# Network simulation using ns-3

Source material: Walid Younes's slides at

https://www.nsnam.org/tutorials/ns-3-tutorial-Walid-Younes.pdf

# Chapter 1 Introduction

**Source material:**

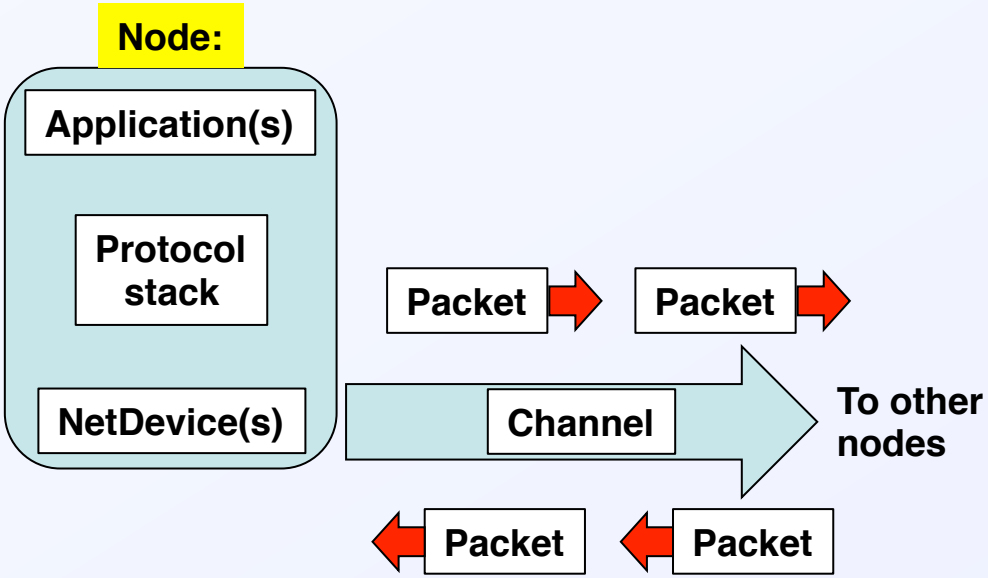
- https://www.nsnam.org/docs/release/3.28/tutorial/html/introduction.html
- https://www.nsnam.org/tutorials/ns-3-tutorial-Walid-Younes.pdf
- https://www.nsnam.org/wiki/AnnualTraining2016/ns-3-training-Monday-session.pptx

# Why simulate?

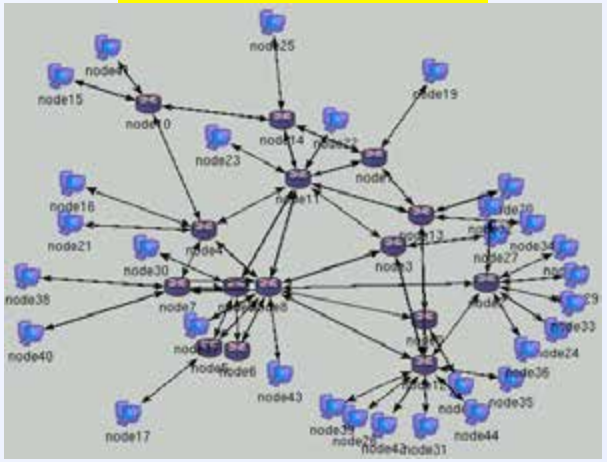
- Arguments against simulation
  - Simulation will never be as good as real thing
  - PC's are cheap and getting cheaper ⇒ test bed experiments
- Arguments for simulation
  - Reproducibility
  - Easier to set up, deploy, maintain
  - Can investigate things that do not yet exist
  - Can scale to larger problem size

**Best is combination of real test beds + simulation**

# What are we actually simulating?




Network topology:



- Nodes: hosts, routers, servers,…
- Applications: generate and consume traffic in network
- Protocols: broker connections, access, addressing, routing,…
- NetDevices: e.g., ethernet & wireless cards
- Channels: transmission medium (cable, EM waves,…)
- Packets: make up network traffic

# Are there some network simulators out there?

**Quite a few actually, for ex:**

NS-2

J-Sim

TOSSIM

GloMoSim

Agent J

SHARPE

Mininet

OPNET

Netsim

ns-3

Qual Net

SPNP

OMNET++

INSANE

TrSim

Query Cycle

Maisie

Neurogrid

**We will focus on ns-3**

# What have people done with ns-3?

- ~145 ns-3 publications for 2017（IEEE digital library）
- ~2579 publications in 2017 (ACM digital library )
  - search of 'ns-3 simulator' on IEEE and ACM digital libraries

1014 IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 20, NO. 6, DECEMBER 2012

FSR: Formal Analysis and Implementation Toolkit for Safe Interdomain Routing

Anduo Wang, Limin Jia, *Member, IEEE*, Wenchao Zhou, Yiqing Ren, Boon Thau Loo, Jennifer Rexford, *Senior Member, IEEE*, Vivek Nigam, Andre Scedrov, 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 homegrown 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 functions from the same algebraic representation of routing policy. We show that routing algebra has a natural translation to both *integer constraints* (to perform safety analysis with SMT solvers) and *declarative programs* (to generate distributed implementations). Our extensive experiments with realistic topologies and policies show how *FSR* can detect problems in an autonomous system's (AS's) iBGP configuration, prove sufficient conditions for Border Gateway Protocol (BGP) safety, and empirically evaluate convergence time.

*Index Terms*—Communications technology, declarative networking, formal analysis, routing algebra.

I. Introduction

THE INTERNET'S global routing system does not necessarily converge, depending on how the Border Gateway Protocol (BGP) policies of individual networks are configured. Since protocol oscillations cause serious performance disruptions and router overhead, 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 guidelines that, if universally adopted by ISPs, ensure global

Manuscript received May 23, 2011; accepted January 21, 2012; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Z. M. Mao. Date of publication March 14, 2012; date of current version December 13, 2012. This work was supported in part by the NSF under Grants CCF-0820208, CNS-0830949, CNS-0845552, CNS-1040672, CPS-0932397, III-0812270, and TC-0905607; the AFOSR, under Grants FA9550-08-1-0352 and FA9550-09-1-0643; the ONR, under Grants N00014-09-1-0770 and N00014-11-1-0555; a gift from Cisco Systems; and the A. von Humboldt Foundation.

A. Wang, W. Zhou, Y. Ren, B. T. Loo, and A. Scedrov are with the University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: anduo@seas.upenn.edu; wenchaoz@seas.upenn.edu; yiqing@seas.upenn.edu; boonloo@seas.upenn.edu; scedrov@math.upenn.edu).

