Winter 12-2012

SAFE: Simulation Automation Framework for Experiments

Luiz Felipe Perrone
perrone@bucknell.edu

Bryan C. Ward
Bucknell University, bcw006@bucknell.edu

Christopher S. Main
Bucknell University, csm024@bucknell.edu

Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

SAFE: Simulation Automation Framework for Experiments

L. Felipe Perrone
Christopher S. Main

Dept. of Computer Science
Bucknell University
Lewisburg, PA 17837, USA

Bryan C. Ward

Dept. of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599, USA

ABSTRACT

The workflow of a network simulation study requires adherence to best practices in methodology so that results are credible and reproducible by third parties. The opportunities for one to introduce errors start at model description and permeate the process through to the reporting of results. The literature indicates that even publications in respected venues include inadvertent mistakes and poor application of methodology. When experts are liable to fail, it is unreasonable to expect that students would fare any better. This paper presents a system designed to provide guidance for inexperienced users of the popular ns-3 network simulator. SAFE automates the workflow from the initialization of model parameters, to the parallelized execution of experiments, to the processing and persistent storage of output data, and to graphical visualization of results. We discuss the architecture and the implementation of the system in the context of similar contributions in the literature.

1 INTRODUCTION

The development of computer network protocols is a complex undertaking and, therefore, not for the faint of heart. In the process of implementing protocols, verifying their correctness, and evaluating their performance, researchers have often resorted to running computer simulations of these systems. Among several benefits, simulation enables one to evaluate systems under many different scenarios with absolute control and maximal observability.

For more than a couple of decades, however, it has become apparent that the intricacies of the simulation workflow have worked against the primary justifications for the use of the technology. The misuse of modeling and simulation methodology in the field of networking has led to an increasing discredit of simulation, which can be exemplified by a real anecdote. One of the authors, while writing code during the coffee breaks in a technical workshop, had the following exchange:

"Hi. What are you working on?"
"A simulator for wireless ad hoc networks."
"Why bother? You know the results are not going to have any connection with reality."

In an isolated context, this exchange would have been of little concern. What invited deep reflection is the fact that this punchline was delivered by someone who had been deeply involved with network simulation. Conversations like this have been happening with distressing frequency, as network protocol experts have grown increasingly suspicious of published simulation studies. Thankfully, the simulation research community has been well-aware of the problem and has engaged in a healthy investigation of the roots of the problem.

The landmark paper by Pawlikowski, Jeong, and Lee (2002) pointed out that this crisis of credibility could be resolved with the application of common-sense guidelines: the use of i.i.d. random number

generators and high quality simulation output data analysis. However, the roots of the problem went broader and deeper. Later on, Kurkowski, Camp, and Colagrosso (2005) showed that many published studies leave something to be desired in reproducibility, statistical rigor, and suitability of experimental scenario. The paper goes on to evaluate in finer detail how publications have failed to properly deal with important issues in the simulation experimental workflow. "The Incredibles" exposed a number of pitfalls that have traditionally caught network simulation researchers. These papers have analyzed a large body of publications from highly regarded sources: conferences such as the IEEE International Conference on Computer Communications (INFOCOM), the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), and journals such as the IEEE Transactions on Communications, IEEE/ACM Transactions on Networking, and Performance Evaluation Journal. It is fair to assume that the problems that undermine the credibility of network simulation studies extend well-beyond these venues and are at least as severe.

Borrowing from Goethe, we can say *"der Worte sind genug gewechselt, laßt mich auch endlich Taten sehn"*, or in English: *"we have enough* analyses*, its time to see* deeds*,"* as translated by Walter Kaufmann (von Goethe 1961). At this point, what the community needs are tools that supplement and enhance the expertise of the experimenter to provide guidance through the workflow of the process, while avoiding the known pitfalls. We are developing the *Simulation Automation Framework for Experiments (SAFE)* to address these needs, but also very importantly, so that it can be used as a tool in the teaching of computer networks. Our previous work (Perrone et al. 2008, Perrone et al. 2009) identified simulation workflow problems that can be addressed with the use of automation software and presented a proof-of-concept tool for a specialized simulator. Our current work is focused on developing production tools to serve the community of users of *ns-3*, a modern, popular network simulator.

The main contribution of this paper is the presentation of SAFE's conception, design, and implementation. We expect that the ideas and decisions that drive this project are transferrable to other domains and can have a positive impact in the way that the community thinks of end-user software for computer simulation. In a sense, our work could be seen as an effort to redefine the term *computer-aided simulation* to describe systems that guide the experimenter through the process from model construction all the way to output data analysis and visualization.

