

Network Simulator-3: A Review

Ashish Mohta, Sonali I. Ajankar, M. M. Chandane

Department of Computer Technology
Veeramata Jijabai Technological Institute
Mumbai, India

urashu06@gmail.com

s.ajankar9@gmail.com

mmchandane@vjti.org.in

ABSTRACT— Due to growth of computer networks and complex scenarios the role of Network simulators in research field cannot be ignored. Simulators are useful tools when one wants to consider time and resources, implementation of new security solutions, performance estimation etc. it provides a virtual environment for an assortment of desirable features such as modeling a network based on a specific criteria and analyzing its performance under different scenarios.

The newly proposed network simulator NS-3 supports coupling, interoperability, good memory management, debugging of split language objects, coding in C++ and object oriented concepts, as well as supports models supported by NS-2 and most suitable for wireless networks. The primary purpose of this paper is to compare and review this new simulator, as well as find its advantages in the field of research and how it is different from others. We also discuss the current demand of industry as well proposed a framework for a simulator which most research people want.

Keywords- Network Simulator, Tool Command Language, Network Animator

I. INTRODUCTION

Simulation is the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviors of a selected physical or abstract system. Simulation is used in many contexts, including the modeling of natural systems or human systems in order to gain insight into their functioning, simulation of technology for performance optimization, safety engineering, testing, training and education. Simulation can be used to show the eventual real effects of alternative conditions and courses of action. Key issues in simulation include acquisition of valid source information about the relevant selection of key characteristics and behaviors, the use of simplifying approximations and assumptions within the simulation, and fidelity and validity of the simulation outcomes.

Why simulate?

To test any network we either need real system or tools or simulators, but cost of installing real network is too high which is not suitable for all cases. The following problems occur at the time of testing:

- Field tests are expensive
- Food, lodging, equipment rental, labour, etc.
- Experiments (especially wireless) can be hard to reproduce Collaboration

A typical network simulator can provide the programmer with the abstraction of multiple threads of control and inter-thread communication. Functions and protocols are described either by finite-state machine, native programming code, or a combination of the two. A simulator typically comes with a set of predefined modules and user-friendly GUI. Some network simulators even provide extensive support for visualization and animations.

At present there are many simulators in market like QualNet, GTNets, Opnet, NS-2 etc. There are some problems with one of the mostly used open source simulator ns-2; to overcome them a new simulator is proposed which is called “Network Simulator-3”. Section 2 will discuss about network simulators and their roll and will compare ns-2 and ns-3. The overview of NS-3 is given in section-3 with its features and models; the code architecture is also given in this section. Section-4 deals with building network scenario using ns-3 and reading the output. The challenges and future work with conclusion are discussed next.

II. NETWORK SIMULATOR

NS or the network simulator is a discrete event network simulator. It is popular in academia for its extensibility (due to its open source model) and plentiful online documentation. ns is popularly used in the simulation of routing and multicast protocols, among others, and is heavily used in ad-hoc networking research. ns supports an array of popular network protocols, offering simulation results for wired and wireless networks alike. It can be also used as limited-functionality network emulator.

NS began development in 1989 as a variant of the REAL network simulator. By 1995, ns had gained support from DARPA, the VINT (Virtual Inter Network Testbed) project. at LBL, Xerox PARC, UCB, and USC/ISI.

A. *Ns-2 Simulator*

NS-2 was built in C++ and provides a simulation interface through OTcl, an object-oriented dialect of Tcl. The user

describes a network topology by writing OTcl scripts, and then the main ns program simulates that topology with specified parameters [1]. Fig. 1 shows ns-2 object creation model.

The ns-2 makes use of flat earth model in which it assumes that the environment is flat without any elevations or depressions. However the real world does have geographical features like valleys and mountains.