L. Jia is with Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: liminjia@cmu.edu).

J. Rexford is with Princeton University, Princeton, NJ 08540 USA (e-mail: jrex@cs.princeton.edu).

V. Nigam is with the Computer Science Department, Ludwig-Maximilians University of Munich, Munich 80539, Germany.

C. Talcott is with the SRI International, Menlo Park, CA 94025 USA (e-mail: clt@csl.sri.com).

Digital Object Identifier 10.1109/TNET.2012.2187924

1063-6692/$31.00 © 2012 IEEE



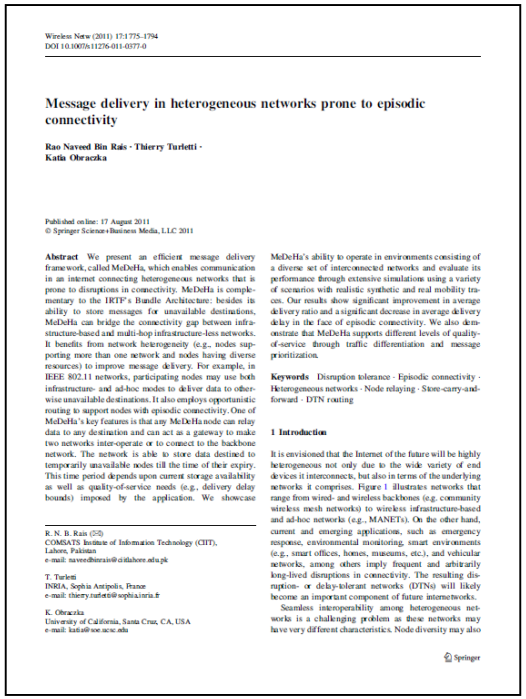

Fig. 1. *FSR* architecture.

safety [4], [8]–[11], [33]. While our understanding of BGP safety has improved dramatically in the past decade, each research study still proceeds independently—manually 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 (iBGP) 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 unify research in *routing algebras* [13], [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 algebra [13], [36] allow researchers and network operators to express policy configurations in an abstract algebraic form. These configurations can be anything from high-level *policy guidelines* (e.g., proposed constraints that a researcher wants to study) or a completely specified *policy instance* [e.g., an iBGP configuration or a multi-autonomous-system (AS) network that an operator wants to analyze]. Router configuration files 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 Netw (2011) 17:1775–1794
DOI 10.1007/s11276-011-0377-0

Message delivery in heterogeneous networks prone to episodic connectivity

Rao Naveed Bin Rais · Thierry Turletti · Katia Obraczka

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

**Abstract** We present an efficient message delivery framework, called MeDeHa, which enables communication in an internet connecting heterogeneous networks that is prone to disruptions in connectivity. MeDeHa is complementary to the IRTF's Bundle Architecture: besides its ability to store messages for unavailable destinations, MeDeHa 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 modes to deliver data to otherwise unavailable destinations. It also employs opportunistic routing to support nodes with episodic connectivity. One of MeDeHa's key features is that any MeDeHa node can relay data to any destination and can act as a gateway to make two networks inter-operate or to connect to the backbone network. The network is able to store data destined to temporarily unavailable nodes till 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 MeDeHa'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 MeDeHa 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 interconnects, but also in terms of the underlying networks 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 arbitrarily long-lived disruptions in connectivity. The resulting disruption- or delay-tolerant networks (DTNs) will likely become an important component of future internetworks.

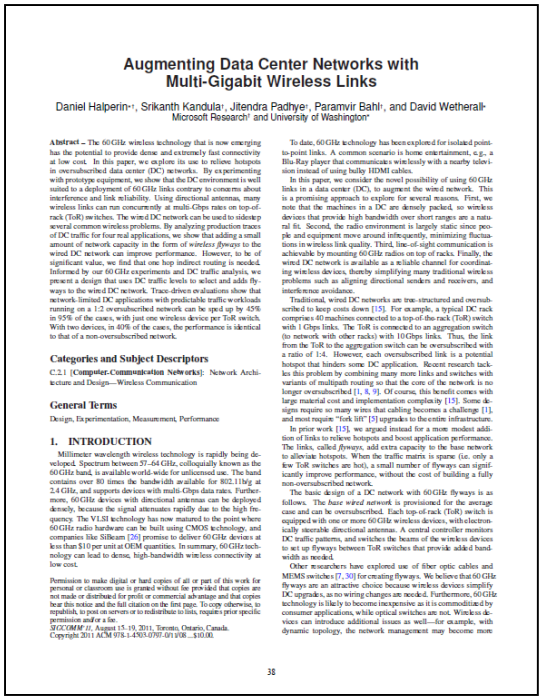
Seamless interoperability among heterogeneous networks is a challenging problem as these networks may have very different characteristics. Node diversity may also

R. N. B. Rais (✉)
COMSATS Institute of Information Technology (CIIT), Lahore, Pakistan
e-mail: naveedbinrais@ciitlahore.edu.pk

T. Turletti
INRIA, Sophia Antipolis, France
e-mail: thierry.turletti@sophia.inria.fr

K. Obraczka
University of California, Santa Cruz, CA, USA
e-mail: katia@soe.ucsc.edu

Springer



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 wireless 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 hotspots in oversubscribed data center (DC) networks. By experimenting with prototype equipment, we show that the DC environment is 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 wired DC network can be used to sidestep 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 wired 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 adds flyways to the wired DC network. Trace-driven evaluations show that network-limited DC applications with predictable traffic workloads running on a 1:2 oversubscribed network can be sped up by 45% in 95% of the cases, with just one wireless device per ToR switch. With two devices, in 40% of the cases, the performance is identical to that of a non-oversubscribed network.

**Categories and Subject Descriptors**
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—Wireless Communication

**General Terms**
Design, Experimentation, Measurement, Performance

1. INTRODUCTION

Millimeter wavelength wireless technology is rapidly being developed. Spectrum between 57–64 GHz, colloquially known as the 60 GHz band, is available world-wide for unlicensed use. The band contains over 80 times the bandwidth available for 802.11b/g at 2.4 GHz, and supports devices with multi-Gbps data rates. Furthermore, 60 GHz devices with directional antennas can be deployed densely, because the signal attenuates rapidly due to the high frequency. The VLSI technology has now matured to the point where 60 GHz radio hardware can be built using CMOS technology, and companies like Silicam [26] promise to deliver 60 GHz devices at less than $10 per unit at OEM quantities. In summary, 60 GHz technology can lead to dense, high-bandwidth wireless connectivity at low cost.


*SIGCOMM'11*, August 15–19, 2011, Toronto, Ontario, Canada.


To date, 60 GHz technology has been explored for isolated point-to-point links. A common scenario is home entertainment, e.g., a Blu-Ray player that communicates wirelessly with a nearby television instead of using bulky HDMI cables.

In this paper, we consider the novel possibility of using 60 GHz links in a data center (DC), to augment the wired network. This is a promising approach to explore for several reasons. First, we note that the machines in a DC are densely packed, so wireless devices that provide high bandwidth over short ranges are a natural fit. Second, the radio environment is largely static since people and equipment move around infrequently, minimizing fluctuations in wireless link quality. Third, line-of-sight communication is achievable by mounting 60 GHz radios on top of racks. Finally, the wired DC network is available as a reliable channel for coordinating wireless devices, thereby simplifying many traditional wireless problems such as aligning directional senders and receivers, and interference avoidance.

