ns-3 Project Goals

Thomas R. Henderson and Sumit Roy
Department of Electrical Engineering
University of Washington
Seattle, Washington 98195–2500
Email: thenders, roy@ee.washington.edu

Sally Floyd
ICSI Center for Internet Research
1947 Center Street, Suite 600
Berkeley, CA 94704
Email: floyd@icir.org

George F. Riley
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: riley@ece.gatech.edu

***Abstract*— This paper reports on the project plan to develop a new major version of the popular *ns-2* networking simulator. The authors have organized an NSF-funded, four-year community infrastructure project to develop the next version of *ns*. The project will also be oriented towards community development and open source software practices to encourage participation from the broader research and educational community. The purpose of this paper is to expand on the goals and initial design concepts for this new software development effort.**

I. Introduction

Discrete-event network simulation is a powerful research tool for investigating protocol design, protocol interactions, and large-scale performance issues. While simulation is not the only tool used for data networking research, it is extremely useful because it often allows research questions and prototypes to be explored at many orders-of-magnitude less cost and time than that required to experiment with real implementations and networks. Simulation is also quite effective in education, for the same reasons as above but also because key concepts can be studied and highlighted more clearly in isolation from other system elements.

One simulation tool in particular, the *Network Simulator (ns) version 2*, has seen heavy use in data networking research over the past decade. *ns-2* is the second major iteration of a discrete-event network simulation platform programmed in *C++*. *ns-2* was first released in 1996, and derives from earlier work on S. Keshav's REAL simulator [1] and the original *ns* (*ns-1*) simulator released by Lawrence Berkeley National Laboratory in 1995. *ns-2* is a major architectural change from *ns-1*– the simulator became entirely based on the blend of MIT Object Tcl (OTcl) and *C++*. The development of *ns-2* has been funded by the DARPA VINT (Virtual InterNetwork Testbed) project from 1997-2000, and by DARPA SAMAN (Simulation Augmented by Measurement and Analysis for Networks) and NSF CONSER (Collaborative Simulation for Education and Research) from 2000-2004. Presently, development is not funded but is performed by volunteers, and the project is hosted at USC ISI and Sourceforge.

The core of *ns-2* is written in *C++*, but the *C++* simulation objects are also linked to shadow objects in OTcl. Simulation scripts are written in the OTcl language (an extension of the Tcl scripting language). This structure permits simulations to be written and modified in an interpreted environment without having to resort to recompiling the simulator each time a structural change is made. In the timeframe that *ns-2* was introduced (mid-1990s), this provided both a significant convenience in avoiding many time-consuming recompilations, and also allowing potentially easier scripting syntax for describing simulations. *ns-2* has a companion animation object known as the Network Animator (*nam*), used for visualization of the simulation output and for (limited) graphical configuration of simulation scenarios. Presently, *ns-2* consists of over 300,000 lines of source code, with probably a comparable amount of contributed code that is not integrated directly into the main distribution.

ns-2 is a research community resource, and counts several thousand downloads per month from its source code mirrors. While *ns-2* has been funded by a number of previous research projects, no funding has been in place since 2004, and none directly as an infrastructure project since 2000. The authors of this paper and their respective institutions (University of Washington, Georgia Institute of Technology, and the ICSI Center for Internet Research), with collaborative support from the Planete research group at INRIA Sophia-Antipolis, have developed a four-year, NSF-funded program about to commmence. While a portion of the new project will be devoted to maintaining the existing *ns-2* codebase, the bulk of the funding will go towards the design and development of a new major version of the simulator, called *ns* version 3 (*ns-3*). This paper describes the initial *ns-3* project plan; this plan is subject to future revision based on community input once the project is announced.

II. Functional Goals of ns-3

This section lists some of the goals of *ns-3* from a functional standpoint. While the project plans to reuse many of the models from *ns-2*, strict backward compatibility (being able to run unmodified *ns-2* scripts in *ns-3*) is not a project goal.