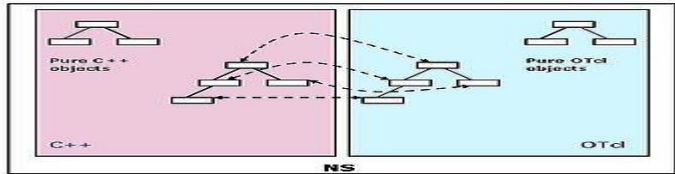


Fig. 1 Showing NS split objects model. Object created on OTcl has a corresponding object in C++

Ns-2 fails to capture this model in it. Tcl is not a good interpreter; a programmer has to suffer while writing code in tcl. There is need of good compiler for ns-2.

B. Ns-3 Simulator

Generation 3 of ns has begun development as of July 1, 2006 and is projected to take four years. It is funded by the institutes like University of Washington, Georgia Institute of Technology and the ICSI Center for Internet Research with collaborative support from the Planète research group at INRIA Sophia-Antipolis. Currently ns-3 is in development phase. It is an event based network simulator [3].

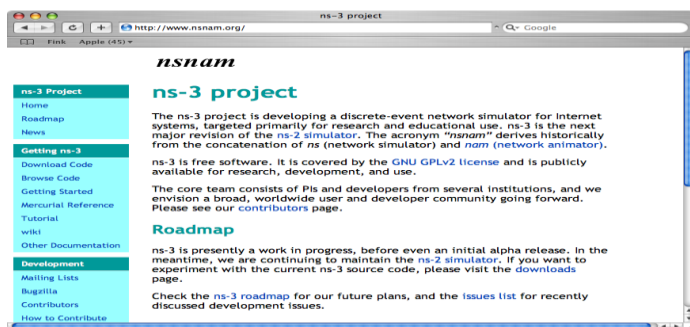


Fig. 2 NS3 Project Page

A few key points are worth noting at the onset [9]:

- Ns-3 is not an extension of ns-2; it is a new simulator. The two simulators are both written in C++ but ns-3 is a new simulator that does not support the ns-2 APIs. Some models from ns-2 have already been ported from ns-2 to ns-3.
- Ns-3 is intended to eventually replace the popular ns-2 simulator.
- Ns-3 is open-source, and the project strives to maintain an open environment for researchers to contribute and share their software.

ns-3 timeline and roadmap

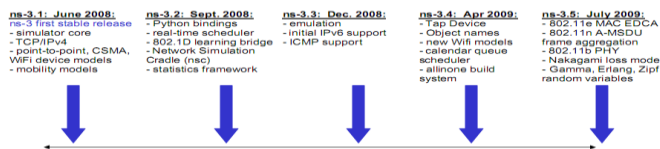


Fig. 3 an open source project building a new network simulator to replace ns2

C. How Ns-3 is different from Ns-2

The differences between these two simulators are discussed in table 1 [1][2][3][9].

TABLE 1 DIFFERENCE BETWEEN NS-2 AND NS-3

| | NS-2 | NS-3 |
|-----------------|-----------------------------------|---|
| First Release | 1996 | 2008 |
| Based on | NS-1 & REAL simulators | NS-2, GTNets, YANS |
| Architecture | OTcl & C++ | C++ & optional Python Scripting |
| Funded by | DARPA VINT SAMAN & NSF CONSER | NSF CISE & INRIA |
| Current Support | Volunteers, USC ISI & Sourceforge | NSF, INRIA, GT, WashU & Volunteers |
| Scripting | OTcl | Python |
| Visualization | NAM | NS-3-viz, pyviz, nam, iNSpect (all under development) |
| Scalability | Sequential Simulation | Distributed Simulation |

III. NS-3 OVERVIEW

The brief overview of simulator is discussed in this section with its features [4]-[8]:

A. Ns-3 Features

The ns-2 simulator has long been a widely used simulator for research and education on Internet and other network systems. However, work is progressing on a replacement for ns-2. Borrowing concepts and implementations from several open source simulators including ns-2, YANS and GTNetS, ns-3 differs from ns-2 in several ways, including:

- 1) *New software core:* Designed to improve scalability, modularity, coding style, and documentation, the core is written in C++ but with an optional Python scripting interface (instead of OTcl). Several C++ design patterns such as smart pointers, templates, callbacks, and copy-on-write are leveraged. Object

aggregation capabilities enable easier model and packet extensions.

