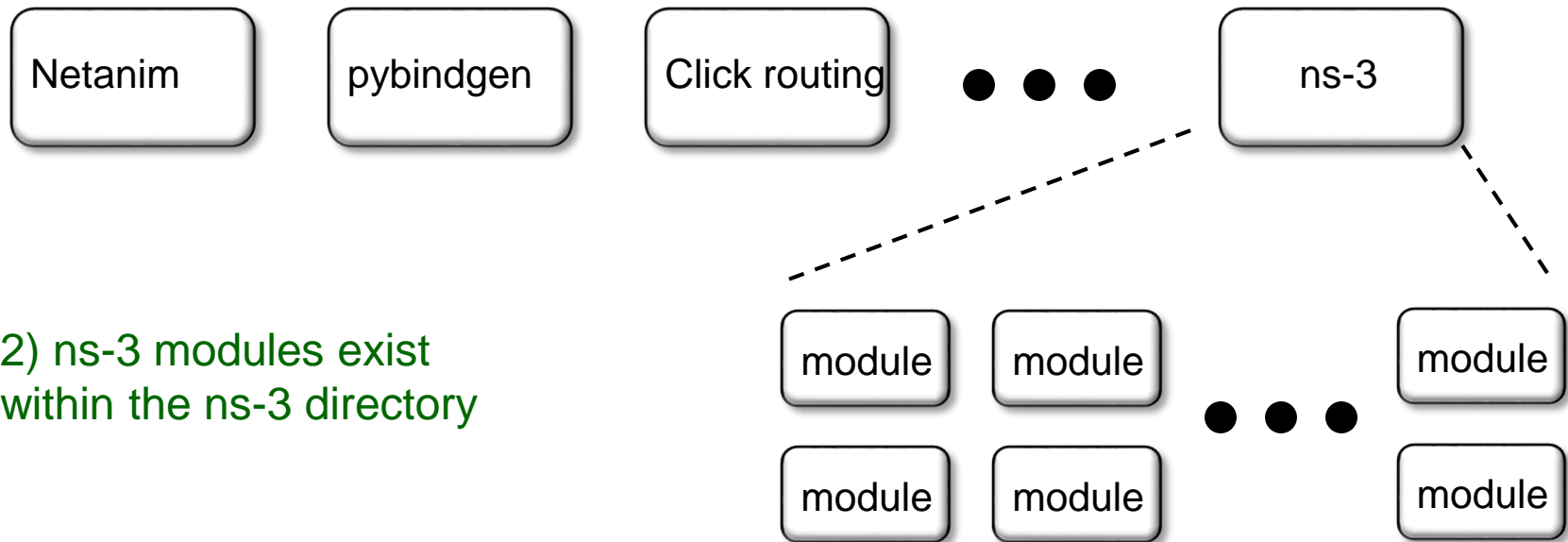
NetAnim key features

- Animate packets over wired-links and wireless-links
 - limited support for LTE traces
- Packet timeline with regex filter on packet meta-data.
- Node position statistics with node trajectory plotting (path of a mobile node).
- Print brief packet-meta data on packets