Traditional, wired DC networks are tree-structured and oversubscribed 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 oversubscribed with a ratio of 1:4. However, each oversubscribed link is a potential hotspot that hinders some DC application. Recent research tackles this problem by combining many more links and switches with variants of multipath routing so that the core of the network is no longer oversubscribed [1, 8, 9]. Of course, this benefit comes with large material cost and implementation complexity [13]. Some designs require so many wires that cabling becomes a challenge [1], and most require "fork lift" [5] 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 base network 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-oversubscribed network.

The basic design of a DC network with 60 GHz flyways is as follows. The *base wired network* is provisioned for the average case and can be oversubscribed. Each top-of-rack (ToR) switch is equipped with one or more 60 GHz wireless devices, with electronically 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 use of fiber optic cables and MEMS switches [7, 30] for creating 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

# Examples of recent publications

- P. Fazio, D. F. Rango, and C. Sottile, “A Predictive Cross-Layered Interference Management in a Multichannel MAC with Reactive Routing in VANET,” IEEE Transactions on Mobile Computing, vol. 15, iss. 8, pp. 1850-1862, 2016.
- H. Yu, N. Yao, T. Wang, G. Li, Z. Gao, and G. Tan, “WDFAD-DBR: Weighting depth and forwarding area division DBR routing protocol for UASNs,” Ad Hoc Networks, vol. 37, pp. 256-282, 2016
- D. Kim, J. Kim, C. Moon, J. Choi, and I. Yeom, “Efficient content delivery in mobile ad-hoc networks using CCN,” Ad Hoc Networks, vol. 36, pp. 81-99, 2016.
- F. Aalamifar and L. Lampe, “Optimized WiMAX Profile Configuration for Smart Grid Communications,” IEEE Transactions on Smart Grid, vol. PP, iss. 99, pp. 1-10, 2016.

# What have people done with ns-3?

- Educational use (from ns-3 wiki)

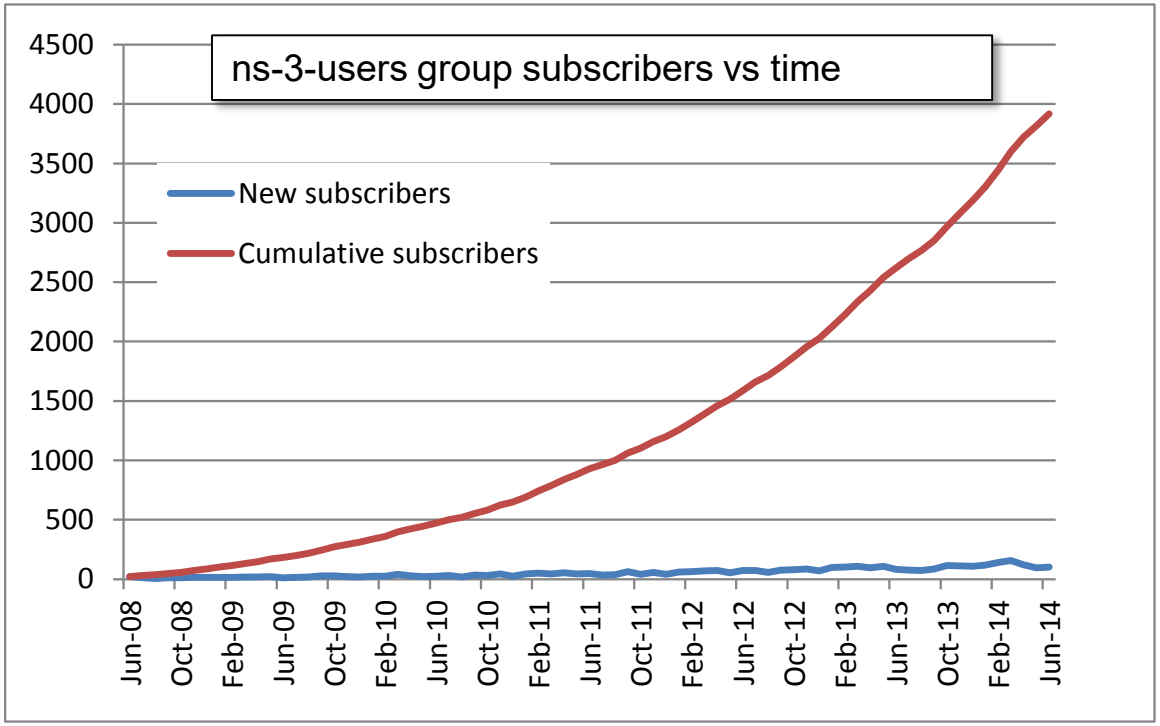
## Courses using ns-3

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

- Georgia Tech. ECE 6110 Dr. George Riley, Spring 2013, Fall 2011, and Fall 2010
- The University of Kansas
  - EECS 780 Dr. James Sterbenz, Fall 2017, Fall 2016, Fall 2015, Fall 2014, Spring 2013, Spring 2012, Spring 2011, Spring 2010, Spring 2009
  - EECS 882 Dr. James Sterbenz, Spring 2016, Fall 2013, Fall 2011, Fall 2009
  - EECS 983 Dr. James Sterbenz, Spring 2014, Spring 2012, Spring 2010
- University of Pennsylvania CIS 553/TCOM 512 Dr. Boon Thau Loo, Spring 2013, also Fall 2010
- Aalto University Jose Costa-Requena and Markus Peuhkuri, Fall 2011
- Indian Institute of Technology Bombay Bhaskaran Raman, Autumn 2011, Autumn 2010, Autumn 2009, and Autumn 2008
- University of Rijeka
  - RM2-Inf, Dr. Mario Radovan and Dr. Vedran Miletić, Spring 2015, Spring 2014, Spring 2013, and Spring 2012
  - RM-RiTeh, Dr. Mladen Tomić and Dr. Vedran Miletić, Spring 2014 and Spring 2013
- Queen's University CISC 834 Dr. Hossam S. Hassanein, Ramy Atawia and Hisham Farahat, Winter 2015, Fall 2013, and Fall 2012

# Statistics (Jan 2018)

- 8183 subscribers to ns-3-users
- 1560 subscribers to ns-developers
- ~ 23 maintainers
- ~ 220 authors/contributors
- 72,000 ns-3 source downloads in 2016