The remainder of this paper is structured as follows. Section 2 presents a historical perspective on the evolution of the concepts that led to SAFE. Section 3 discusses the design objectives and the architecture of SAFE, while Section 4 presents some of the broader implementation details. Section 5 situates the project in the context of related work, and finally, Section 6 wraps up the paper with a summary of lessons learned and directions for future work.

2 EVOLUTION

Network simulation models often consist of a collection of sub-models, which characterize the various components of the protocol stack, the node mobility models, and the radio propagation channel. Within this collection, the total number of model parameters can be unwieldy, but most often researchers are interested in simulation studies that vary a small subset of them. The workflow for these studies starts with the construction of the experimental scenario and its corresponding simulation model, the identification of the parameters with which to experiment (*"factors"*), and the specific values these parameters will be assigned (*"levels"*). This defines a collection of experiment *design points*, which constitute the *design of experiment space*. As the number of factors and levels grows, there can be a combinatorial explosion of design points, which would make the experiment unmanageable without computer assistance.

In order to support these large, parametric simulation studies with the SSFNet simulator (Cowie et al. 1999), a group from AT&T Research and Dartmouth College created a collection of scripts called *Scripts for Organizing 'Spiriments* (SOS) (Griffin et al. 2002). With relative ease, the SOS user can define experiment runs, launch their execution, extract output data, and save it in a database for persistent storage. Once runs terminate and data become available in the database, the entire history of the experiment is aggregated in

one single repository (a database), which scripts can query to build plots. SOS works well in performing these functions, but it requires the user to have expertise in understanding the format of the simulator's output, in writing Perl to create custom "extractors" (scripts that find the data of interest in the output stream), and of databases/SQL. While a graduate student or an experienced researcher might have little trouble ascending SOS' learning curve, undergraduate students don't fare as well.

A more suitable tool for supporting the needs of an undergraduate student would have a simple user interface and would require little to no programming effort in customization. Our SWAN Tools (Perrone, Kenna, and Ward 2008) was conceived to meet those goals; it was a web-based application that worked from a complete network simulation previously implemented by an experienced user. Although this simulation script could be complex and contain a large number of model parameters, only small a subset of them was exposed through the web-interface as customizable factors. SWAN Tools had users enter levels from the web interface, constructed the design of experiment space, dispatched each design point to be executed on an independent host computer, extracted from the simulation output streams only a predefined set of metrics, stored them in a database, and offered results through the web interface for visualization as text and by a very simple plotting utility. Although it was only a proof-of-concept, SWAN Tools taught us important lessons. The restrictive environment provided the guidance that inexperienced users of network simulation needed to create credible experiments. On the other hand, adjusting SWAN Tools to support the needs of experienced users required too much effort – almost every customization required programming knowledge in Ruby on Rails and "plumbing into the guts" of the system.

These two systems have given us first hand experience with the benefits in automating the simulation workflow. They made it clear that going forward with this type of project, the priorities should include comprehensive support for users at all stages of the workflow and different interfaces for users of different skill levels. Such projects will have a much stronger impact in our field if their design goals are well aligned with findings from the literature that identify the most egregious pitfalls in simulation.

Our work on SAFE, which is supported by the U.S. National Science Foundation, is creating computer aided simulation tools for the *ns-3* network simulator following the stated priorities. We expect that these tools will enable undergraduate and graduate students in computer networking classes to use the popular *ns-3* without having to get into the details of programming for the simulator or creating ancillary tools for experiment execution and output data processing. Next, we present the design and the implementation of the SAFE project.

3 SYSTEM ARCHITECTURE

3.1 Design Objectives

Many published simulation studies fail to report the complete scenario necessary for a third party to replicate the experiment, use models composed of incompatible pieces or which fail to include components that have interdependencies, determine the length of simulation in manners inconsistent with the study's goals, and do not use best practices for output data collection, processing, and analysis. These problems arise from individuals in the network simulation community not always having the required combined expertise in the areas of simulation methodology and network modeling. SAFE is constructed to address these issues, which have been identified by Pawlikowski, Jeong, and Lee (2002) and Kurkowski, Camp, and Colagrosso (2005).

SAFE is designed to meet additional goals. It should be easy to use by students and novice users, that is, it should offer simple interfaces for model and experiment configuration, for simulation execution, and for output data visualization, and it should automate output data analysis. (The configuration interface should protect these users from mistakes, validating the users' choices against best practices.) At the same time, SAFE should be flexible to use by power users, that is, it should allow these users to bypass some of its safeguards while allowing them to use mechanisms that make it easy to stage experiments and visualize their results. The system should support multiple users and provide a protected compartment for each

of them in which to store input configurations and output data. Finally, the system should implement the *multiple replications in parallel* (MRIP) paradigm supporting networks of workstations and compute servers with multi-core processors, as discussed by Pawlikowski (2003).