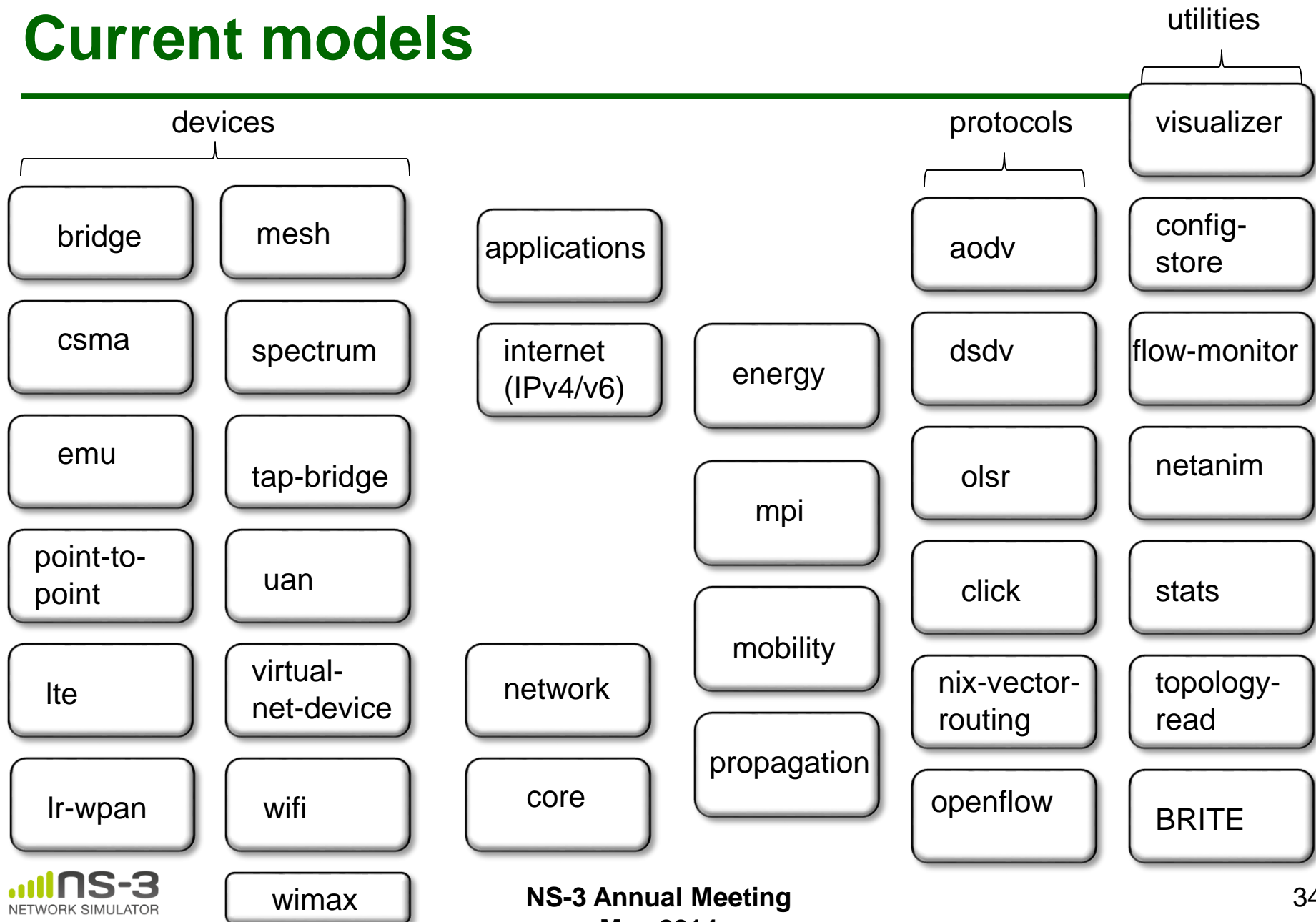
Software organization

- Two levels of ns-3 software and libraries

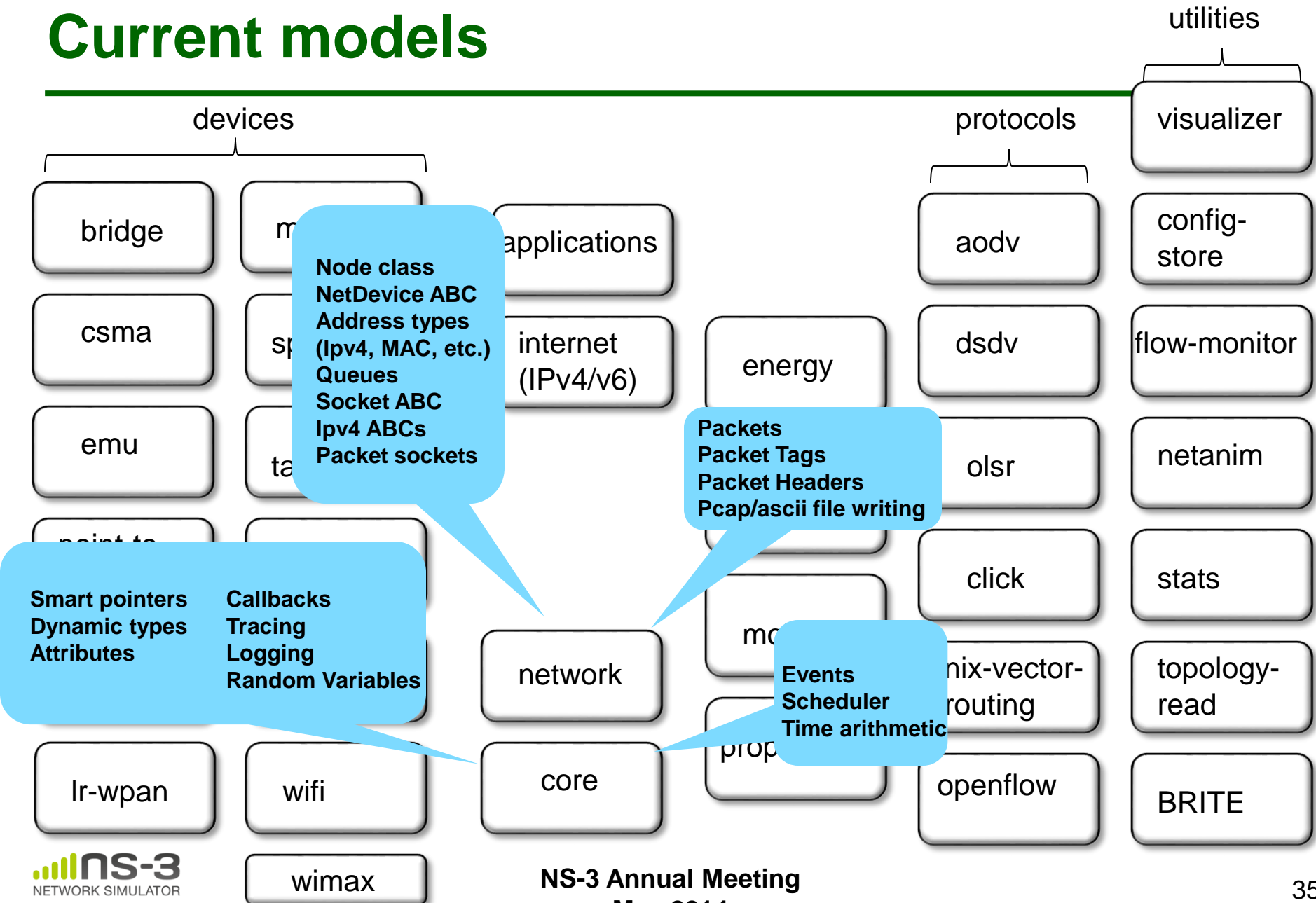
1) Several supporting libraries, not system-installed, can be in parallel to ns-3