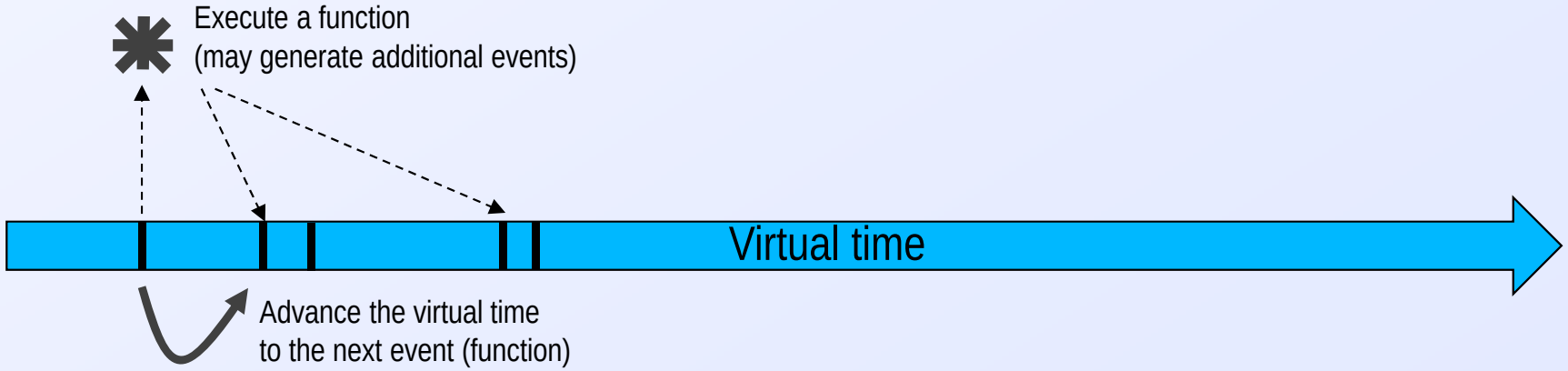


# What is ns-3? What makes it special?

- Discrete event simulator
  - Events model packet transmission, receipt, timers, etc…
  - Events are maintained in time-ordered events list
- Aimed at research and education
- Written entirely in C++ (single language ⇒ easier to debug user code)
- Open-source ⇒ many contributors/maintainers ⇒ longevity of the project
- Models are close to real world
  - Easier to execute real-life codes
  - Can interact with real-world packets
  - Aligned with input/output standards (pcap traces)
- Modular and well-documented

# 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() executes a single-threaded event list
- Simulation stops at specific time or when events end



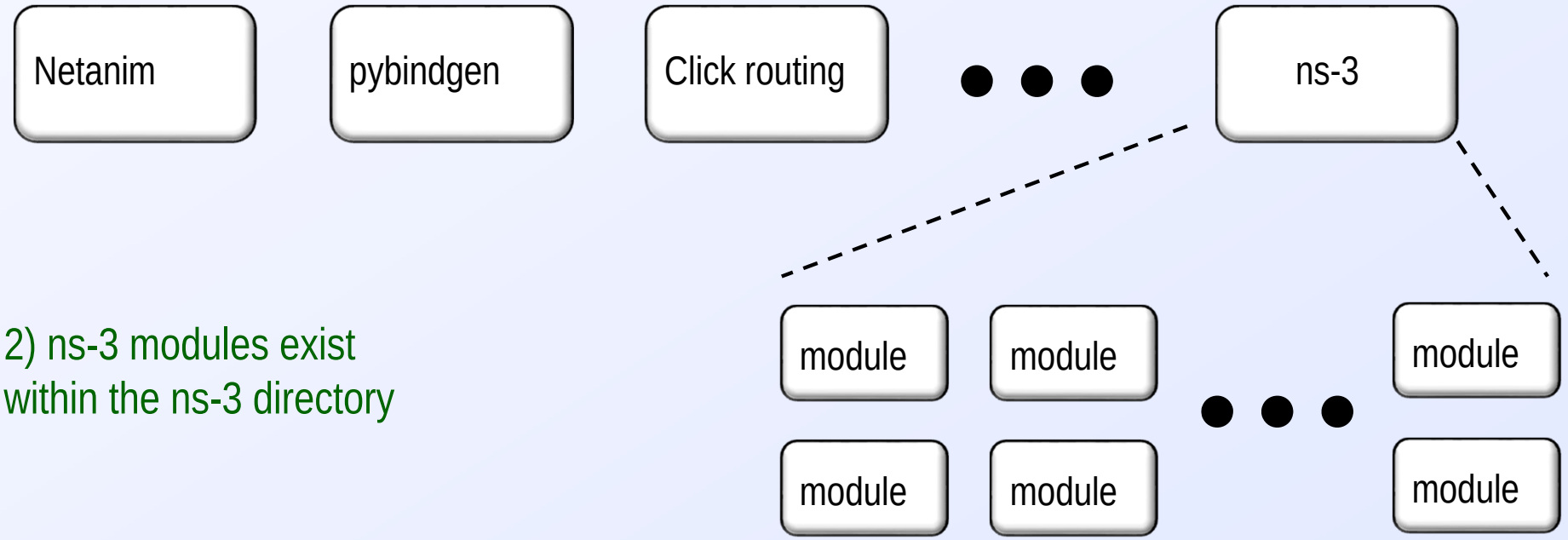
# Software overview

- ns-3 is written in C++, with bindings available for Python
  - simulation programs are C++ executables or Python programs
  - ~350,000 lines of C++ (cloc estimate)
  - almost exclusively C++98, beginning to use C++11
- 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
- Key differences from other network simulators:
  - Command-line, Unix orientation
    - vs. Integrated Development Environment (IDE)
  - Simulations and models written directly in C++ and Python
    - vs. a domain-specific simulation language
  - ns-3 does not have a graphical IDE
  - ns-3 not written in a high-level language

# Software organization

- Two levels of ns-3 software and libraries

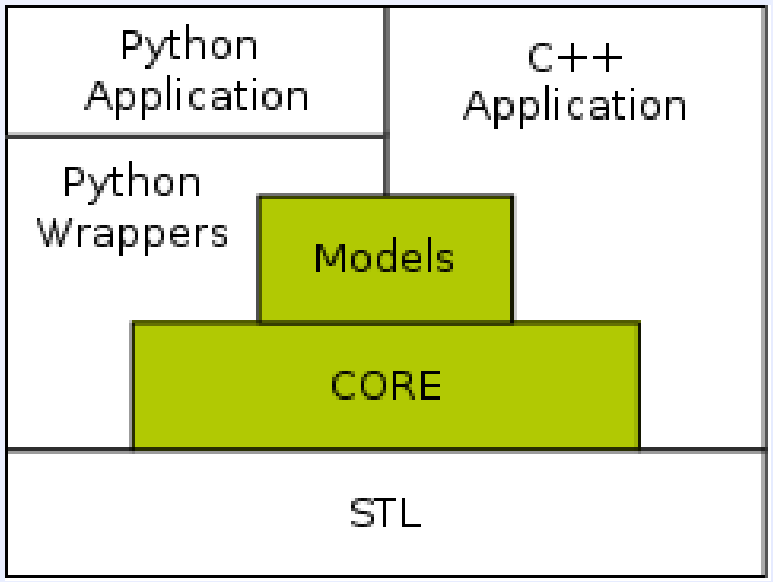
1) Several supporting libraries, not system-installed, can be in parallel to ns-3



