On The Automation of Computer Network Simulators

L. Felipe Perrone
Dept. of Computer Science
Bucknell University
Lewisburg, PA, U.S.A. 17837
perrone@bucknell.edu

Claudio Cicconetti,
Giovanni Stea
Dip. Ingegneria
dell'Informazione
Università di Pisa
Via Diotisalvi, 2
56122 Pisa, ITALY
{c.cicconetti,
g.stea}@iet.unipi.it

Bryan C. Ward
Dept. of Computer Science
Bucknell University
Lewisburg, PA, U.S.A. 17837
bryan.ward@bucknell.edu

ABSTRACT

Simulation has been an important resource for functional and performance analyses of computer networks. Although the number of widely adopted network simulators is small, new ones continue to be created to address gaps in the functionality of existing tools. It can be argued, however, that the greatest need of the scientific community is to raise the credibility of published simulation studies. In this paper, we show that this need can be addressed by enabling network simulators to provide fool-proof automation of the experimental process. Ideally, the simulator's interface would provide users with an environment to minimize set up time for experiments and to guarantee their reproducibility, and to safeguard the statistical rigor of data analysis. We demonstrate that advances toward this goal have been made by three different tools. Our contributions in this paper culminate with the derivation of requirements for automation tools from recent literature and from our own experience in tool construction. Once these requirements are fulfilled, network simulation tools can have a stronger impact in education, in carrying out large simulation studies, and in enhancing the credibility of simulation results.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation support systems—*environments*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*statistical software*; D.2.6 [**Software Engineering**]: Programming environments—*performance measures*

General Terms

Experimentation, Measurement, Performance

Keywords

Simulation tools, network simulation, best practices

SIMUTools 2009 Rome, Italy.

1. INTRODUCTION

It is undeniable that stochastic simulation has been of crucial importance in the development and in the analysis of protocols for computer networks. Simulation has been used to produce qualitative results in the verification of the correctness of protocol interactions. It has also been helpful in producing estimates of various metrics to predict, quantitatively, the performance of networks under user-defined and user-controlled conditions. There are inherent challenges in carrying out simulation studies that produce solid scientific value, however.

We have learned from the literature that a number of procedural difficulties stand in the way of the production of credible simulation-based studies of computer networks [1, 2]. These papers enumerate problems in methodology that cast doubts on the accuracy of simulation studies. Extrapolating from the observations that these and other related papers have brought to light, we can state a hypothesis to explain what lies at the root of the problem: *The level of complexity of rigorous simulation methodology requires more from networking researchers than they are capable of handling without additional support from software tools.*

Consider the most cited network simulators in research literature: *ns-2* [3], GloMoSim [4], Qualnet [5], and OPNET [6]. The learning curve associated with the simple application of these tools is non-trivial for all but the savviest users. To aggravate the problem, most often, the needs of network research lead users to customize one or more components of the simulator before they can apply it in their studies. This customization requires intimate knowledge of the framework upon which the simulation engine is built, as well as a clear understanding of the interactions between components of the larger simulation model. The process demands time, effort, and skills that potential users cannot always contribute and, commonly, results in solutions that aren't reusable or accurate. In this paper we focus on three use-cases which can benefit from advances in the level of support offered by network simulation tools: *education*, the *execution of large simulation studies*, and the *reproducibility of scientific results*.

First, we consider the user of network simulation in the education of graduate and undergraduate students. At the University of Pisa, students use the *ns-2* network simulator to study performance evaluation fundamentals in a class in their fifth year (that is, their last). Experience has shown that the time students spend on activities outside the core

of their project assignment is about 80% of the total. Ideally, students are supposed to concentrate on implementing a scheduler and on understanding its performance through the statistical analysis of data produced by simulation experiments. What happens in practice is that they spend more time than it is reasonable learning to use and to customize the simulator to produce the data of interest. They also have to spend time creating custom tools to parse simulator output and compute statistics from it. The accuracy of the results they obtain is as limited as their knowledge of rigorous statistical methodology. Due to the constrained time for the assignment, this required investment in set up for the study compromises the achievement of the main goal of the simulation activity.