Current models



Current models



Module organization

- models/
- examples/
- tests/
- bindings/
- doc/
- wscript

Contributed code and associated projects

The screenshot shows the 'Overall ndnSIM documentation' page. It features a sidebar with navigation links like 'Next topic', 'This Page', and 'Quick search'. The main content area is titled 'Welcome to ndnSIM NS-3 based NDN simulator' and includes a 'Contents' section with a detailed table of contents listing topics such as Introduction, Getting Started, and various Content Store and Forwarding strategy options.

The screenshot displays the 'mptcp-ns3' project page, which implements Multipath TCP on ns-3. The page includes a 'Project description' section explaining the project's focus on developing Multipath TCP for research. It also features a 'Current Status' section with a list of options and congestion control mechanisms. A 'Getting Started' section provides a link to the project's wiki page for instructions on running simulations. The page layout includes a header with the project name and a sidebar with links to project information, code license, and labels.

The screenshot shows the website of the Karlsruhe Institute of Technology (KIT), specifically the 'Decentralized Systems and Network Services Research Group - TM & SCC'. The page features a navigation menu on the left with links to Home, News, Teaching, Staff, Research, and Publications. The main content area is titled 'PhySimWiFi for NS-3' and provides an overview of the project, including contact information and a list of publications and patches.

Integrating other tools and libraries

Gnuplot

- `src/tools/gnuplot.{cc,h}`
- C++ wrapper around gnuplot
- classes:
 - Gnuplot
 - GnuplotDataset
 - Gnuplot2dDataset, Gnuplot2dFunction
 - Gnuplot3dDataset, Gnuplot3dFunction

Enabling gnuplot for your code

- `examples/wireless/wifi-clear-channel-cmu.cc`

```
CommandLine cmd;  
cmd.Parse (argc, argv);  
  
Gnuplot gnuplot = Gnuplot ("clear-channel.eps");  
for (uint32_t i = 0; i < modes.size (); i++)  
{  
    std::cout << modes[i] << std::endl;  
    Gnuplot2dDataset dataset (modes[i]);
```

produce a plot file that
will generate an EPS figure

one dataset per mode

```
    uint32_t pktsRecvd = experiment.Run (wifi, wifiPhy, wifiMac, wifiChannel);  
    dataset.Add (rss, pktsRecvd);  
}  
  
gnuplot.AddDataset (dataset);
```