A. Overview

The main goal of the *ns-3* project is to produce a discrete-event network simulator for Internet systems, with an emphasis on layers 2-4 of the network stack, targeted primarily for research and educational use.

The following goals are also important:

- The project should adopt community-oriented open source development practices.

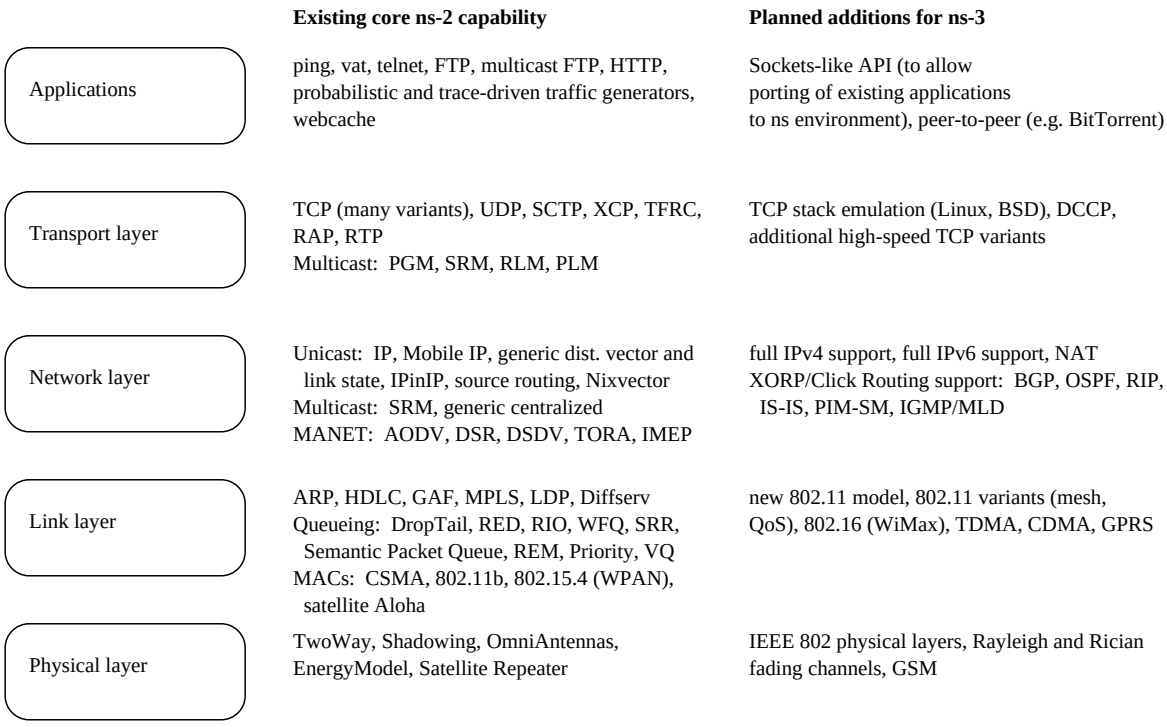


Fig. 1. Models planned for *ns-3* project.

- The simulator should be distributed as free and open source software, and should leverage and permit inclusion of other free and open source networking software.
- The simulator should be architected for scalability, extensibility, modularity, emulation, and clarity (of design), and should be well documented.
- Core models should be well tested and validated.
- The project should develop a set of canonical simulation-based experiments for use in networking courseware.

B. Architectural goals

While striving to maintain as much model reuse as possible (including a backward compatibility capability), we plan to rearchitect the simulator for better ease of use, scalability (principally by class redesign, natively supporting multiprocessor and distributed simulations, and support for 64-bit machines), and support for integration of other software. The simulator should easily, with realistic models at different levels of abstraction, allow for simulations of IPv4 and IPv6 networks, as well as novel, research-oriented network architectures.

We describe in more detail the current plan for refactoring the core of the simulator below in Section III.