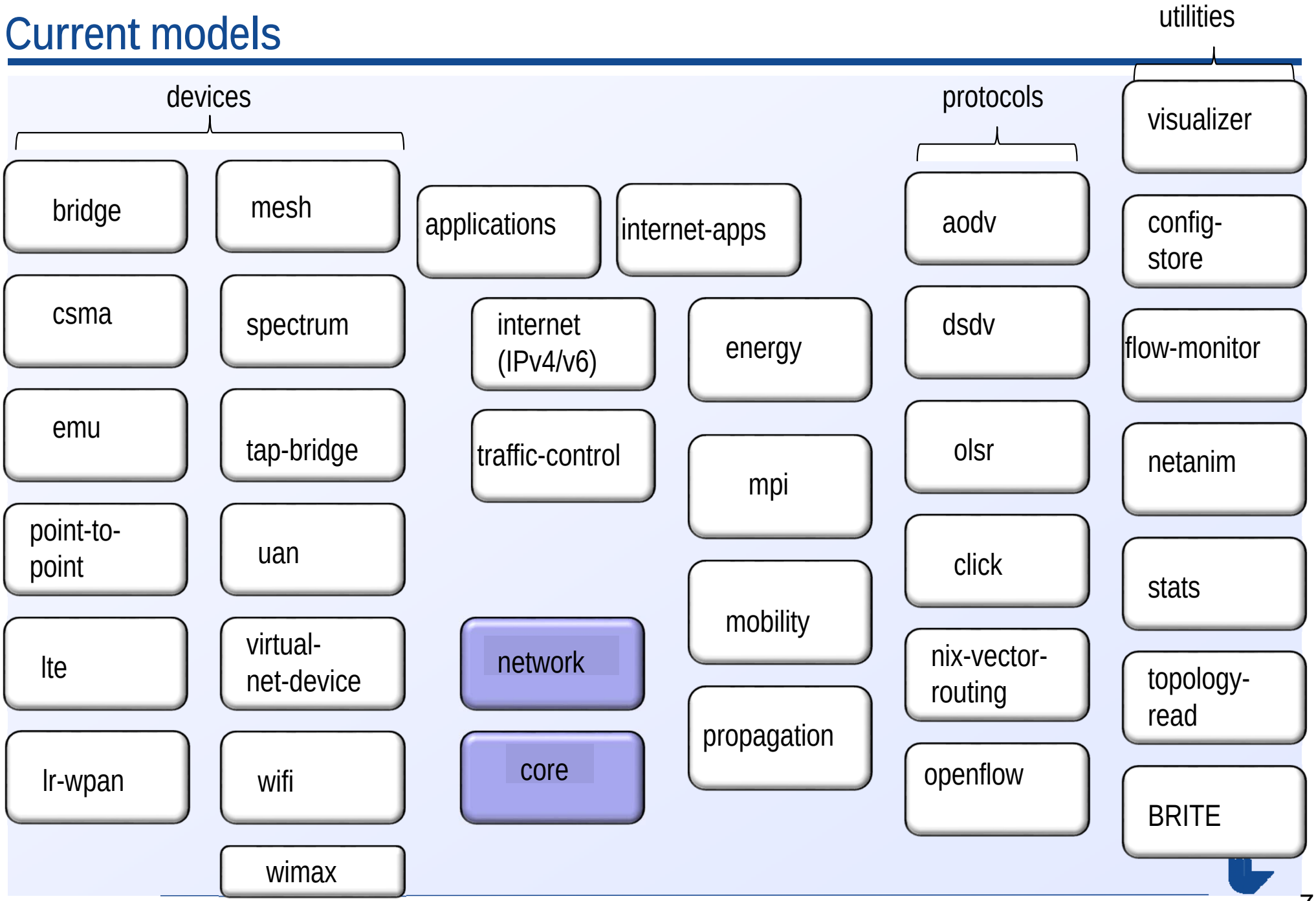

2) ns-3 modules exist within the ns-3 directory

# ns-3 programs

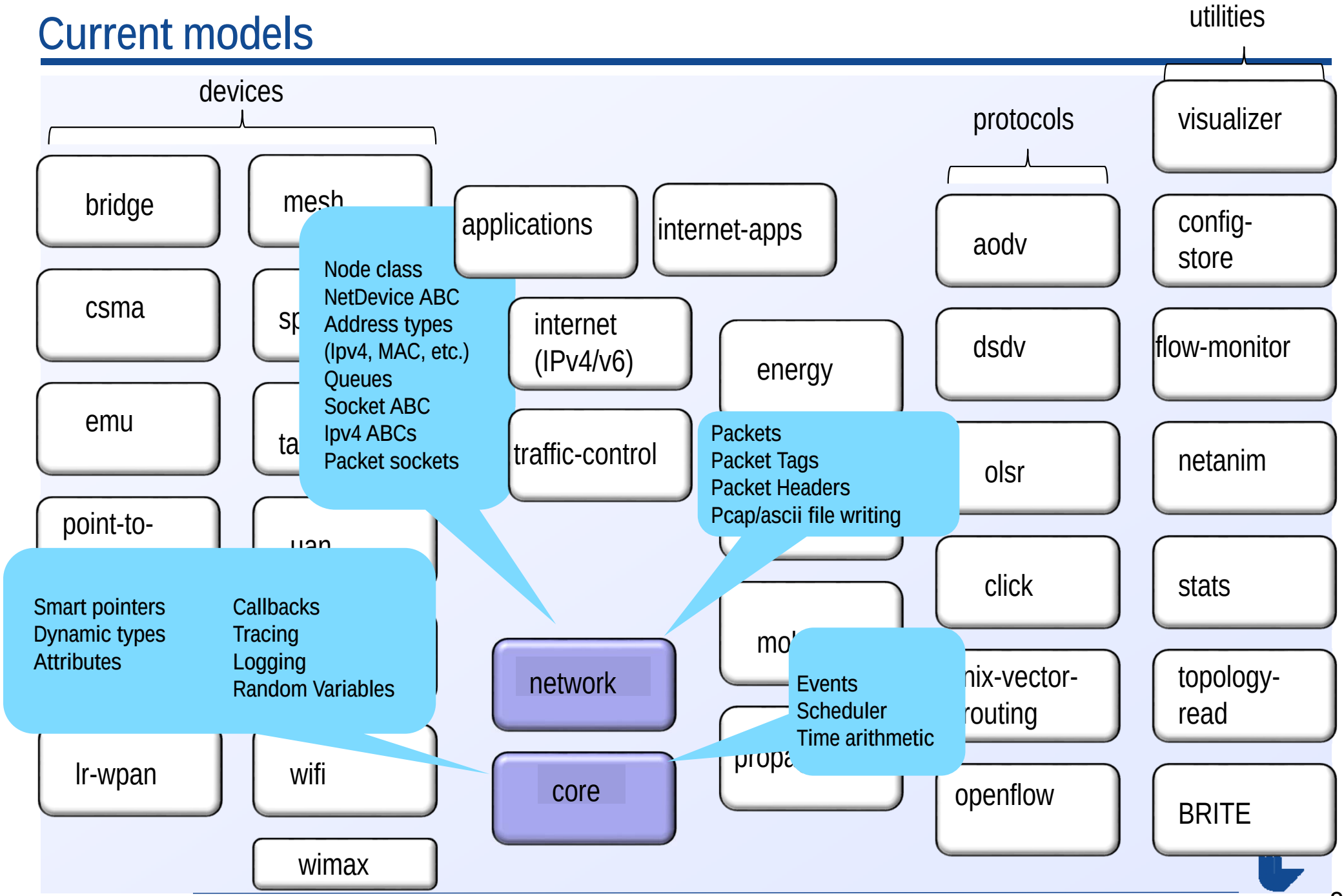
- ns-3 programs are C++ executables that link the needed shared libraries
  - or Python programs that import the needed modules
- The ns-3 build tool, called 'waf', can be used to run programs
- waf will place headers, object files, libraries, and executables in a 'build' directory
- ns-3 uses a program called PyBindGen to generate Python bindings for all libraries