SAFE implements differentiated user interfaces by supporting two user stories, one for *power users* and another for *novices*. Our working definition of power user is someone who prefers command-line interfaces, can write XML experiment descriptions and model configurations, works comfortably with system tools for remote login and file transfer (`ssh`, `sftp`, `scp`), and can write C++ simulation scripts for *ns-3*. A novice user is someone who will work with web-based interfaces, will not have expertise to write configuration code in XML, will only run experiments with previously written C++ simulation scripts stored in the system, and will rely on automated output data processing. In the remainder of this section, we describe how power users and novices use the system, and give an overview of the architecture that supports these user stories.

3.2 Story for Power User

The requirements that power users make of the SAFE system are driven less by the niceties of a sheltering user interface than by access to the core functionality of the system. This type of user works in a local *ns-3* installation, which they will have downloaded from http://www.nsnam.org. They develop and/or customize models, and write their own simulation scripts. We assume that these scripts will be initialized with levels for experimental factors via a collection of command line options constructed when the program is executed.

Once the code base is compiled and debugged, the simulation script is ready for use. The user writes a file containing details of the execution of the experiment, that is, defining the design of experiment space and the termination condition for the simulation. Next, the user starts the execution by invoking a SAFE script to launch the experiment. The system deploys the code in the collection of worker machines specified in its configuration and each one contributes to the computational effort of covering the design of experiment space by running simulations of design points. Each simulation run generates output data that is relayed to SAFE for persistent storage, and executes until the chosen termination condition is met. The simulation data is logged to a database, which can be queried by the user directly or via automated scripts that extract data and build plots.

The main functionality that SAFE provides for power users is support for the execution of experiments and for the safekeeping, analysis, and visualization of output data. The system helps one to record experiment scenarios in persistent storage together with output data, offers reliable mechanisms for the processing of results, and executes the experiment using the MRIP paradigm. Since the usage constraints for this type of user are few, users can express freely the scenarios they envision and also make procedural mistakes. The user must have expertise in writing valid *ns-3* simulation scripts and in debugging them.

3.3 Story for Novice User

One of the main motivations for the development of SAFE is to support educational uses of *ns-3*. For every simulator, there is a learning curve that users have to ascend in order to be able to make effective use of the tool's features. In the context of an undergraduate or graduate course on computer networks, it is often desirable to have students use simulations rather than real systems so that they can have full control over their experimental scenario. Training students to use simulators correctly, however, is a task that requires substantial time. For this reason, it is highly desirable to offer students simplified interfaces that abstract away most complexities and provide plenty of guidance in staging experiments and analyzing their results.

For the novice user, SAFE provides a web-based interface that only allows the execution of experiments previously crafted by experienced programmers, which have been installed in the system. The user points a web-browser to a given URL, presents credentials to authenticate with the system, and finds a collection of experiments that can be customized within pre-specified constraints. These experiments are associated with configuration files that determine which factors are exposed through the web interface and how the user

supplied levels are validated. Once all factors have been associated with levels, the design of experiment space is defined and the execution of the experiment can be started. The data produced by simulations associated with each design point are logged to the database. Through the web interface, the user can explore the data visually, download portions of the data, and create custom plots to be saved in the database and/or downloaded in different graphics formats.

The protections against common errors that SAFE offers to this type of user are multiple. First and foremost, novices do not write *ns-3* simulation scripts of their own. This does away with the need to learn to program in the simulator's environment, which would expose the user to issues of validation and verification. The user selects one instance among several *ns-3* simulation scripts that have been carefully constructed and debugged by experts, and only defines the design of experiment space. Second, the system validates each level in the experiments' design points against pre-defined model constraints, what guarantees that the specific values chosen for levels are always within the permissible ranges. This type of safeguard, for instance, would not let a user create a simulation with an IEEE 802.11b wireless model using bandwidth outside the range from 1 Mbps to 11 Mbps. Third, whenever models require levels to be chosen from a discrete list of levels, the interface restricts the user to pick from that specific list. Using the IEEE 802.11b example again, the user would be constrained to select a bandwidth level from the list $\{1, 2, 5.5, 11\}$ Mbps. The other protections for novice users include the thorough recording of experimental scenario, the organization of experiment output, and the execution of experiments until enough samples are collected to yield the correct coverage for specified confidence levels.