Add data to dataset

Add dataset to plot

Matplotlib

- `src/core/examples/sample-rng-plot.py`

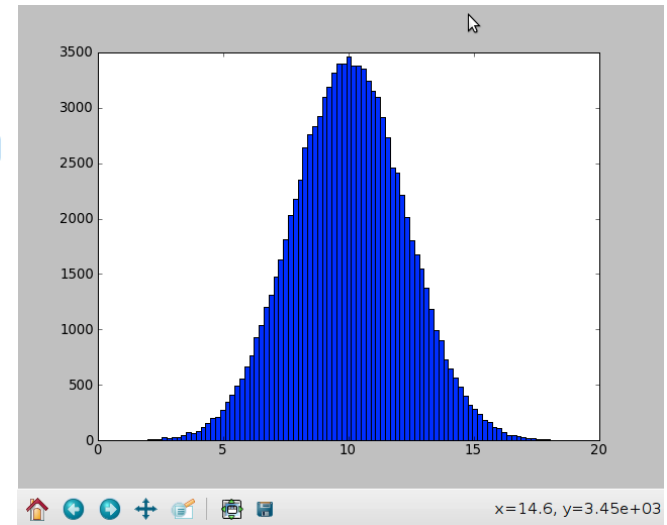
```
# Demonstrate use of ns-3 as a random number generator integrated  
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial
```

```
import numpy as np  
import matplotlib.pyplot as plt  
import ns.core
```

```
# mu, var = 100, 225  
rng = ns.core.NormalVariable(100.0, 225.0)  
x = [rng.GetValue() for t in range(10000)]
```

```
# the histogram of the data  
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)
```

```
plt.title('ns-3 histogram')  
plt.text(60, .025, r'$\mu=100, \sigma=15$')  
plt.axis([40, 160, 0, 0.03])  
plt.grid(True)  
plt.show()
```



Other libraries

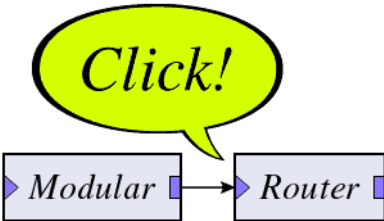
- ns-3 supports additional libraries (click, openflow, nsc)
- ns-3 has optional libraries (libxml2, gsl, mysql)
- both are typically enabled/disabled through the wscript
- users are free to write their own Makefiles or wscripts to do something special

Click Modular Router

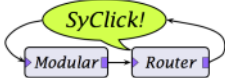
[Click!](#)

[Login](#)

[Show pagesource](#) [Old revisions](#) [Sitemap](#) [Recent changes](#) [Search](#)



The Click Modular Router Project




NEWS (September 24, 2011): **Click 2.0.1 released!**

[SyClick: Symposium on Click Modular Router](#) was **November 23-24, 2009, Ghent, Belgium!** An excellent time was had. Video of the presentations is now available.

This is the [DokuWiki](#) for the Click modular router. Click was originally developed at [MIT](#) with subsequent development at [Mazu Networks](#), [ICIR](#), [UCLA](#), and [Meraki](#).

OpenFlow Switch

Please Note: This website has been archived and is no longer maintained.
See the [Open Networking Foundation](#) for current OpenFlow-related information.



Google™ Custom Search

HomeVideosDocumentsNewsResearchAbout

PageDiscussionView sourceHistory

MPLS with OpenFlow/SDN

Contents [hide]

- 1 Motivation
- 2 Changes to the OpenFlow protocol
- 3 Demos
- 4 Publications
- 5 Talks
- 6 People


Motivation

MPLS networks have evolved over the last 10-15 years to become critically important for ISPs. They provide two key services: traffic engineering in IP networks and L2 or L3 enterprise VPNs. However as carriers deploy MPLS networks, they find that (a) even though the MPLS data plane was meant to be simple, vendors end up supporting MPLS as an additional feature on complex, energy hogging, expensive core routers; and (b) the IP/MPLS control plane has become exceedingly complex with a wide variety of protocols tightly intertwined with the associated data-plane mechanisms.


Quick Navigation

- » [OpenFlow Specs](#)
- » [Bug Tracking](#)
- » [Wiki](#)
- » [Legal](#)
- » [Log in](#)

OpenFlow White Paper

 [Download the OpenFlow Whitepaper \(PDF\)](#)

OpenFlow Specification

 [Download v1.1.0 Implemented \(PDF\)](#)

Wiki Tools


Personal tools

- » [Log in](#)

Navigation

- » [OpenFlow.org](#)

CORE emulator




Networks and Communication Systems Branch

[Focus Areas](#) | [Projects](#) | [Products](#) | [Organization](#)


/ NRL / ITD / NCS / Common Open Research Emulator (CORE)

Common Open Research Emulator (CORE)