# Current models



# Current models



# Module organization

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

# Chapter 2 Resources

Web site:

http://www.nsnam.org

Tutorial:

https://www.nsnam.org/docs/release/3.28/tutorial/html/index.html

Manual, Model Library, API Documentation:

**https://www.nsnam.org/ns-3-28/documentation/**

Wiki:

http://www.nsnam.org/wiki/

Mailing lists:

https://groups.google.com/forum/#!forum/ns-3-users

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

# Chapter 3 Getting started

**Source material:**

- https://www.nsnam.org/docs/release/3.28/tutorial/html/getting-started.html
- https://www.nsnam.org/wiki/Installation
- https://www.nsnam.org/tutorials/ns-3-tutorial-Walid-Younes.pdf

# Road Map

- System requirements
- Downloading ns-3
- Building ns-3
- Testing ns-3
- Running a Script

# System Requirements

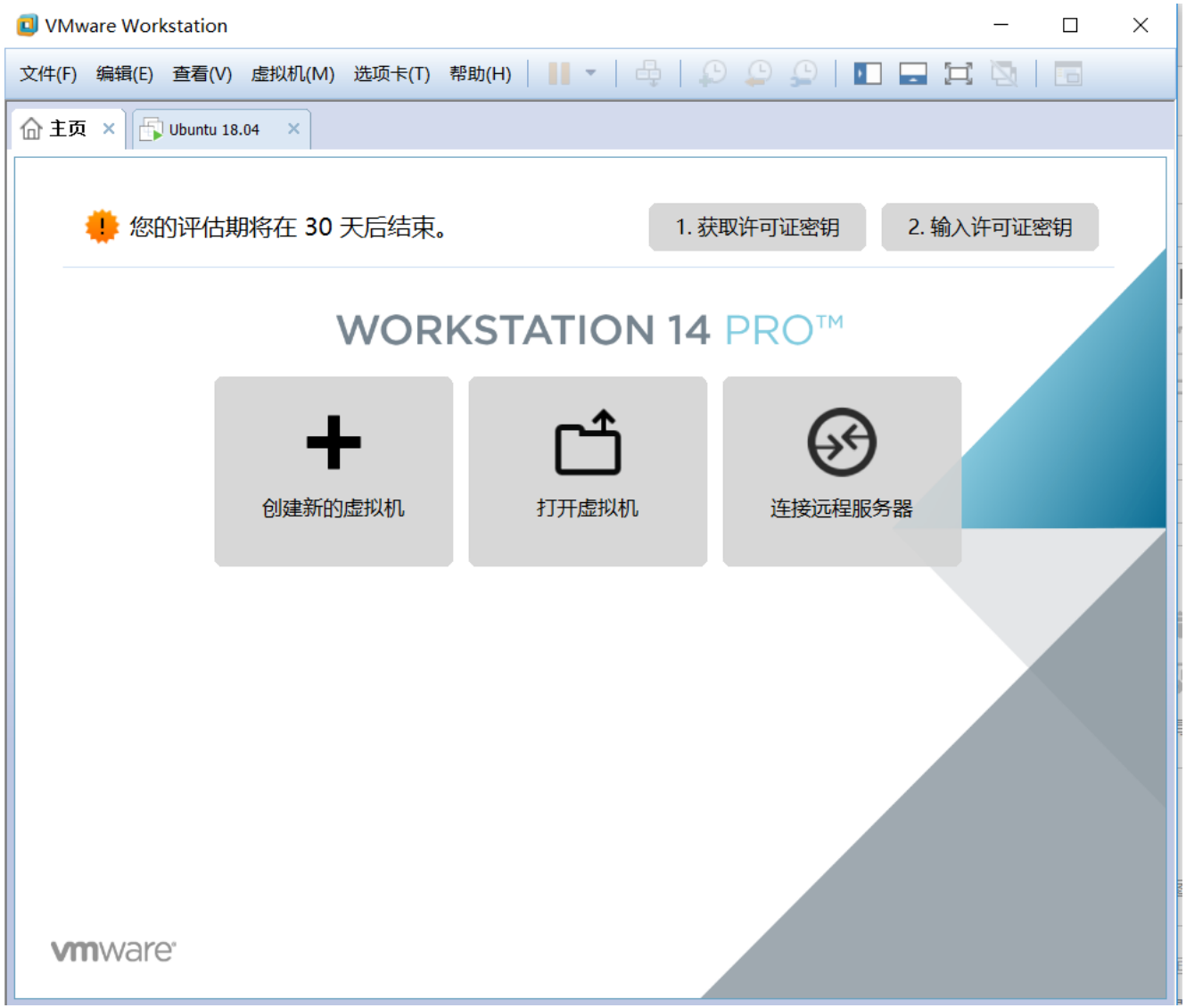
## ns-3 is supported on

- Linux platforms: gcc or **g++** **versions 4.9 and above**
- **MacOS Apple LLVM: version 8.0.0 and above (version 7.0.0 may work)**
- **FreeBSD and Linux (x86_64): clang/LLVM version 3.9 and above**