2) *Attention to realism:* The Internet nodes are designed to be a more faithful representation of real computers, including the support for key interfaces such as sockets and network devices, multiple interfaces per nodes, use of IP addresses, and other similarities.

3) *Software integration:* Architecture to support the incorporation of more open-source networking software such as kernel protocol stacks, routing daemons, and packet trace analyzers, reducing the need to port or rewrite models and tools for simulation.

4) *Support for virtualization:* Lightweight virtual machines running over a (possibly wireless) simulation network are an attractive combination for current research; ns-3 plans to support a few modes of such operation including a native “process” environment where Posix-compliant applications can be easily ported to run in simulation space with their own private stack, and including support for tying together virtual machines of various types.

5) *Testbed integration:* Ns-3 will enable the testbed-based researcher to experiment with novel protocol stacks and emit/consume network packets over real device drivers or VLANs. The internal representation of packets is network-byte order to facilitate serialization.

6) *Attribute system:* Researchers require a means to identify and possibly reassign all values used to configure parameters in the simulator. Ns-3 provides an attribute system that integrates the handling and documentation of default and configured values.

7) *Tracing architecture:* Ns-3 is building a tracing and statistics gathering framework using a callback-based design that decouples trace sources from trace sinks, enabling customization of the tracing or statistics output without rebuilding the simulation core.

8) *Topology:* For ease of use, a number of stock topology objects should be predefined. These stock objects can be instantiated by a single line of C++ code constructing the object, with configurable arguments. Stock objects should include trees, meshes, stars, and random topologies of arbitrary size. They have incorporated such topology objects from GTNetS.

Like ns-2, ns-3 is open-source and licensed under GNU GPLv2, and welcomes developers and contributed code from across academia, industry, and government.

B. Ns-3 Models

The simulator needs updating to account for the rapid growth in wireless networking, including the many variants of IEEE 802.11 networking, emerging IEEE standards such as WiMax (802.16), and cellular data services (GPRS, CDMA). Additional models beyond wireless are also needed; Table 2 summarizes the

models used in the current ns-2, as well as models planned for ns-3. Many of the planned models may already exist in some form as contributed code; for a new model to be incorporated into the main branch of ns-3, it will need to be validated, conform as appropriate to the coding style, be licensed in a compatible way, and be maintained going forward[8][10].

TABLE 2 NS-3 MODEL COMPARISON WITH EXISTING NS-2

| Layers | Existing CoreNS-2 Capability | Planned additions for NS-3 |
|---------------------------|--|---|
| Application and Transport | Ping, vat, telnet, FTP, multicast, FTP, probabilistic and trace-driven traffic gen., web cache | Sockets-like API, P2P, traffic generator |
| | TCP, UDP, SCTP, XCP, TFRC, RAP, RTP, Multicast: PGM, SRM, RLM, PLM | TCP stack emulation (Linux, BSD), DDCP, additional high speed TCP variants, UDP |
| Network | Unicast: IP, MIP, DV, LS, IPinIP, SR, Multicast: SRM, | full IPv4 & IPv6 support, NAT, BGP, OSPF, RIP, IS-IS, PIM-SM, IGMP/MLD |
| | MANET: AODV, DSR, DSDV, TORA, IMEP | MANET: OLSR |
| Link & MAC | ARP, HDLC, GAF, MPLS, LDP MAC: CSMA, 802.11b, 802.15.4, satellite Aloha | new 802.11 model, 802.11 variants (mesh, QoS), 802.16, TDMA, CDMA, GPRS, CSMA |
| Physical and Mobility | Two-way, shadowing, Omni Antennas, Energy model Satellite repeater | IEEE 802 physical layers, Rayleigh and Rician fading channels, GSM, Jakes composite loss model, Friis, log-distance |