Second, we discuss the suitability of current simulation tools to the realization of large simulation studies. Simulation is often used as a resource for testing some kind of hypothesis about a system. It is also often applied in the evaluation of the sensitivity of the system to a *potentially large* number of parameters. When the latter is the case, a study requires large numbers of simulation runs, under a number of different conditions that challenges one's organizational skills, and produces inordinate amounts of output data. More to the point, when such studies are carried out by teams, rather than single persons, the lack of organization in the output data may considerably slow down the progress of the whole study.

Finally, we submit that there exists a pressing need for the development of solutions to address problems that compromise the *reproducibility* of published simulation studies. All too often, as indicated in [1], the experiments described in a large number of papers on network simulation cannot be reproduced by third parties. The causes behind this effect include the lack of space in publication vehicles to enumerate all conditions in the scenario, but also lack of attention to detail. As a result, scientists find it difficult (if not impossible) to directly build upon published results and are left with no option but to try to recreate entire studies from vague information, without much guarantee of success.

We argue that the needs of the three use-cases above can be addressed by standardizing the network simulation workflow with the use of automation tools. As long as the network simulator provides some measure of support, one can construct around it a framework to organize scenarios for simulation experiments, execute simulation runs, process output data, and store results in a system that guarantees access in future references. The automation tools can be designed as a higher layer of abstraction that augments the functionality offered by the underlying network simulator. In the sense that a high-level programming language creates a safer environment for the development of programs, a network simulation automation tool can meet the needs of the less experienced user. At the same time, it can help the more experienced user by enforcing best-practices that enhance the credibility and the reproducibility of the simulation study. The use of automation tools would create common benefits to both constituencies such as reduced set up times for experiments and strict adherence to methodology.

Our contributions are threefold. We analyze existing open source network simulators with respect to the level of support they provide for the creation of automation tools. We discuss three tools that aim to automate the simulation workflow and reflect on their strengths and shortcomings. Finally, we reflect upon on our experience and evidence from the literature to present requirements that would produce significant advance in the development of automation tools.

The remainder of the paper is structured as follows. Section 2 discusses the complexity of the workflow of the simulation process in its current context. Section 3 discusses the current state of development of different network simulators via case studies of well-known projects. Section 4 describes tools for experiment automation and statistical analysis highlighting the impact of their features on the usability of networks simulators and on the credibility they can impart to simulation studies. Section 5 concludes the paper by summarizing what we identified as desirable features for simulation automation tools.

2. THE SIMULATION WORKFLOW

Computer simulation is a powerful technique that has been in use for nearly as long as computers have been around. Nonetheless, a considerable number of people are skeptical about it and for a very good reason. Simulation goes well beyond "writing some code to test a few ideas." In order for simulation results to be credible, they must have been produced by a sequence of actions that follows the established methodology. The complete process is enumerated and described in detail in [7]. In this section, we consider a higher level abstraction of these steps with the goal of identifying opportunities for the application of automating tools to aid in the process.

A simulation study starts with the definition of its objectives, that is, what one wants to learn from the study. For instance, whether a new protocol for wireless ad hoc routing outperforms existing ones. Before that kind of conclusion be reached, however, one must have defined a set of criteria to assess the performance of the routing protocol. This involves the definition of *metrics* to characterize what "performance" means for the specific goals of the study. To illustrate this and other points in this section, we use the example of a mobile wireless ad hoc network simulation. We might say that protocol X is better than another protocol Y, if a network built with X reaches higher data throughput than a network built with Y, with all other operational conditions being equal.

Once objectives are defined, the next step involves creating *models* that abstract away some of the complexity of the real system. The models must retain enough detail to allow for the goals of the simulation study to be met. Modeling is a complex activity; one that requires a good dose of insight and a fair amount of experimentation. The model chosen to represent a wireless ad hoc network, for instance, might include a variety of sub-models to account for radio propagation, node mobility patterns, the composition of the protocol stack software in each node, and the packet traffic generation. Each of these sub-models will contain *factors* to represent configurable parameters. Models need to be validated, that is, one must make sure that they correctly represent the systems for which they stand in. The computational implementation of the models need to be verified, that is, debugged.