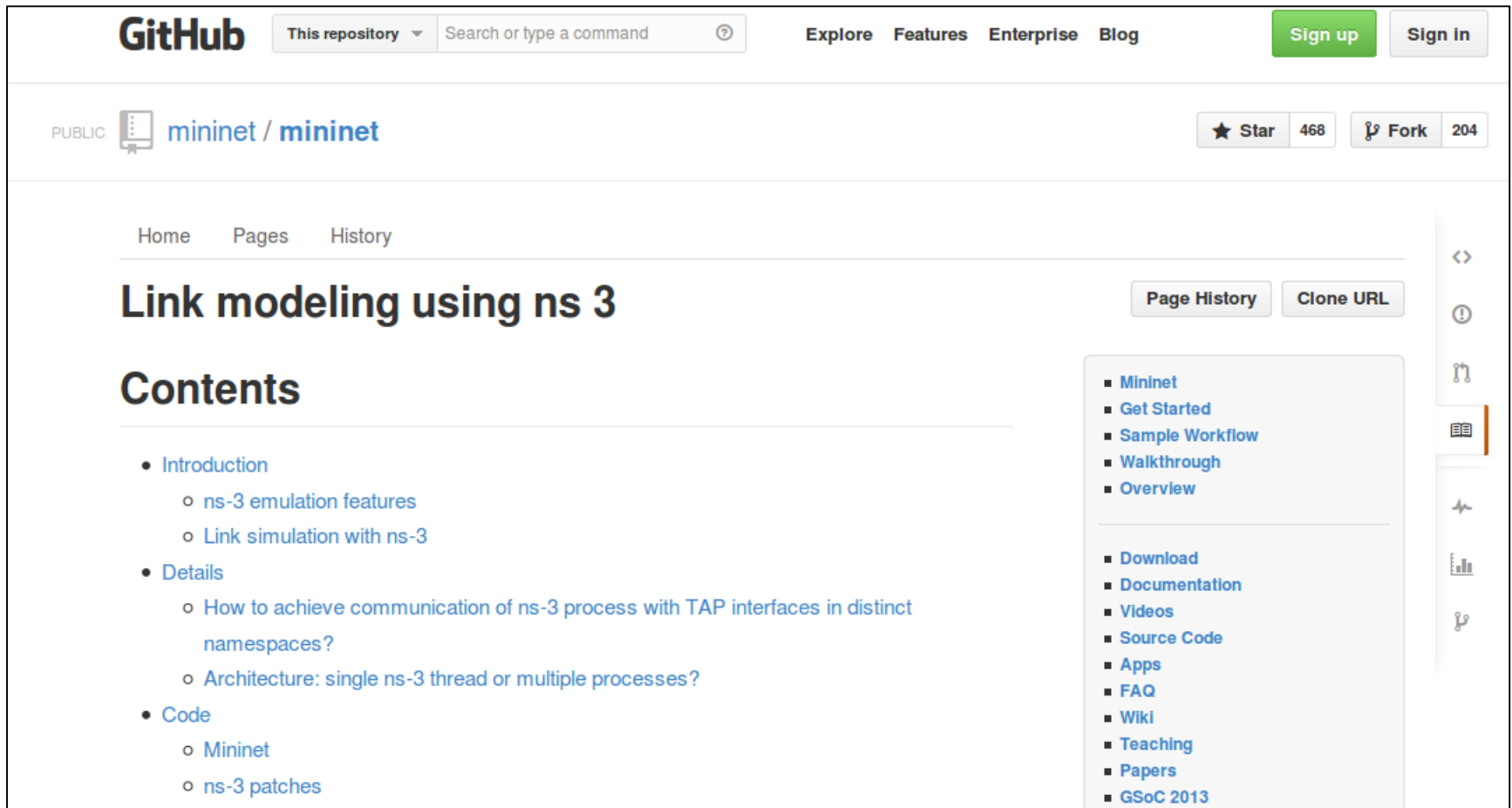
The Common Open Research Emulator (CORE) is a tool for emulating networks on one or more machines. You can connect these emulated networks to live networks. CORE consists of a GUI for drawing topologies of lightweight virtual machines, and Python modules for scripting network emulation.



[NCS Home](#)
[Focus Areas](#)
[Projects](#)
[Products](#)
[Organization](#)



mininet emulator



The screenshot shows the GitHub repository page for `mininet / mininet`. The repository is public and has 468 stars and 204 forks. The current branch is `Link modeling using ns 3`. The page displays the repository's contents, including a table of contents with sections like Introduction, Details, and Code. A sidebar on the right lists various resources such as Mininet, Get Started, Sample Workflow, Walkthrough, Overview, Download, Documentation, Videos, Source Code, Apps, FAQ, Wiki, Teaching, Papers, and GSoC 2013.