3.4 The Big Picture

In order to meet our design objectives and to support the user stories described above, we have created a system around three main structural components, which are described as follows:

- One database that aggregates all the data pertaining to each experiment (input configuration and output data). This database enables the systematic organization and the persistence of the experimental data.
- One *experiment execution manager* (EEM): a server-side process that works as a central coordinator that implements functionalities behind user interfaces, intermediates access to the database, dispatches replications of simulations to worker machines, and processes output data that comes back from replications. The EEM is aware of a collection of worker machines that are registered with the framework as compute servers.
- Multiple *simulation clients* (SC): a client-side process that run on each worker machine. Each SC requests design points to the EEM, starts *ns-3* runs to simulate it, and relays output data to the EEM.

SAFE implements differentiated user interfaces by supporting two user stories, one for *power users* and another for *novices*. Our working definition of power user is someone who prefers command-line interfaces, can write XML experiment descriptions and model configurations, works comfortably with system tools for remote login and file transfer (`ssh`, `sftp`, `scp`), and can write C++ simulation scripts for *ns-3*. A novice user is someone who will work with web-based interfaces, will not have expertise to write configuration code in XML, will only run experiments with previously written C++ simulation scripts stored in the system, and will rely on automated output data processing.

Both types of users must first authenticate with SAFE before using any of its functionality, each using their choice of interface. The users' credentials are kept in one same database, regardless the user interface through which the user accesses the system. Authentication is used not only to grant access to system resources, but also to create individual data compartments for each user's experiment data.

The system is designed around a common *server* backend, which provides all the basic functionality that is exposed to the user in the guise of the two different interfaces. Figure 1 shows SAFE's general

structure and outlines the sequence of steps in the execution of an experiment. In the following description of how the system works, the numbers in parentheses correspond to the numbered balloons in this figure.

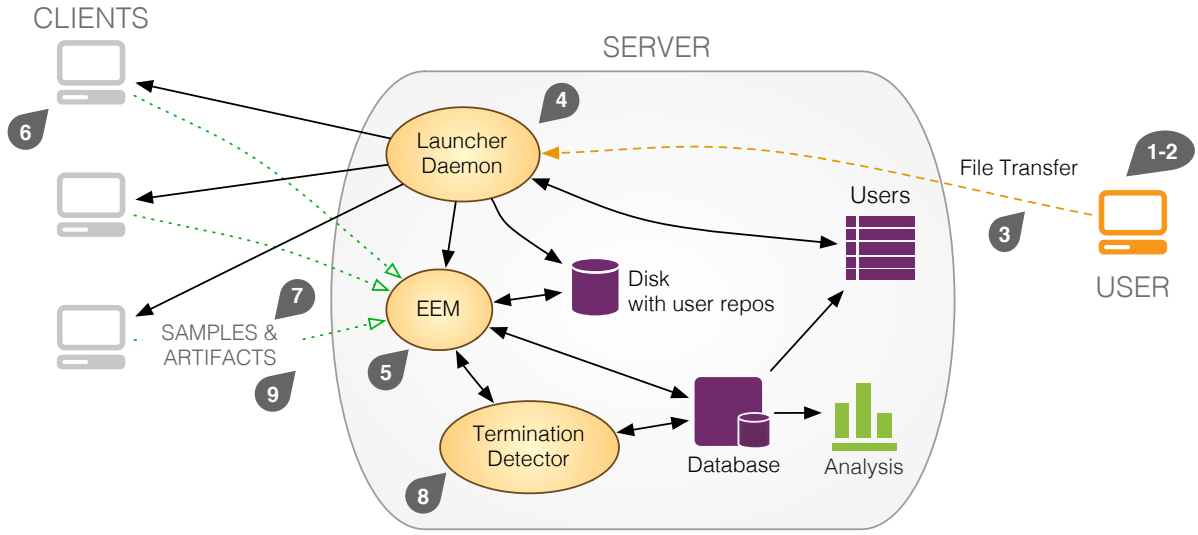

Figure 1: High level architecture of SAFE.

The process starts with the user writing or obtaining an *ns-3* simulation script for the experiment and placing it in a directory within the local *ns-3* installation (1). Next, in (2), the user either writes or has the system generate an experiment configuration file in the *ns-3 Experiment Description Language* (NEDL), which is an XML-based language designed for SAFE (Hallagan 2010). NEDL is constructed to express the design of experiment space. The main elements provided by the language are as follows:

- `factorlist` aggregates the list of experiment factors,
- `conditions` defines the specific levels associated with each factor,
- `exclusionrestriction` and `linkingrestriction` defines combinations of factors and levels that should be not be included in the design of experiment space,
- `termination` defines the type of simulation and the termination condition (confidence level and precision for steady-state simulations, and horizon for terminating simulations).