C. User experience

ns-3 should be installable from source or package formats on popular desktop and server platforms, such as `i386`, `x86_64`, and `ppc`, and the Linux, OS X (Darwin), Windows (Cygwin emulation), and FreeBSD operating systems.

ns-3 should continue to offer a text/script-based (non-GUI) configuration. It should be possible to create GUI-based configurators, but such configurators are outside the scope of the project. The simulator should output trace (including `pcap`), log, statistics, and animation files; trace and log files should be convertible to the existing *nam* format, via some external scripting technique, for backward compatibility.

A key question is what type of scripting interface to provide. There seems to be unanimous consent that OTcl should be replaced, but whether a scripting front-end is used or whether scenarios are to be written directly in the compiled (*C++*) language is an open issue. An area of interest is providing hooks for the Simplified Wrapper and Interface Generator (`swig`), so that users may use whatever scripting language they prefer.

D. Integration

We see a tremendous opportunity to leverage the networking software developed under other open source software projects, and believe that this can be a key goal for a new simulator. We have three specific goals in mind:

1) Extension of the simulation capability via integration with open source tools;
2) Abstraction layers and interfaces for porting implementation code into the *ns* environment; and
3) Interfaces to allow users to easily migrate between simulation and network emulation environments.

In Section IV, we describe in more detail our plans for achieving better integration.

E. 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; Figure 1 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.

III. Core refactoring

In this section, we outline a set of design objectives for the core of the *ns-3* simulator. We see these as basic requirements for the design, implementation, and overall success of this tool. First, the *ns-3* design and implementation should leverage large parts of the code base of existing tools, such as *ns-2*, the Georgia Tech Network Simulator (*GTNetS*) [12], and others. These existing tools have hundreds of thousands of lines of code, and hundreds of existing network models. While we fully expect some modifications to these code bases will be necessary to fit within our overall design, we will strive to avoid complete re–design or re–implementation of existing modules as much as possible. We expect to be able to reuse large sections of the existing *ns-2* and *GTNetS* code. A key consideration in the design is in preserving as much backward compatibility as possible, most likely through providing a backward compatibility mode for existing scripts, and syntactic migration of legacy scripts to the new core, with re-verification of the output.

A. Scalability

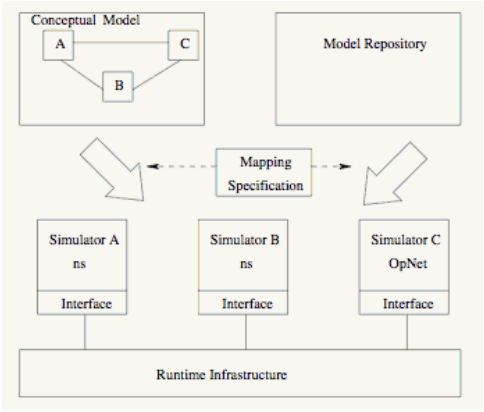
One of the major concerns about *ns-2* cited by its users is scalability. *ns-2* is a sequential execution simulator with a single event processing loop running on a single processor. Although such a simulator can scale to hundreds of nodes when underlying communications models are heavily abstracted, the memory and processing resources of a single machine become a bottleneck when more sophisticated channel models (e.g., wireless) or higher-rate links (e.g., 10 Gbps) are included. Researchers have taken various approaches to improve *ns-2* scalability, including the caching of redundant computations and function calls (the "Staged NS (SNS)" project at Cornell), use of on-demand route computation [2], and partitioning the simulation into wireless clusters (the *ns-2 gridkeeper* and similar structures developed by Naoumov and Gross in [3]).

There are a number of factors contributing to the scalability limitation, including overall software architecture, but the fundamental bottleneck is the execution on a single processor. We believe that the *ns-3* simulator should be designed from the outset to support parallel and distributed simulation. The COMPASS, PADS, and MASCOTS research groups at Georgia Tech have developed a *Federated Simulation Developers Kit* (*FDK*) and a *ghost–node* approach, used in *GTNetS* [6], which we plan to investigate for *ns-3*. This approach creates a federated network simulation consisting of a number of instances of the simulator tied together using a Runtime Infrastructure (RTI) middleware software layer (Figure 2(a)), with the RTI-interconnected nodes communicating over Myrinet or Ethernet. With this approach, a small memory overhead is incurred at each of the distributed simulation processes, which obviates the need for complicated inter–simulator routing decisions and a simulated routing protocol.