GitHub This repository Search or type a command ? Explore Features Enterprise Blog Sign up Sign in

PUBLIC mininet / mininet ★ Star 468 Fork 204

Home Pages History

Link modeling using ns 3

Page History Clone URL

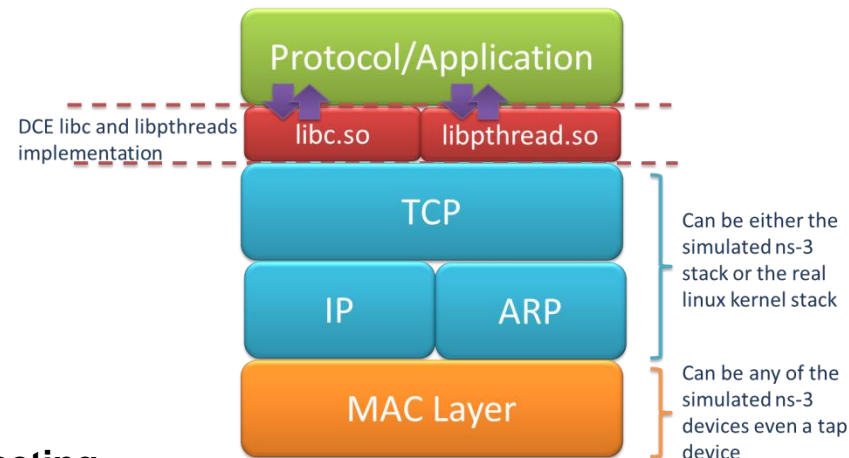
Contents

- Introduction
 - ns-3 emulation features
 - Link simulation with ns-3
- Details
 - How to achieve communication of ns-3 process with TAP interfaces in distinct namespaces?
 - Architecture: single ns-3 thread or multiple processes?
- Code
 - Mininet
 - ns-3 patches

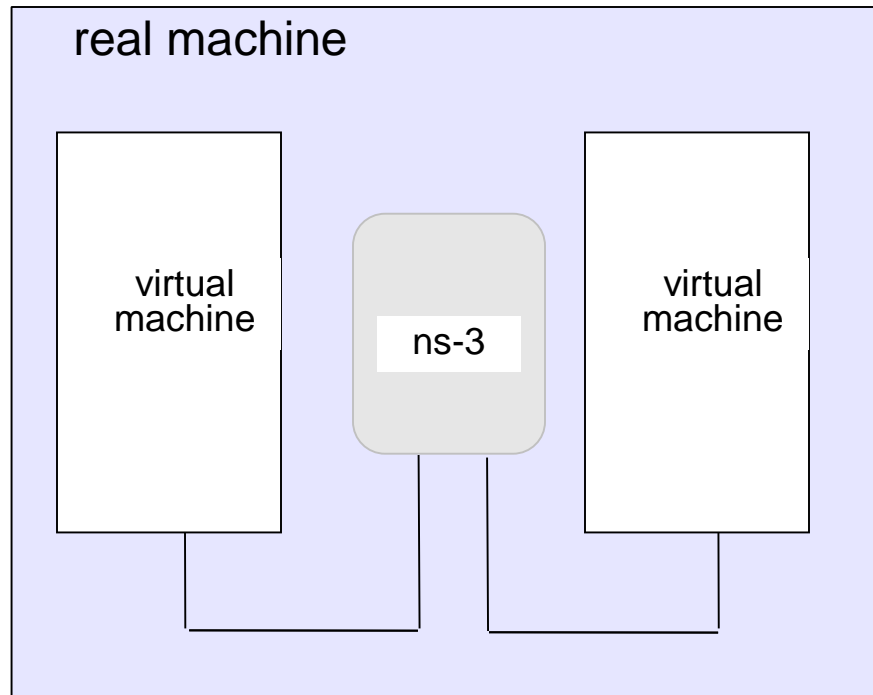
- Mininet
- Get Started
- Sample Workflow
- Walkthrough
- Overview
- Download
- Documentation
- Videos
- Source Code
- Apps
- FAQ
- Wiki
- Teaching
- Papers
- GSoC 2013