The following step initiates the execution of the simulation script (3). Up until this time, all files associated with the experiment reside on the user's development machine. The entire *ns-3* installation in this machine is archived and compressed so that it can be transferred to the server. Before the actual transfer takes place, however, the user's credentials are checked and, only upon success, the *launcher daemon* (LD) running on the server receives the file bundle and stores it in a local compartment for that specific user associated to a unique identification for the experiment. These measures serve to record the environment of the experiment (the simulator code base and configuration files) so that it may be faithfully replicated in the future, possibly by third parties. The server replicates this bundle in each designated *client*, or worker machine and each one invokes the build script for its *ns-3* installation (4).

The LD initiates the execution of the experiment by instantiating the EEM with the experiment description file received from the user and also a *termination detector* (TD) process (5). Using the information in this file, the EEM generates the design of experiment space and then waits for requests for design points. In the meanwhile, each client runs a bootstrap program to detect how many cores it has and then it creates one instance of the SC process for each core (6). Figure 2 shows that as SCs are started, they register themselves with the EEM, request a simulation to run (that is, a design point), and wait for the EEM's response. The arrival of a design point causes the SC to start an *ns-3* run locally, in the worker machine.

As the simulation executes, for each sample of a metric of interest, *ns-3* generates a tuple {*expid, mid, time, value*}, which corresponds to the unique identification for the experiment, a unique identification for the corresponding metric, a simulation time value, and the metric value, respectively. This tuple is sent from *ns-3* to the SC, which packages it into a *result message* that it sends to the EEM. As result messages are received by the EEM, it records the data in the database (7). When the termination condition for the simulation run is reached, the EEM sends a message to the SC, which in turn, terminates the simulator. This process is repeated until all design points have been simulated; at that point, the EEM sends a message to terminate all SCs.

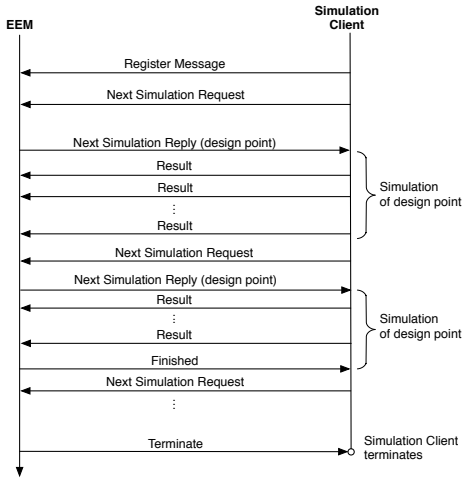

Figure 2: Communication protocol for EEM and SCs.

As samples accumulate in the database, the TD process evaluates whether there is enough of them to compute confidence intervals of each metric meeting the confidence level and precision specified by the user. Once the termination condition is ascertained, the TD communicates the fact to the EEM, which then proceeds to send termination signals to all the clients (8). Before the experiment is completely finalized, it is important to verify whether *ns-3* simulations have generated output files as additional artifacts (it is common, for instance, to have simulations record the packets that pass through a simulated network interface to `pcap` files). When that is the case, those files archived, compressed, and sent from clients to the EEM, which archives them as part of the experiment's output data. (9)

In order to implement authorization controls from the web interface, SAFE needs to manage its own internal collection of user credentials, as indicated in Figure 1. While the use cases for power user can rely on system level authentication, novice users accessing SAFE over the web might not even have system accounts.

4 IMPLEMENTATION

Usability is one of the main requirements driving SAFE's implementation. Our interpretation of this requirement has implications at multiple levels. First and foremost, SAFE must be comfortable to use through both the power user and the novice interfaces. Second, SAFE must be easy to deploy and work on the platforms most favored by *ns-3* users. Finally, SAFE must be easy to maintain and extend. These requirements have steered us toward some important choices, which are reflected in the early design described by Ward (2010). The framework is developed in the Python programming language and its central components rely on well-established, open source software such as:

- Twisted – http://www.twistedmatrix.com: an event-driven network programming framework, which provides a wealth of useful components that implement protocols and client/server applications.

This framework supports most of the network communication between SAFE's components such as the file transfer from users' computer to the SAFE server and from the server to client computers.