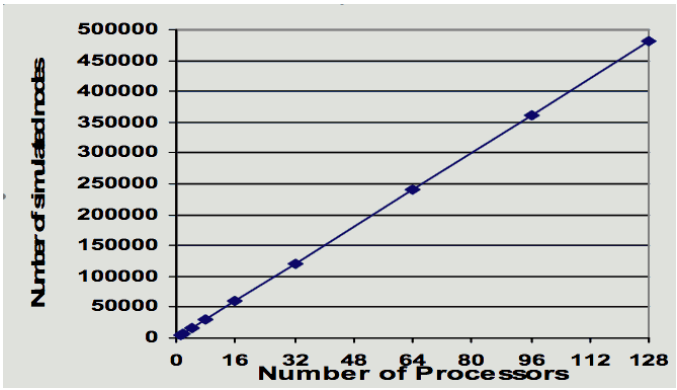
PDNS uses a conservative (blocking based) approach to synchronization. No federate in the parallel simulation will ever process an event that would later have to be undone due to receiving messages in the simulated past. This avoids the need to implement state saving in the existing *ns* code. The PADS research group at Georgia Tech has previously developed an extensive library of support software for implementing parallel and distributed simulations. The sofware has support for global virtual time management, group data communications, and message buffer management. It has support for a variety of communication interconnects, including shared memory, Myrinet and TCP/IP networks, and runs on a variety of platforms. By using this synchronization software for the parallelization of *ns*, we were able to rapidly modify the main event processing loop of *ns* to support the distributed time management functions needed to insure that no unsafe event is ever processed by any federate.

The functionality to enable distributed simulations can be broadly classified in two major categories: modifications to the *ns* event processing infrastructure, and extensions to the *ns* OTcl script syntax for describing simulations. In particular,

1) Any reference to a remote endpoint of a connection must be by network elements such as IP address and port number, rather than memory address pointers to the remote object. In a distributed simulation, there is no guarantee that a remote protocol object has a memory address or a representation on any given simulator instance.
2) Packet routing decisions may need topology information not present on a simulator instance, and thus some method must be designed to properly route packets in the presence of incomplete topology information. The obvious solution to this issue is the use of a *routing protocol*, such as *OSPF*, to compute routes. However, this approach reduces scalability due to the need for potentially large routing tables at each simulated node. *ns-2* already provides a choice between centralized routing, the use of simple models of distributed routing (distance vector and link state) and NixVector routing– a form of source routing that can retain and use a complete routing path from a source to a destination in a very compact form.
3) Some method of packet serialization and reassembly must be provided, to allow for the flow of packets from one simulator instance to another.
4) The distributed simulation features of the simulator



(a) Conceptual overview of PDNS (from [4])



(b) Scalability of PDNS (from [5])

Fig. 2. Parallel distributed *ns* (PDNS)

should not hinder in any way a simple sequential simulation.

5) It should be no more difficult to construct a distributed simulation than to construct a sequential simulation.

Figure 2(b) illustrates scalability results from running PDNS on a large Linux cluster consisting of 16 SMP machines with eight 550MHz Pentium III XEON processors. The eight CPUs of each machine share 4 GB of RAM, and each processor contains 32KB of non-blocking L1 cache and 2MB of full-speed, non-blocking unified L2 cache. The 16 SMP machines were interconnected via a dual Gigabit Etherent switch with EtherChannel aggregation. Our RTI software used shared memory for communications within an SMP, and TCP/IP for communications across SMPs. We were able to scale the sample "campus network topology" simulations to over 450000 nodes across 128 processors. PDNS achieved a speedup of a factor of 80 and a simulation speed of 6 million packet hops per second, on 128 processors [5]). The capacity was ultimately limited by memory to 8*4500 nodes/4GB.