Direct Code Execution

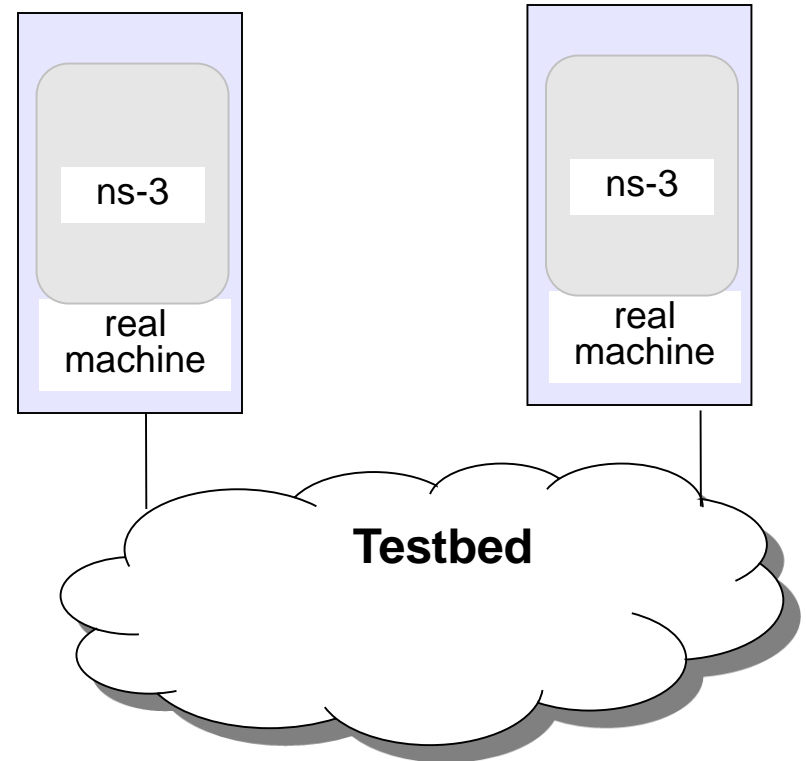
- DCE/ns-3 framework requires the virtualization of a series of services
 - Multiple isolated instances of the same protocol on the same machine
- System calls are captured and treated by DCE
- Network stack protocols calls are captured and redirected
- To perform its work DCE re-implement the Linux program loader and parts of *libc* and *libpthread*



ns-3 emulation modes



1) ns-3 interconnects real or virtual machines



2) testbeds interconnect ns-3 stacks

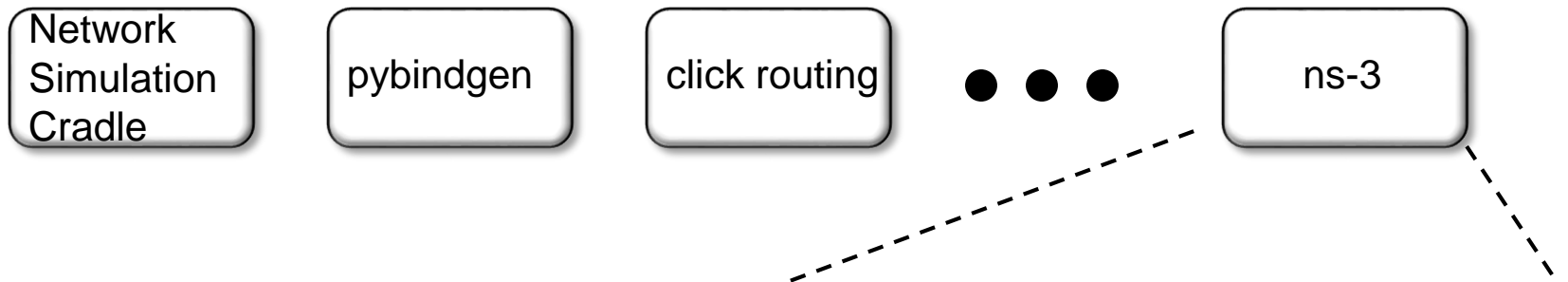
Various hybrids of the above are possible

ns-3 build systems

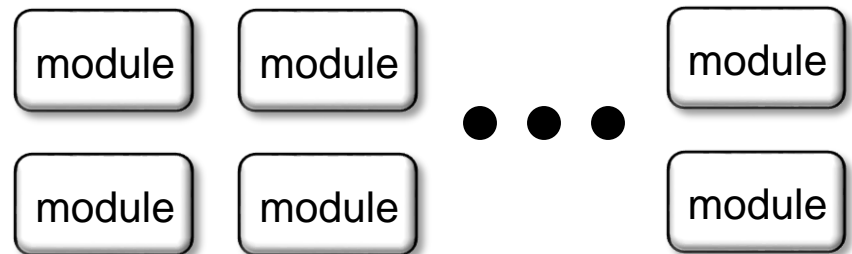
Software building

- Two levels of ns-3 build

1) **bake** (a Python-based build system to control an ordered build of ns-3 and its libraries)



2) **waf**, a build system written in Python



3) **build.py** (a custom Python build script to control an ordered build of ns-3 and its libraries) <--- may eventually be deprecated

ns-3 uses 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/>
- For those familiar with autotools:
 - `configure` → `./waf configure`
 - `make` → `./waf build`

waf configuration

- Key waf configuration examples

```
./waf configure  
--enable-examples  
--enable-tests  
--disable-python  
--enable-modules
```