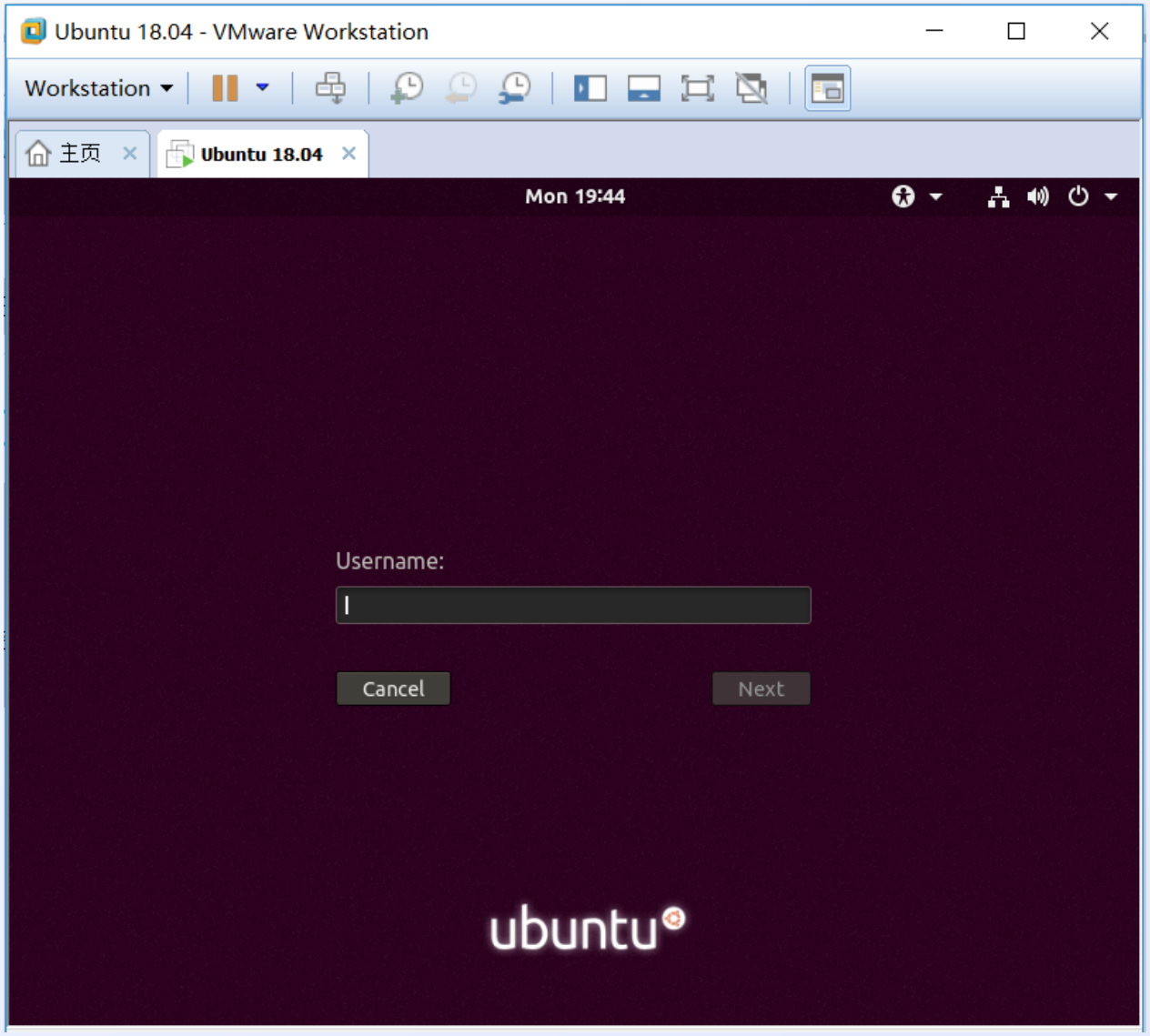
## For Windows user

1. Install VMware workstation **14 PRO**
2. Install Ubuntu 18.04 on VMware
3. Follow the instructions on https://www.nsnam.org/wiki/Installation and install the required packages for Ubuntu