B. Other Design Considerations

Our *ns-3* design will also consider the following:

1) *Object-design.* A network simulation tool designed for use by the networking research community must be easily extended to include new protocols, modifications to existing protocols, or new types of routing (just to name a few). To achieve these goals, we carry forward the core object-oriented *C++* design of *ns-2* with a large number of base classes that describe basic functionality, and subclasses implementing this functionality. For example, we have a base `Queue` class which describes the methods needed by all specific queue implementations. Then, subclasses of this `Queue` object implement various queuing methods, such as DropTail and RED. Similarly, a base `TCP` object describes and implements much of the functionality of the TCP protocol, but subclasses provide the specific functionality of the TCP variants, such as Reno, New Reno, and SACK. Further, extensibility can be achieved by the use of *callbacks* at various points in the simulation execution. For example, a callback between layers 2 and 3 could implement the functionality of a firewall or other network filtering devices without requiring modification of the implementation of layer 2 or layer 3.
However, the overall software architecture of *ns-2* needs revision. Specifically, the current OTcl/*C++* split-object paradigm of *ns-2*, while providing scripting flexibility at the user interface, introduces a number of problems. First, it is frequently cited as a barrier to learning and debugging the tool, because students are not familiar with object-oriented Tcl, the OTcl/*C++* glue is poorly documented, and disparate debugging tools are required. Second, the code introduces restrictions on how the *C++* objects can be combined, in ways that are not immediately obvious. For example, to use a particular Channel object with a MobileNode, the Channel must support the `add-node` OTcl method, although it is not specified by the *C++* interface and one must carefully read the MobileNode OTcl code to figure this out. Because the *C++* top-level interfaces are not very typed, a lot of *C++* objects perform downcasting, introducing hidden dependencies between *C++* objects. Since a primary complaint of the *ns-2* user community is the inability to build novel combinations of simulator objects, we plan to clean up these interfaces and explore the revision or elimination of the split-object framework, in favor of a purely *C++* design or one controlled only in limited fashion through scripting interfaces.

2) *Realism.* The basic design of the simulator should closely match the design of real networks and real network elements. When questions arise about the design of an object, function, or interface, deference should be given to how things will be implemented in real-life, unless there are strong performance or efficiency considerations. This design principle should greatly help in code portability and educational use. One key to achieving this goal is the support of important interfaces used in practice, such as the user-to-kernel interface

(sockets API) and the kernel-to-device-driver interface (i.e., IP to sub-IP). Further, there should be clear distinction between protocol layers in the software design, with well defined intra–layer exchange points. A protocol graph should be designed to facilitate multiplexing and demultiplexing decisions as packets flow up and down the stack. Simulator objects representing *packets* should consist of one or more *protocol data units* (*PDUs*) with sub–classed objects for each distinct protocol type. Simulation objects representing applications should interface with layer 4 protocols in a fashion familiar to Internet application developers, including calls for establishing connections, sending and receiving data, accepting or refusing incoming connection requests, and closing connections.