C. Ns-3 Code Architecture

NS-3 code is divided into different parts. Here we start with topology definition and then we define models to use, after that we configure over model by giving them some addresses and setting other parameters, and finally we executed the code. The output which is generated in trace format will be analysed with some tools like Wireshark. The procedure of code creation is shown in the fig. 4:

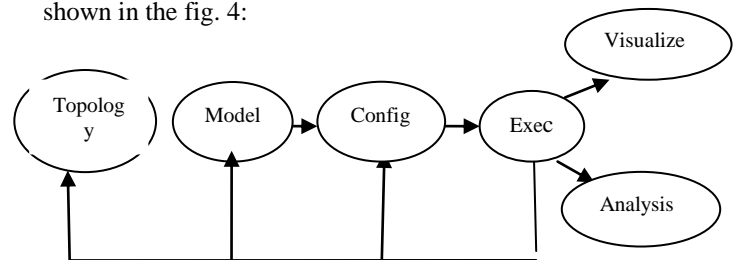


Fig. 4 ns-3 Source Code

The detailed source code showing use of different classes in ns-3 shown in fig 5.

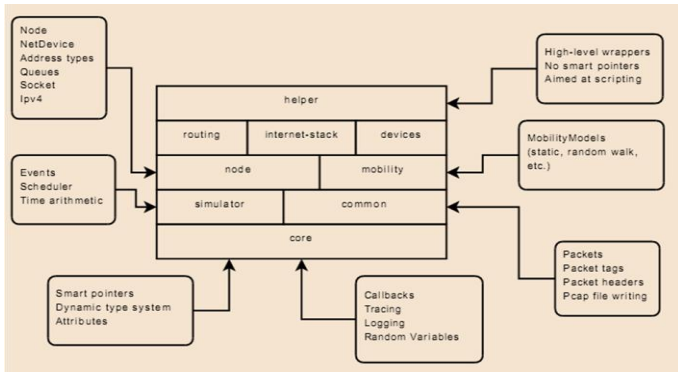


Fig 5 ns-3 Detailed Source Code

IV. BUILDING A SCENARIO

For showing how to create any network scenario using NS-3¹ we have created a wireless scenario with fixed sources and sinks and nodes placed at specific position having OLSR routing installed on them. The output is generated in trace form which can be readable by Wireshark [7]-[11]. Because of lack of space we have not giving full code here but specific part of code with detailed explanation