- Django – http://www.djangoproject.com: a framework based on the *model-view-controller* (MVC) design pattern, which supports the agile development of database-centered web applications. This framework is the keystone of SAFE's web-based interface and database storage.

4.1 The SAFE Server

As shown in Figure 1, the SAFE server comprises multiple components. The Launcher Daemon (LD) is responsible for transferring files associated with the experiment from a user's computer to the SAFE server and for kickstarting other server processes. The connection between LD and a power user is implemented in Twisted, using a library for secure file transfer between user and server (`sftp`). The user's credentials are authenticated against information stored internally in SAFE. If the authentication succeeds, the files are transferred to into a compartment for the specific user on the server's disk space. We opted to handle authentication directly so as to use the same mechanism and credentials via the two interfaces for the framework (web and command line).

The story for novice users is implemented differently: an experienced user sets up a repository for this use case with one compartment for each experiment that can be configured through the web interface. Someone without any knowledge of *ns-3* programming can access the server through a web browser, select an experiment, and go through configuration dialogs that expose only a small, manageable set of experimental factors to be initialized. When supported by mechanism for simulation output data visualization through the web browser, this feature has the potential to enable the extensive use of network simulation in the classroom, particularly in undergraduate courses. The SAFE server supports data analysis and visualization allowing users to construct representations of the data visually, through the web browser, and to save them in different graphical formats.

The LD supports both user stories by deploying, on client worker machines, the files in a disk compartment that holds the *ns-3* simulator and experiment configuration files. The actual simulations are performed in the client workers under a special-purpose user identity associated with SAFE. This guarantees that all simulations dispatched across a network of workstations and compute servers are authorized to access system resources. Currently, the LD relies on Fabric (http://fabfile.org), a Python library for secure application deployment over the SSH protocol.

Another responsibility of the LD is to start additional processes that run in the server, namely, the experiment execution manager (EEM) and the termination detector (TD). The EEM is the central component in the server. It processes experiment description files to generate design points, dispatches simulation runs to client computers, and receives and stores output data generated by simulations. The EEM does not perform complex, computationally expensive tasks, but instead serves as coordinator and communication broker for the processes that perform the heavy work.

In order to play well its role without becoming a bottleneck in the framework, the EEM must be highly responsive and available. This type of application is often implemented using the *reactor* design pattern (Schmidt 1995). In this event-driven programming model, the application registers functions to handle each type of event (known as *callback*) and enters a main loop where it simply waits for events to occur. Effectively, events and callbacks are analogous to hardware interrupts and their software handlers. The Twisted library (Fettig 2005) implements the reactor design pattern and the EEM relies on it to handle message exchanges with other processes, such as the TD and the simulation client (SC), to handle queries from the database, and to handle additional events.

While the EEM starts simulation runs, the TD works toward terminating them. Among the egregious problems in the credibility of simulation studies are the use of samples of output data collected during model "warm up" (or transient) phase and also terminating the runs before enough valid samples have been collected (Pawlikowski, Jeong, and Lee 2002). To address these problems, which arise in experiments with *steady-state simulations*, SAFE will provide two mechanisms. First, we are investigating methods for

automatic steady-state detection – having the time when metrics enter steady-state will allow for accurate deletion of data collected during transients. Second, we are investigating methods for detecting when enough samples have been collected to yield estimates of metrics with a chosen confidence level and precision. In the architecture of the system, the role of the TD is to encapsulate algorithms to analyze globally the samples produced by multiple replicas of a design point's simulation. Once the TD determines that the simulation of a design point has met its termination condition, it notifies the EEM, which in turn communicates with SCs to shutdown gracefully their *ns-3* runs. Since the discussion of steady-state and termination detection algorithms is beyond the scope of this paper, we refer the interested reader to Pasupathy and Schmeiser (2010) and Pawlikowski (1990).

4.2 The Simulation Client

In comparison with the EEM, the simulation client (SC) is a much simpler component of SAFE. As discussed in 3.4, after registering with the EEM, the SC requests a design point to run. Once the response arrives, the SC spawns a new process, which encapsulates the *ns-3* run. The SC passes to the *ns-3* script, as command line parameters, the levels that constitute the design point for the run.

As the simulator runs, the SC stays out of the way, until it receives communications from the EEM or from *ns-3*. When the server has enough data to terminate the simulations associated with a given design point, any related runs are shut down. If the simulations have created artifacts in the file system of client workers, those are sent back to the server for archival, followed by a request for another design point. When the server tells SCs that the experiment is completed, they first terminate their *ns-3* runs and then terminate themselves. The SC also listens for results from *ns-3*, which are relayed to the EEM. Again, Twisted's reactor design pattern is useful in the implementation of the SC, as it handles asynchronous events from multiple sources.