3) *Memory-efficiency.* The simulator should continue to support both *data–less* and *data–full* flows. In other words, an application model might specify to send 100,000 bytes of data to a remote endpoint, but the actual data contents are not meaningful and can be abstracted away. Current *ns-2* is optimized for data–less operation. In other applications, such as a simulated application modeling the *Gnutella* peer–to–peer protocol, applications must be able to specify search requests, replies, and ultra–peer information in the data contents. This capability is also needed for directly ported application code, whose data is meaningful and must be supported by the simulation. There should be no difference between data–less and data–full packets below the transport layer, and we plan on straightforward support of both in *ns-3*.
4) *Tracing.* To facilitate large–scale simulations, the amount of data logged and traced as part of the simulation statistics should be highly configurable by the simulator user. For example, when studying end–to–end behavior of TCP, the tracing of packet data at layers 2 and 3 at all hops along the TCP path may not be of interest. Thus, a user must be able to specify that only a particular layer's information is logged, and additionally can limit the logging to specific nodes, links, or applications. Some capabilities along these lines already exist in *ns-2*; e.g. the ability to trace only packets on an individual link. Further, the type of data recorded at each layer should be defined by the user. For example, when studying TCP performance, the contents of the *source port* and *destination port* fields may or may not be of interest, depending on the number of flows defined at each node. In this case, the user should specify whether or not these fields should be logged as part of the simulator output. We plan to add support to directly trace packet flows in standard trace file formats, so that tools such as `tcpdump` can process them.
5) *Statistics.* For ease of use, the simulation must provide a number of support objects to facilitate data collection during the simulation execution. *ns-2* already includes support for integration and averaging of random samples. We will extend this include objects for histograms, minimum and maximum tracking, probability distribution functions, cumulative distribution functions, and sequence versus time plots. We plan to add support for better statistics generation from similar objects in *GTNetS*.
6) *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. For example, the `Dumbbell` object would naturally create a dumbbell topology with a specified number of leaf nodes at each end, and with a specified bandwidth constriction factor at the bottleneck link. Other stock objects should include trees, meshes, stars, and random topologies of arbitrary size. We will incorporate such topology objects from *GTNetS*.
7) *Visualization* The simulator must support some form of visual animation of all or part of a simulation. This is useful for debugging and demonstrating the simulation to others. The network animator *nam* has been part of the simulator from the beginning, but outside of wireless extensions developed under NSF CONSER, has not been the subject of development for several years.

IV. Integration

A key step forward of our proposed *ns-3* project will be the level of integration that we will obtain, leveraging the vast amount of free, open source software and research projects available on the web. We have three specific goals in mind:

1) Extension of the simulation capability via integration with open source software (e.g., Ethereal packet analysis, Click/XORP routing);
2) Abstraction layers and interfaces for porting implementation code into the *ns* environment; and
3) Interfaces to allow users to easily migrate between simulation and network emulation environments.

An opportunity that most every simulation environment has missed is the opportunity to leverage the extensive amount of free, open source networking code within operating systems and applications. Typically, simulators re-implement protocols from scratch, leading to a costly software effort and divergence from actual implementation code. There are limited exceptions (the TCP code in QualNet [9], for example, is ported from BSD, and NCTUns [11] uses a kernel-reentrant programming paradigm to use actual Linux stack code), but predominantly, protocols are reimplemented for the simulation environment.

Furthermore, simulation code often does not interact well with real implementations. Most commonly, simulation implementations of protocols are rewritten for use as implementation code, often because the simulation code makes use of abstractions and simplifications not present in real systems. We are aware of a number of cases in which companies or organizations have written an abstraction library for an existing simulator such as OPNET [8] or *ns-2*, and also support the same library in an operating system, allowing software to be

written once and used in both implementation and simulation environments; the Naval Research Laboratory's `protolib` toolkit is an example of a publicly-available library of this type. This approach works well if a protocol implementation is written from scratch; however, it generally does not work so well when *existing* software, often written in a lower-level, non-object-oriented language such as `C`, is used.

In the *ns-3* project, we intend to focus on simulator design that facilitates the reuse of existing software and applications. Although there are dangers in relying exclusively on open source networking code (such as inheriting bugs and design peculiarities of, e.g., Linux TCP), there simply is no cost-effective substitute for reusing software packages that others have spent years developing and maintaining. Such an approach helps to meet our educational goals as well, since the simulation models mimic how the software is run in real implementations. Our team has experience in porting actual implementation code into the *GTNetS* and *ns-2* simulation environments, including BGP and OSPF software from the `quagga` open source routing suite, and application-level code that uses the sockets API. Insights from these past efforts will guide our development of more general support for such integration.

Specifically, we see the following key opportunities for software integration and reuse: i) ported *application code* using sockets API, ii) *routing protocols* such as XORP, `quagga`, and OpenBGP, iii) *network stack code* such as the Network Simulation Cradle [7], and iv) *tools to parse output data* such as tools that work on `libpcap` format traces such as `tcpdump` and Ethereal.