A. Example

```
//First line of code for every scenarios
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
//include required library files
#include "ns3/core-module.h"
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
.....
//main () function
Int main (int argc, char *argv[])
{ //variable declaration
  bool verbose = true;
  .....
//parameter definition
  Config::SetDefault ("ns3::OnOffApplication::PacketSize",
    StringValue("500"));
  .....
// Creating ns-3 node obj, create, manage and access any node
  NodeContainer wifiStaNodes;// declare node
  wifiStaNodes.Create (nWifi);// creating nWifi nodes
// Construct wifi devices and the interconn channel b/w these
//nodes
  YansWifiChannelHelper channel =
  YansWifiChannelHelper::Default ();
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
  phy.SetChannel (channel.Create ());
//focusing on MAC layer, setting type of rate ctrl algo to use
```

```
WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager
("ns3::ConstantRateWifiManager", "DataMode",
StringValue("wifia-6mbs"));
NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
mac.SetType ("ns3::AdhocWifiMac", "Slot", StringValue
("16us"));
```

```
.....
//configreing the devices and channels
  NetDeviceContainer staDevices; //device container
  staDevices = wifi.Install (phy, mac, wifiStaNodes
  .....
```

```
//Position Allocation
  MobilityHelper mobility;
  Ptr<ListPositionAllocator> positionAlloc =
  CreateObject<ListPositionAllocator> ();
  for (uint32_t x = 0; x < nWifi; x++) {
    positionAlloc->Add (Vector ((x*latDistance), 0.0, 0.0));
  }
  mobility.SetPositionAllocator (positionAlloc);
  .....
```

```
// Nodes are stationary, constPos or RandomWalk2d
  mobility.SetMobilityModel
  ("ns3::ConstantPositionMobilityModel");
  mobility.Install (wifiStaNodes);
  .....
```

```
//Enable Routing on nodes
  OlsrHelper olsr;
  olsr.Create (wifiStaNodes.Get(1));
  olsr.Create (wifiStaNodes.Get(2));
  .....
```

```
// Protocol stacks installation on nodes
  InternetStackHelper stack; //topo helper
  stack.Install (wifiStaNodes);//NodeContainer as a param
//Ip add association on nodes base address and network mask
  Ipv4AddressHelper address;
  address.SetBase ("10.1.3.0", "255.255.255.0");
  address.Assign (staDevices);
  .....
```

```
uint16_t port = 9; // Discard port is 9 (RFC 863)
  Ipv4Address remoteAddr = "255.255.255.255";
  OnOffHelper onoff ("ns3::UdpSocketFactory", Address
  (InetSocketAddress (remoteAddr, port)));
  onoff.SetAttribute ("OnTime", RandomVariableValue
  (ConstantVariable (10)));
  .....
```

```
// set Nodes to broadcast
  ApplicationContainer apps;
  Ptr<Node> appSource = NodeList::GetNode (0);
  apps = onoff.Install (appSource);
```

¹Getting started with ns-3, discussed in Appendix

```
.....
// Create a packet sink to receive these packets
// Output does not change if the sink is not installed
```

```

PacketSinkHelper sink ("ns3::UdpSocketFactory",
InetSocketAddress (Ipv4Address::GetAny (),port));
for (uint32_t nNodes = 1; nNodes < (nWifi-1); nNodes++)
{
std::cout << "Node " << nNodes << " is a sink." <<
std::endl;
Ptr<Node> appSink = NodeList::GetNode (nNodes);
apps = sink.Install (appSink);
apps.Start (Seconds (0.0));
}
}
// Tracing configuration
NS_LOG_INFO ("Configure Tracing.");
phy.EnablePcapAll ("Output_Adhoc");
Simulator::Stop (Seconds (10.0));
Simulator::Run ();
Simulator::Destroy ();
return 0;
}

```

B. Running the Program

For running the current program we can use *waf* script [12] as shown in fig. 6



```

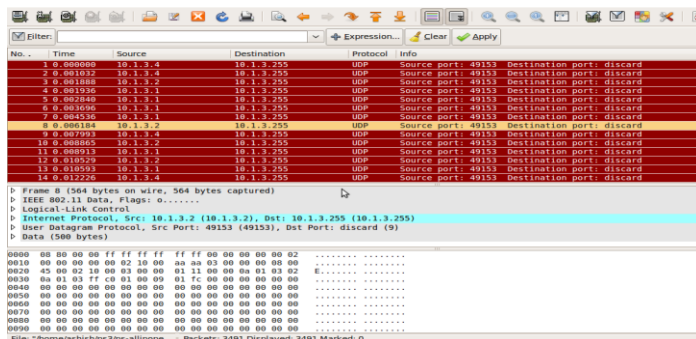
root@ashish-laptop: /home/ashish/ns3/ns-allinnone-3.5/ns-3.5
File Edit View Terminal Help
root@ashish-laptop: /home/ashish/ns3/ns-allinnone-3.5/ns-3.5# ./waf --run scratch/
nsn
Waf: Entering directory `~/home/ashish/ns3/ns-allinnone-3.5/ns-3.5/build'
Waf: Leaving directory `~/home/ashish/ns3/ns-allinnone-3.5/ns-3.5/build'
'build' finished successfully (0.469s)
Node 0 0 0
Node 1 50 0
Node 2 100 0
Node 3 150 0
Node 1 is a sink.
Node 2 is a sink.
root@ashish-laptop: /home/ashish/ns3/ns-allinnone-3.5/ns-3.5#