5 RELATED WORK

The literature reports on several frameworks that share with SAFE common aspects in terms of motivation, goals, and implementation. Among these, Akaroa2 (Pawlikowski 2003) stands out as a major turning point toward automating the simulation experimental process to address issues of credibility. Akaroa2 includes support for executing simulation experiments on networks of workstations and compute clusters according to the the MRIP paradigm. This control framework automates the analysis of simulation output data and the determination of run length. Although it has been adapted to work with popular network simulators, such as *ns-2* (http://www.isi.edu/nsnam/ns/) and OMNeT++ (http://www.omnetpp.org), we didn't try to leverage it for this project because its licensing restrictions would constrain our ability to redistribute the work as open source software. Instead, we have chosen to create for *ns-3* our own implementations of functionality offered by Akaroa2, although in a broader scope.

Another very interesting, albeit much younger, network simulation framework is STARS, which aims at conducting statistically rigorous experiments (Millman, Arora, and Neville 2011). STARS implements the MRIP paradigm with OMNeT++ as the underlying simulator. The architecture of the framework consists of a central manager, a central data store, and numerous workers that carry out the simulation runs. As the simulations begin to generate results, MATLAB (http://www.mathworks.com/products/matlab) is used to test for both stationarity and ergodicity of selected metrics. A core feature of STARS is its ability to automatically launch a new experiment if the current experiment fails to produce sufficient ergodic data. The experimenter can either continue to wait until the framework produces a sufficiently large dataset, or define an upper limit on the number of runs that can be conducted.

Outside the area of network simulation, we have found also frameworks that seems closely aligned with the scope of SAFE's goals and functionalities. A notable example is JAMES II (Ewald et al. 2010), which provides researchers in computational systems biology with support for modeling, experiment design and execution, run length control, experiment storage and retrieval, and data visualization and analysis. The

concept of *plug ins*, which is widely applied in the architecture of JAMES II, imbues the framework with a great deal of adaptability. We have learned from this architecture and, in the development of SAFE, we are working toward defining clear interfaces between the system core and ancillary components, which will function as plug ins.

SAFE differs from most frameworks for network simulation not only by offering comprehensive support throughout the experimental process. Arguably, one of its more important features is the dual interface (with a common backend) to address the needs of power users and novices alike. In order to be able to make good use of a complex network simulator with minimal investment of training time, novice users need all the help they can get. However, it is important to remember that power users also need guidance and support, as evidenced by the literature on simulation credibility.

6 CONCLUSION

The literature offers plenty of evidence that users of network simulation make mistakes. The analyses of the causes of the crisis of credibility have brought to light a comprehensive list of failures in methodology that have compromised published simulation studies. Increasingly, the community has been applying the lessons from the credibility literature toward the construction of frameworks that shield the experimenter from error prone tasks. By automating the experimental process, one can provide higher level of abstraction interfaces that steer the user away from known pitfalls in methodology. We view the development of such "augmented" simulators as defining *computer-aided simulation* tools, which will likely lead users to produce more credible simulation studies.

The goals of the SAFE project include implementing these ideas for the *ns-3* simulator. In our view, the project is creating a first version of support infrastructure that the community will be able to build upon in future years. Our main contributions, at this point, have been the general architecture of the framework, the definition of an infrastructure to support the needs of both network simulation novices and power users, and a working implementation that can be put to practice as soon as it is released to the public. Just like *ns-3*, SAFE is free software, licensed under the GNU GPLv2 license, and will be publicly available for research, development, and use from http://www.nsnam.org.

ACKNOWLEDGMENTS

This project is supported by the National Science Foundation (NSF) under Grant Nos. CNS-0958139, CNS-0958142, and CNS-0958015. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors' alone and do not necessarily reflect the views of the NSF.

We acknowledge the contributions of former Bucknell undergraduate Andrew W. Hallagan to the development of supporting languages for SAFE, in his honors thesis. Also, we are grateful to colleagues who participate in development and/or have participated in discussions related to our project: Tom Henderson (University of Washington/Boeing); Mitch Watrous (University of Washington); Mathieu Lacage, Daniel Camara, and Alina Quelheihac (INRIA Sophia Antipolis Méediterranée).