Finally, we recognize the significant research infrastructure advances of the past few years, with such projects as PlanetLab, Emulab, and WHYNET. These testbeds provide opportunities to explore protocol interactions in less controlled environments (such as cluster-based, remotely executed testbeds including Utah's Emulab) and to deploy long-running experimental services and overlays on the existing Internet (PlanetLab). Presently, Emulab uses *ns* scripting syntax to describe its experiments, and offers a version of the *ns* emulation environment to experimenters. There is presently no *ns* interface to PlanetLab. Our view is that *ns* should be flexible enough to run as simulations but be easily convertible (such as building with different flags) to run in an emulation environment. To achieve this, the *ns* emulation environment must be further developed (it is based on an older version of FreeBSD, does not fully support all *ns* protocol models, and has not undergone development for several years) and tested with Emulab and PlanetLab. Specifically, we envision that *ns* could run as an application within a PlanetLab slice, with the emulation interface tying into the PlanetLab Safe Raw Sockets API, and that *ns* could run on top of an Emulab virtual interface. Our project will coordinate with testbed projects to ensure that *ns* can successfully execute in those environments and is well documented enough for ease of use.

V. Project Processes

A vital component to a successful open source project is the establishment of good project development processes that encourage participation from the broad community. Our vision is that *ns-3* transitions to a self-sustaining research infrastructure project once this phase of NSF funding concludes.

A. Licensing and copyright

Although this subject is for further discussion, the project is leaning towards the use of the GNU Lesser General Public License (LGPL) as the license on the simulator core. There are two main reasons for this. First, there is a strong preference among some participants to ensure that derivative works of the simulator itself are contributed back to the project. Often, the GNU General Public License (GPLv2) is used in such circumstances. However, since composition of outside software is an explicit goal of the project, the LGPL is believed to provide more flexibility for bundling non-GPLed software with the core of *ns-3*.

The project is likely to continue the practice of allowing each individual software contributor to hold copyright and license his or her software to the *ns-3* project under acceptable terms for the project.

B. Development processes

We plan to adapt processes established at other successful open source projects (such as the Apache Foundation). This includes the following:

- adherence to clear coding standards;
- well-defined roles and responsibilities (management and technical tasks);
- processes for obtaining commit privileges and responsibilities;
- regular software releases;
- detailed patch review; and
- requirements for testing, documentation, and example scripts.

VI. Related Work

Besides *ns-2*, there are a number of other discrete-event network simulators. Prominent commercial tools include OPNET [8], QualNet [9], and MATLAB Simulink. A larger number of open source tools have been developed, including GloMoSim [10], NCTUns [11], *GTNetS*[12], OMNET++ [13], SSFNet [14], and JiST [15]. A recently developed prototype of interest is the yans simulator [16]. Of the open source tools, *ns-2* appears to have the most users and contributed protocol models, and the least restrictive licensing conditions for use.

We also observe the growing trend of the research community towards virtual machines (VNUML [17], IMUNES [18], Netkit [19], Xen [20], VMware [21], Bochs [22]) and cluster-based or distributed network testbeds (PlanetLab [23], Emulab [24], and WHYNET [25]). We do not suggest that *ns-3* replaces any of these techniques– such infrastructure-based testing is essential as part of the technology evaluation process, and the controlled environment of simulation is not

appropriate for all experiments. However, there remains a need for robust simulation tools, particularly to explore issues of scale and large protocol design spaces that are infeasible to accomplish on real testbeds. The challenge, we believe, is to develop simulation tools that integrate well with virtual machines, network testbeds, and actual implementation code, and we place a priority on this topic in our program. Other related projects which we plan to leverage have involved moving network kernel code to user-space, including Alpine [26], the BSD Network Stack Virtualization project [27], and the Network Simulation Cradle for *ns-2* [7].