```

Fig. 6 Running Script

C. Reading Output File

For reading the output file generated by simulator we can use tcpdump as well as Wireshark, we have used Wireshark to read the output file as shown in fig. 7 [14].



| No. | Time | Source | Destination | Protocol | Info |
|-----|----------|----------|-------------|----------|--|
| 1 | 0.000000 | 10.1.3.4 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 2 | 0.001032 | 10.1.3.4 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 3 | 0.001880 | 10.1.3.2 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 4 | 0.001936 | 10.1.3.1 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 5 | 0.002640 | 10.1.3.1 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 6 | 0.003004 | 10.1.3.1 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 7 | 0.004536 | 10.1.3.1 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 8 | 0.006144 | 10.1.3.2 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 9 | 0.007793 | 10.1.3.4 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 10 | 0.008895 | 10.1.3.2 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 11 | 0.008933 | 10.1.3.1 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 12 | 0.010329 | 10.1.3.2 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 13 | 0.010593 | 10.1.3.1 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |
| 14 | 0.012225 | 10.1.3.2 | 10.1.3.255 | UDP | Source port: 49153 Destination port: discard |

Fig. 7 Reading Trace File

D. The Fundamental Objects

The Fundamental objects are node, application, sockets, channels and net devices which we take into consideration while writing any code in ns-3 [8]-[10].

- 1) *Node*: The motherboard of a computer with RAM, CPU, and, IO interfaces
- 2) *Application*: A packet generator and consumer which can run on a Node and talk to a set of network stacks.
- 3) *Socket*: The interface between an application and a network stack.
- 4) *NetDevice*: A network card which can be plugged in an IO interface of a Node
- 5) *Channel*: A physical connector between a set of NetDevice objects

An ns-3 Node is a husk of a computer to which applications, stacks, and NICs are added

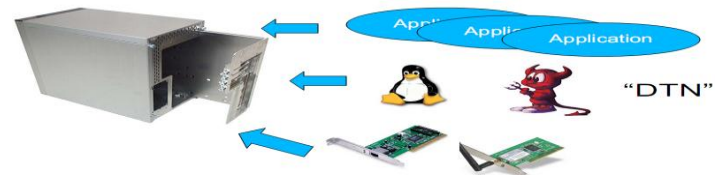


Fig. 8 Node

- 6) *Typical containers and helpers*: There are different container and helper classes in ns-3. NodeContainer, NetDeviceContainer, Ipv4AddressContainer are some of the container classes and InternetStackHelper, WifiHelper, MobilityHelper, OlsrHelper are some of the helper classes in ns3.

V. CHALLENGES

Like all new technologies and tools NS-3 is also facing challenges from other simulators which are in market. Because NS-3 is a new simulator so for overcoming these challenges and fulfilling current needs of simulation it needs:

- participation from the research community
- Improving simulation credibility
- Contributed and supported models
- Maintainers

VI. FUTURE WORK

There are so many challenges faced by simulator field. Not a single simulator satisfies current user's need. There are so many researches, comparisons and surveys are needed before designing any simulator. Here we have compared two simulators which are open source, where ns-3 is in development phase and needed more support from its users and researchers. Ns-3 overcomes certain problems but there is need of some improvement like, network animator tool for wireless scenarios, user friendliness and ease of use as well as good tutorial and wider community support, so that a naive user can easily get comfortable with it. And most important that before release the stable version it should be well tested, so that it is free of any bugs and errors.