The choice of a simulator might determine the extent to which the experimenter can directly use pre-constructed, validated, and verified models. Commonly, one chooses a simulator that offers a selection of the largest number of re-

quired models. Assuming that those models have been validated and verified, they can be relied upon without modification. Otherwise, any newly constructed models must undergo the usual scrutiny of validation, verification, and debugging. This requires executing pilot runs and fine-tuning the factors in the chosen models to ensure that the desired scenario indeed matches the specification of the simulation experiment.

Up to the design of experiments stage, there are limited opportunities for using automation tools in the simulation process. Our experience has shown, however, that from this point on, automation tools can be used to guide the user through the well-established methodology and to enhance the credibility of the results.

Some of the most important considerations in the simulation setup appear early in the design of experiments. One must determine whether the type of the simulation is *terminating* or *steady-state*. One must determine the length of each simulation run, that is, end time of the simulated clock. For steady-state simulations, one must determine the length of the warm-up period for the model so that initialization biases can be avoided through data deletion. As indicated in [1], a large number of simulation studies of computer networks have relied on arbitrary choices for the length of the simulation and altogether failed to consider initialization biases. These practices, which compromise the credibility of the entire study, can be avoided by automation. The sequential simulation techniques discussed in [8] make it possible for the simulator to recognize the end of warm-up periods and to extend the simulation run until enough samples of metrics have been collected to guarantee the coverage of prespecified confidence intervals. Finally, in order to produce valid statistical analyses for the simulation experiment, at this stage, one must specify the pseudo-random generator seeds for each simulation run.

Simulation set up also includes the development and the selection of scenarios that agree with the goals of the study. Scenario development is a complex topic that deserves careful attention, but one which cannot be automated and therefore falls beyond the scope of our discussion. We refer the interested reader to [9] for a thorough exploration of this topic. The choice of scenario determines the *levels* for all factors, or parameter settings, in the simulation model. A very large number of factors commonly appears in the simulation of wireless ad hoc networks. The experimenter must take care of keeping detailed records of these values and make them available to anyone interested in reproducing the study. The completeness and the integrity of this body of data determine whether the experiment can ever be reproduced. The simulator or an additional support tool can better take care of this bookkeeping and reporting operation than the experimenter.

The design space of the simulation study is determined by the number of possible combinations of parameter settings that need to be investigated. When the study requires simulations with multiple levels for multiple factors, as is the case in sensitivity analysis, a combinatorial explosion may require a very large number of simulation runs. When this is the case, the use of 2^k factorial or 2^{k-p} fractional factorial experimental designs can produce a substantial reduction of the number of runs that is required to meet the goals of the study [10].

Another way to reduce the total execution time of simulation runs is to exploit parallelism, which can be achieved through different approaches. The *parallel discrete-event simulation (PDES)* approach partitions the simulation model into a collection of sub-components. These sub-components execute on multiple processors under the coordination of a mechanism that guarantees that the simulation advances to a consistent global state. PDES has been the focus of much research in the last 30 to 40 years and, although great advances have been made, it continues to be a complex subject. Among the several network simulators that use this approach to parallelism, we can cite SSFNet [11], PRIME [12] and GTNetS [13]. The *Multiple Replications in Parallel (MRIP)* approach offers a simpler alternative by executing different replications of sequential simulations in different processors. Each independent simulation run can explore a different point of the experimental design space. It is also possible to have multiple runs of the same point of the experimental design space execute with different seeds for pseudo-random number generators (PRNGs); in this case the combined effort of the parallel replications reduce the time to produce statistical estimates of the metrics of interest. The MRIP approach is used in support tools for network simulation such as Akaroa 2 [14] and SWAN Tools [15]. All the network simulation tools cited above automate parallelization to some extent.

The next step of the simulation workflow includes extracting the metrics of interest from the output of simulation runs, processing the extracted data according to rigorous statistical methodology, and storing them for later reference. In the execution of these tasks, one must pay careful attention to detail so as to avoid procedural errors that can compromise the validity of the simulation's results. Fortunately, since they follow repetitive patterns dictated by well-established methodology, they can be easily automated.