VII. Conclusion

Despite *n*s-2's popularity, there is a critical need for a new project to perform core refactoring, integration, software maintenance, and extension of the simulator. Experience has strongly shown that issues like software maintenance (e.g., tracking compiler and operating system evolution), documentation, educational development, and code integration are lesser priorities for individual contributors focusing their limited time primarily on producing research output for publications. As a result, we have formed an NSF-funded community infrastructure project to develop a new major version of the simulator. We welcome the research and educational community's input as we define and perform this project; please contact us at `ns-developers@isi.edu`..

Acknowledgements

The authors thank Mathieu Lacage, John Heidemann, and Steven McCanne for discussions on the future of *ns* and recommendations for project organization.

References

[1] S. Keshav, "REAL: A Network Simulator," University of California at Berkeley, Berkeley, CA, USA, Tech. Rep., 1988.

[2] G. F. Riley, E. W. Zegura, and M. H. Ammar, "Efficient routing using nix-vectors (long version)," Mar 2000, technical Report GIT-CC-00-27.

[3] V. Naoumov and A. Gross, "Simulation of Large Ad Hoc Networks," in *In Proceedings of the 6th ACM Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2003.

[4] G. F. Riley, R. M. Fujimoto, and M. H. Ammar, "A Generic Framework for Parallelization of Network Simulations," in *Proceedings of Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, October 1999.

[5] K. Perumalla, R. Fujimoto, A. Park, and G. Riley, "Scalable rti–based parallel simulation of networks," in *17th Workshop on Parallel and Distributed Simulation*, June 2003, pp. 97–104.

[6] G. F. Riley and T. Jaafar, "Space–parallel network simulations using ghosts," in *18th Workshop on Parallel and Distributed Simulation*, May 2004.

[7] S. Jansen, "Network simulation cradle report," The University of Waikato, Tech. Rep., November 2003. [Online]. Available: http://www.wand.net.nz/pubDetail.php?id=199

[8] "OPNET Technologies, Inc." http://www.opnet.com.

[9] "Scalable Network Technologies, Inc." http://www.scalable-networks.com.

[10] "Global Mobile Information Systems Simulation Library," http://pcl.cs.ucla.edu/projects/glomosim/.

[11] "NCTUns Network Simulator and Emulator," http://nsl.csie.nctu.edu.tw/nctuns.html.

[12] G. F. Riley, "The georgia tech network simulator," in *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*. New York, NY, USA: ACM Press, 2003, pp. 5–12.

[13] A. Varga, "The OMNeT++ distrete event simulation system," Software on-line: http://whale.hit.bme.hu/omnetpp/, 1999.

[14] J. Cowie, A. Ogielski, and D. Nicol, "The SSFNet network simulator," Software on-line: http://www.ssfnet.org/homePage.html, 2002, renesys Corporation.

[15] "Java in Simulation Time (JiST)," http://jist.ece.cornell.edu.

[16] M. Lacage and T. Henderson, "Yet Another Network Simulator," in *submitted to Workshop on ns-2 (WNS2)*, Oct. 2006.

[17] "Virtual Network User Mode Linux," http://jungla.dit.upm.es/ vnuml/.

[18] "Integrated Network Topology Emulator/Simulator," http://www.tel.fer.hr/imunes/.

[19] "NetKit," http://www.netkit.org.

[20] "The Xen virtual machine monitor," http://www.cl.cam.ac.uk/Research/SRG/netos/xen/.

[21] "VMware," http://www.vmware.com/.

[22] "bochs: The Open Source IA-32 Emulation Project," http://bochs.sourceforge.net.

[23] L. Peterson, D. Culler, T. Anderson, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *In Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Oct. 2002, planetLab. [Online]. Available: http://citeseer.ist.psu.edu/peterson02blueprint.html

[24] "Network Emulation Testbed," http://www.emulab.net.

[25] "Wireless Hybrid Network Testbed (WHYNET)," http://chenyen.cs.ucla.edu/projects/whynet/index.php.

[26] D. Ely, S. Savage, and D. Wetherall, "Alpine: A User-Level infrastructure for network protocol development," in *In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, 2001, pp. 171–184.

[27] J. Elischer, "Implementing a Clonable Network Stack in the FreeBSD Kernel," in *In Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.