VII. CONCLUSION

There are many simulators (Google for network simulator) like Opnet, QualNet, Shunra but because of terms of use and high cost for industrial partners or publicly-funded research these cannot get education licenses. Despite ns-2's popularity, there is a critical need for a new project to perform core refactoring, integration, software maintenance, and extension of the simulator.

Despite all these NS-3 is an active open-source project and open-source development model, several simulator features designed to aid current Internet research, community-based development and maintenance model, trying to avoid some problems with ns-2, such as interoperability and coupling between models, lack of memory management, debugging of split language objects.

A question that we often hear is "Should I still use ns-2 or move to ns-3?" The answer is that it depends [9]. NS-3 does not have all of the models that NS-2 currently has, on the other hand, NS-3 does have new capabilities (such as handling multiple interfaces on nodes correctly, use of IP addressing and more alignment with Internet protocols and designs, more detailed 802.11 models, etc.). Ns-2 models can usually be ported to Ns-3.

REFERENCES

- [1.] NS-2 user information <http://www.nsnam.isi.edu/nsnam/index.php/UserInformation>
- [2.] Mathieu Lacage, Thomas R. Henderson "Yet Another Network Simulator" INRIA, Planete Project
- [3.] Basic information about NS-3: <http://www.nsnam.org>
- [4.] Getting started with NS-3 http://www.nsnam.org/getting_started.html
- [5.] Detailed documentation of NS-3: <http://nsnam.org/documents.html>
- [6.] Source code of NS-3: <http://code.nsnam.org>
- [7.] NS-3 Community Support: <http://mailman.isi.edu/mailman/listinfo/ns-developers>
- [8.] Mathieu Lacage, "An NS-3 Tutorial" INRIA Tunis, April, 7th and 8th 2009
- [9.] NS-3 Tutorial: <http://www.nsnam.org/docs/tutorial/tutorial.html>
- [10.] Tom Henderson, Mathieu Lacage "ns-3 tutorial", Workshop on ns-3 March 2009
- [11.] NS-3 Wiki: http://www.nsnam.org/wiki/index.php/Main_Page
- [12.] Details of Waf <http://code.google.com/p/waf/>
- [13.] Mercurial main site <http://www.selenic.com/mercurial/>
- [14.] Ulf Lamping, Richard Sharpe "Wireshark User's Guide: 29980 for Wireshark 1.2.0", NS Computer Software and Services P/L, Ed Warnicke,

APPENDIX

1) *Environment Setup: Linux (Working from development version)*

- a) `sudo apt-get install build-essential g++ python mercurial`
- b) `hg clone http://code.nsnam.org/ns-3-dev`
- c) `cd ns-3 dev`
- d) `./download.py` #will download components

- e) `./build.py` #will build NS-3
- 2) *Building From Within Ns-3-Dev*
- a) `cd ns-3-dev`
 - b) `./waf distclean` (similar to `make distclean`)
 - c) `./waf configure`
 - d) Or `./waf -d optimized configure`
 - e) `./waf`

Availability (Linux, osx, cygwin, mingw):

- Released tarballs: <http://www.nsnam.org/releases>
- Development version: <http://code.nsnam.org/ns-3-dev>

The development version is usually stable: a lot of people use it for daily work.

3) *TESTING NS-3:* You can run the unit tests of the ns-3 distribution by running the "--check" option,

`./waf --check`

These tests are run in parallel by waf, so the summary, "Ran n tests" will appear as soon as all of the tasks are launched, but you should eventually see a report saying that,

C++ UNIT TESTS: all 33 tests passed.

RUNNING A SCRIPT

We typically run scripts under the control of Waf. This allows the build system to ensure that the shared library paths are set correctly and that the libraries are available at run time. To run a program, simply use the --run option in Waf. Let's run the ns-3 equivalent of the ubiquitous hello world program by typing the following:

`./waf --run hello-simulator`

Waf first checks to make sure that the program is built correctly and executes a build if required. Waf then executes the program, which produces the following output.

Hello Simulator

Congratulations. You are now an ns-3 user.