- Whenever build scripts change, need to reconfigure

Demo: `./waf --help`
`./waf configure --enable-examples --enable-tests --enable-modules='core'`

Look at: `build/c4che/_cache.py`

wscript example

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

def build(bld):
    obj = bld.create_ns3_module('csma', ['network', 'applications'])
    obj.source = [
        'model/backoff.cc',
        'model/csma-net-device.cc',
        'model/csma-channel.cc',
        'helper/csma-helper.cc',
    ]
    headers = bld.new_task_gen(features=['ns3header'])
    headers.module = 'csma'
    headers.source = [
        'model/backoff.h',
        'model/csma-net-device.h',
        'model/csma-channel.h',
        'helper/csma-helper.h',
    ]

    if bld.env['ENABLE_EXAMPLES']:
        bld.add_subdirs('examples')

    bld.ns3_python_bindings()
```

waf build

- Once project is configured, can build via `./waf build` or `./waf`
- waf will build in parallel on multiple cores
- waf displays modules built at end of build

Demo: `./waf build`

Look at: `build/` libraries and executables

Running programs

- `./waf shell` provides a special shell for running programs
 - Sets key environment variables

```
./waf --run sample-simulator
```

```
./waf --pyrun src/core/examples/sample-simulator.py
```

Build variations

- Configure a build type is done at waf configuration time
- debug build (default): all asserts and debugging code enabled

```
./waf -d debug configure
```

- optimized

```
./waf -d optimized configure
```

- static libraries

```
./waf --enable-static configure
```


Controlling the modular build

- One way to disable modules:
 - `./waf configure --enable-modules='a','b','c'`
- The `.ns3rc` file (found in `utils/` directory) can be used to control the modules built
- Precedence in controlling build
 - 1) command line arguments
 - 2) `.ns3rc` in ns-3 top level directory
 - 3) `.ns3rc` in user's home directory

Demo how `.ns3rc` works

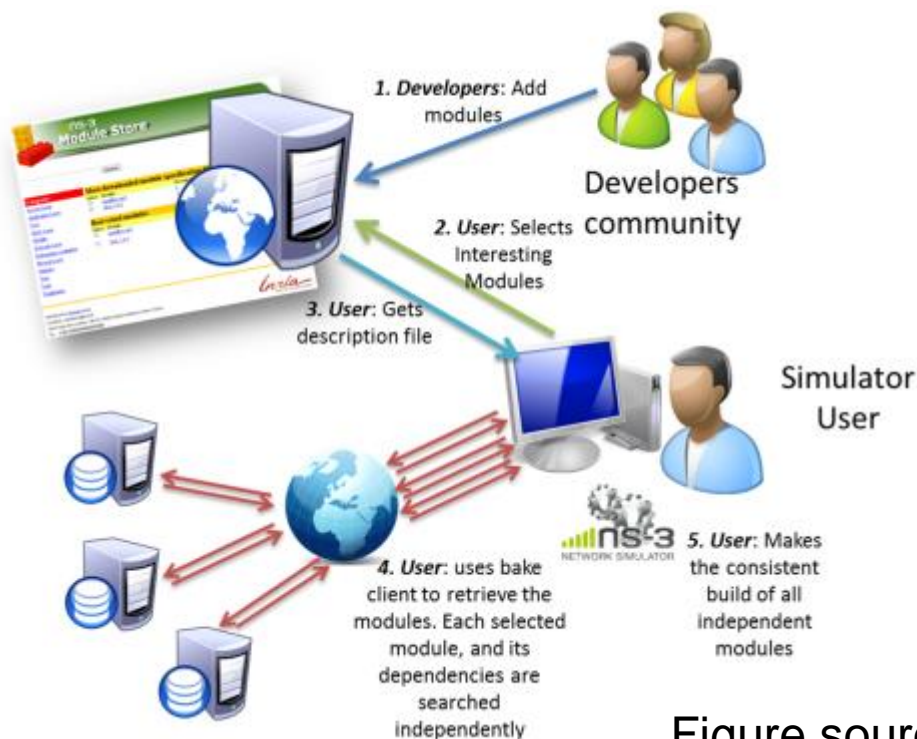
Building without wscript

- The scratch/ directory can be used to build programs without wscripts

Demo how programs can be built without wscripts

bake overview

- Open source project maintains a (more stable) core
- Models migrate to a more federated development process



"bake" tool (Lacage and Camara)

Components:

- build client
- "module store" server
- module metadata

Figure source: Daniel Camara

Placeholder slide for demoing bake

Demo: `./waf build`

Look at: `build/` libraries and executables