AUTHOR BIOGRAPHIES

L. FELIPE PERRONE is an Associate Professor of Computer Science at Bucknell University. He holds Ph.D. and M.Sc. degrees from the College of William & Mary (USA), as well as an M.Sc. and the degree of Electrical Engineer from the Universidade Federal do Rio de Janeiro (Brazil). He has served the simulation community as Associate Editor of the ACM TOMACS and in several organizational roles in conferences, including Proceedings Editor of the Winter Simulation Conference 2006, and Program and General Chair of SIMUTools (2009 and 2010, respectively). His interests include modeling and simulation, computer networks, and high performance computing. His email address is perrone@bucknell.edu and his web page is http://www.eg.bucknell.edu/~perrone.

CHRISTOPHER S. MAIN is an undergraduate student pursuing a B.S. in Computer Science, at Bucknell University (USA). In addition to having strong interests in web applications, data mining, and data visualization, he is passionate about environmental studies. His email address is csm024@bucknell.edu.

BRYAN C. WARD is a Ph.D. student in the Dept. of Computer Science, at the University of North Carolina Chapel Hill. He is advised by James H. Anderson and works with the Real-Time Systems Group on multi-processor real-time locking protocols. Bryan earned a B.S. in Computer Science and Engineering and a B.A. in Mathematics Computer Science from Bucknell University (USA), where he worked on tools for automating network simulation experiments. His email address is bcw@cs.unc.edu and his web page is http://www.cs.unc.edu/~bcw.

REFERENCES

Cowie, J. H., H. Liu, J. Liu, D. M. Nicol, and A. T. Ogielski. 1999, June. "Towards Realistic Million-Node Internet Simulations". In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '99)*.

Ewald, R., J. Himmelspach, M. Jeschke, S. Leye, and A. M. Uhrmacher. 2010. "Flexible Experimentation in the Modeling and Simulation Framework JAMES II-Implications for Computational Systems Biology". *Briefings in Bioinformatics* 2 (3): 290–300.

Fettig, A. 2005. *Twisted Network Programming Essentials*. 1st ed. O'Reilly Media.

Griffin, T. G., S. Petrovic, A. Poplawski, and B. J. Premore. 2002. "SOS: Scripts for Organizing 'Speriments". Available at ⟨www.ssfnet.org/sos/README⟩. [Accessed May 29, 2012].

Hallagan, Andrew W. 2010. "The Design of XML-based Model and Experiment Description Languages for Network Simulation". Honors Thesis, Bucknell University.

Kurkowski, S., T. Camp, and M. Colagrosso. 2005. "MANET Simulation Studies: The Incredibles". *ACM SIGMOBILE Mobile Computing and Communications Review* 9 (4): 50–61.

Millman, E., D. Arora, and S. Neville. 2011, March. "STARS: A Framework for Statistically Rigorous Simulation-Based Network Research". In *Proceedings of the 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications (WAINA)*, 733–739.

Pasupathy, R., and B. Schmeiser. 2010, December. "The Initial Transient in Steady-State Point Estimation: Context, a Bibliography, the MSE Criterion, and the MSER Statistic". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 184–197. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pawlikowski, K. 1990. "Steady-state Simulation of Queueing Processes: A Survey of Problems and Solutions". *ACM Computing Surveys* 22 (2): 123–170.

Pawlikowski, K. 2003, March. "Towards Credible and Fast Quantitative Stochastic Simulation". In *Proceedings of the International Conference on Design, Analysis and Simulation of Distributed Systems (DASD '03)*. Orlando, FL, USA.

Pawlikowski, K., H.-D. J. Jeong, and J.-S. R. Lee. 2002, January. "On Credibility of Simulation Studies of Telecommunication Networks". *IEEE Communications Magazine* 40:132–139.

Perrone, L. F., C. Cicconetti, G. Stea, and B. C. Ward. 2009. "On the Automation of Computer Network Simulators". In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques (SIMUTools '09)*, 1–10. ICST, Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Perrone, L. F., C. J. Kenna, and B. C. Ward. 2008. "Enhancing the Credibility of Wireless Network Simulations with Experiment Automation". In *Proceedings of the 2008 IEEE International Conference on Wireless & Mobile Computing, Networking & Communication (WiMob '08)*, 631–637. Washington, DC, USA: IEEE Computer Society.

Schmidt, D. C. 1995. *Pattern Languages of Program Design*, Chapter Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching, 529–545. New York, NY, USA: Addison-Wesley Professional.

von Goethe, J. W. 1961. *Goethe's Faust (Walter Kaufmann, trans.)*. New York, New York, U.S.A.: Doubleday.

Ward, Bryan C. 2010. "A Framework for the Automation of Discrete-Event Simulation Experiments". Honors Thesis, Bucknell University.