# 1. Install VMware workstation 14 **PRO**

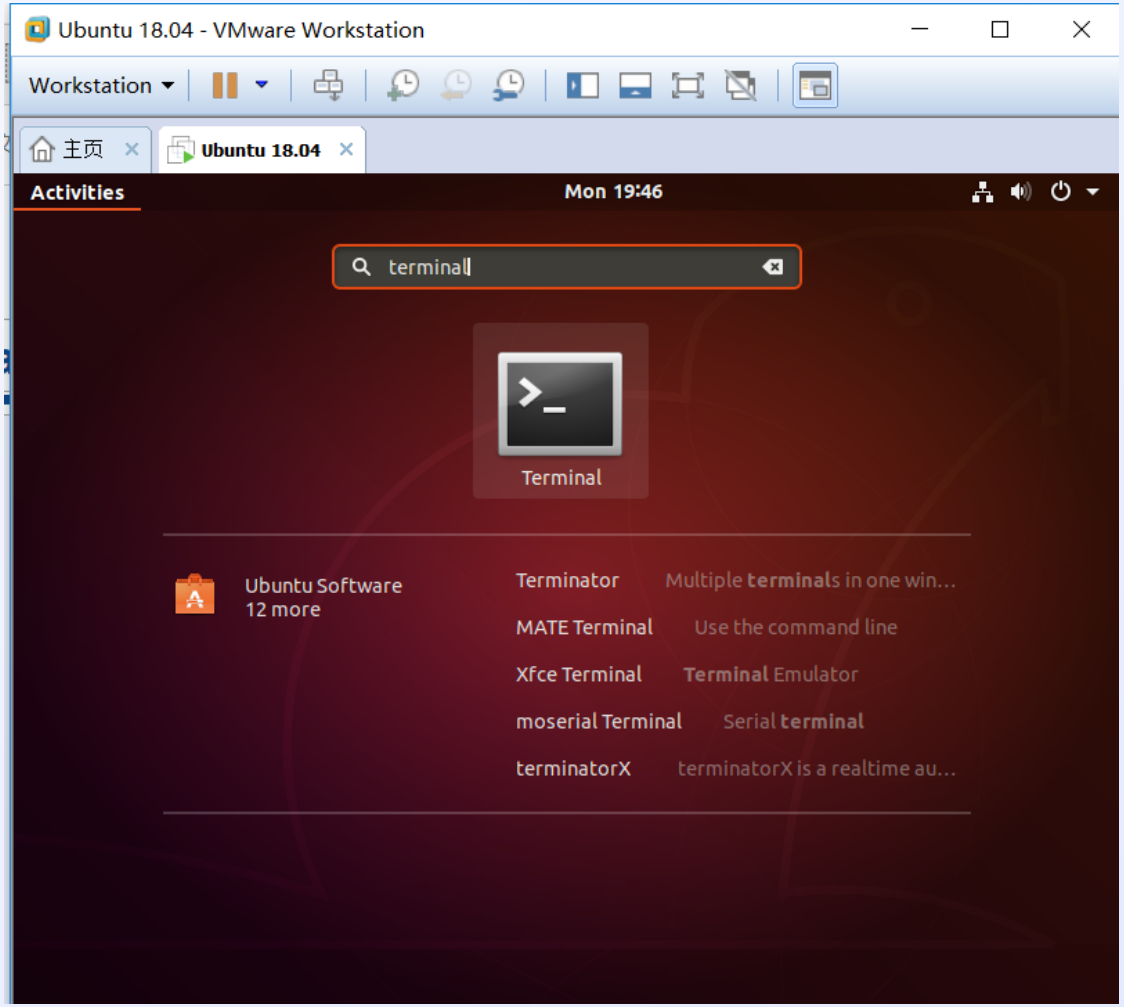


# 2. Install Ubuntu 18.04 on VMware

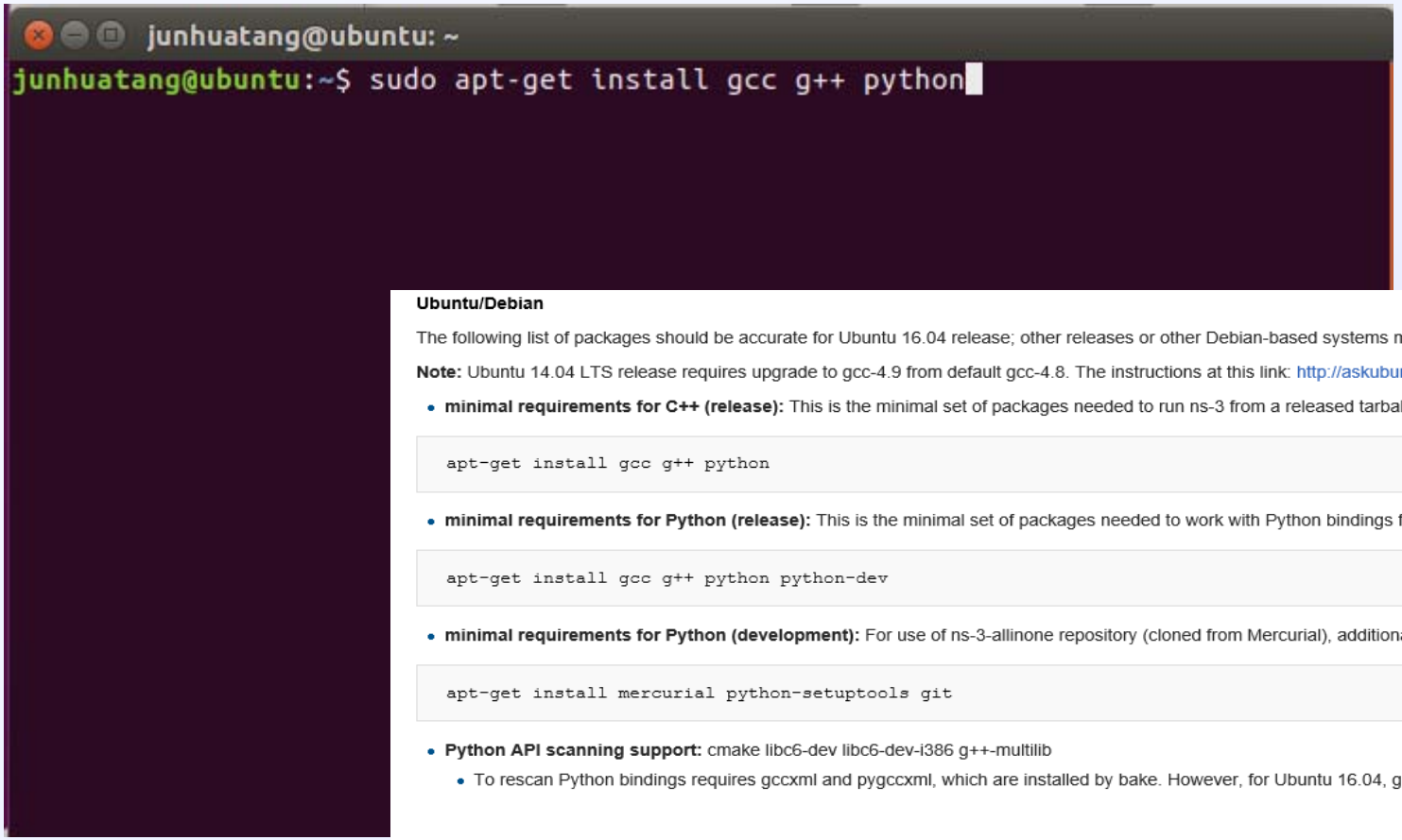


# 3. Install required packages for Ubuntu

- First open a terminal



# 3. Install required packages for Ubuntu




**Ubuntu/Debian**

The following list of packages should be accurate for Ubuntu 16.04 release; other releases or other Debian-based systems may slightly vary.

**Note:** Ubuntu 14.04 LTS release requires upgrade to gcc-4.9 from default gcc-4.8. The instructions at this link: http://askubuntu.com/questions/4666

- **minimal requirements for C++ (release):** This is the minimal set of packages needed to run ns-3 from a released tarball.

```
apt-get install gcc g++ python
```

- **minimal requirements for Python (release):** This is the minimal set of packages needed to work with Python bindings from a released tarball.

```
apt-get install gcc g++ python python-dev
```

- **minimal requirements for Python (development):** For use of ns-3-allinone repository (cloned from Mercurial), additional packages are needed

```
apt-get install mercurial python-setuptools git
```

- **Python API scanning support:** cmake libc6-dev libc6-dev-i386 g++-multilib
  - To rescan Python bindings requires gccxml and pygccxml, which are installed by bake. However, for Ubuntu 16.04, gccxml will not build with

- qt4 development tools are needed for Netanim animator (**Note:** qt version qt4, not qt5, is required)

```
apt-get install qt4-dev-tools libqt4-dev
```

Install all the required packages for Ubuntu (refer to https://www.nsnam.org/wiki/Installation )

# Downloading ns-3 Using a Tarball

$ cd

~$ mkdir workspace

~$ cd workspace

~/workspace$ wget http://www.nsnam.org/release/ns-allinone-3.28.tar.bz2

~/workspace$ tar xjf ns-allinone-3.28.tar.bz2

Detailed instructions at https://www.nsnam.org/docs/release/3.28/tutorial/html/getting-started.html#downloading-ns3-using-a-tarball

# Building ns-3

```
~/workspace$
~/workspace$ cd ns-allinone-3.28/
~/workspace/ns-allinone-3.28$ ./build.py --enable-examples --enable-tests
```

Detailed instructions at https://www.nsnam.org/docs/release/3.28/tutorial/html/getting-started.html#building-with-build-py

# Testing ns-3

- ~/workspace/ns-allinone-3.28$ cd ns-3.28
- ~/workspace/ns-allinone-3.28/ns-3.28$ ./test.py -c core

- https://www.nsnam.org/docs/release/3.28/tutorial/html/getting-started.html#testing-ns3

## Running a Script

- ~/workspace/ns-allinone-3.28/ns-3.28$ ./waf --run hello-simulator
- https://www.nsnam.org/docs/release/3.28/tutorial/html/getting-started.html#running-a-script

Congratulations! You are now an ns-3 user!