The automation framework for processing and storing simulation data can have a significant impact in the documentation and in the reporting of the results of a study. As we discuss later, as long as the framework is constructed around a relational database, it can unequivocally associate the output data of a study with its previously recorded experimental setup. This can guarantee that, when the time comes for the dissemination of the results, the information reported is complete and accurate. Another important consequence is the database can serve as a central repository of information for anyone interested in pursuing further analysis of the experiment or in reproducing it.

Having presented the case for increasing the level of automation in the simulation workflow, in the next section, we evaluate the existing resources and the challenges for this task in three different situations. The case studies we discuss include modern network simulators, such as *ns-2*, *ns-3*, and those based on the Scalable Simulation Framework (SSF) standard.

3. REFLECTIONS ON THE STATUS QUO

The number of different simulators for wireless ad hoc networks is not small and continues to grow. Although it is hard to speculate on the reasons that connect individual users with particular simulators, it is fair to expect that they include attraction to particular feature sets.

More often than not, the purpose of a simulation study is more in line with the study of characteristics of the system than with the development of the simulation tool. For that

reason, it is natural for the user to want to remain a *user* by selecting an existing simulator that can serve the goals of the study without requiring extensive modifications. When the choice is made, the user bites the proverbial bullet to accept the simulator features together with its shortcomings. Regrettably, not enough regard has been given to how the construction of network simulation tools might affect the outcomes of study produced with them.

The credibility of a simulator depends in great part on how closely the models in its libraries relate to the realities of the physical system as well as to prescribed standards, as is commonly the case for network protocol models. The credibility of a simulation study, however, depends on how closely the process follows the best practices described in Section 2. Looking at network simulation tools from a usability perspective, however, one can explore the relation between the credibility of the simulation tools and of the simulation studies produced with them.

It can be argued that the majority of open source network simulators has a target audience that is comprised of power users. There seems to be an expectation that people who engage in the *simulation* of computer systems are skilled in the *use* of computer systems. It is due to this expectation that, at times, network simulators have steep learning curves even for less ambitious users. The level of difficulty in working with a network simulator rises fast and in direct proportion to the level of customization it requires in a study. Consequently, the potential for the introduction of errors in simulation methodology is tightly coupled with customization.

Ideally, network simulators would be like bicycles for children. They could provide mechanisms analogous to training wheels to protect the less experienced users from their own mistakes. Either the simulator or some support tool could optionally constrain the user to follow the methodology while, at the same time, minimizing the requirements of knowledge of programming languages and other system utilities. Unfortunately, the most popular network simulators don't include many protections for the experimenter.

We argue that there exists a potentially large number of ways in which one can make mistakes in setting up the scenario for a simulation study or in processing output data. The extent to which these errors might affect the goals of the study is unpredictable, but troubling. There can be a substantial difference between what the user intends to do and what the user actually ends up doing. When that is the case, all the resources spent in the execution of the simulation study will have been wasted.

Experience has shown that scenario construction is a hard problem on its own [16]. No tool can possibly prevent the simulation user from constructing an inconsistent, implausible scenario. It is possible, however, to construct tools that help the user avoid introducing errors in simulation workflow that would result from mishandling scenario configuration. For instance, Graphical User Interfaces (GUIs) can be implemented as an additional layer on top of the simulator's existing configuration functionality. This type of interface can be very effective in bridging the distance between a user's mental model of the configuration and the text files that describe it for the simulator. Another effective use of the GUI is to expose to the user a reduced number of configurable parameters for the simulation model. The raised level of abstraction not only hides complexity, but can be used to prevent less experienced users from accessing parameters that could render the simulation meaningless if misconfigured. The GUI can also be tasked to perform validation of the user input, serving as a first line of defense against errors of type and range. Visual configuration tools are available in commercial simulators such as OPNET [6] and QualNet [5], but also in open source tools, such as OMNeT++ [17] and NCTUns [18]. Just as high-level languages raise the level of abstraction for programming and create a safer environment for the developer in comparison with assembly, we should consider tools that produce analogous benefit in simulation.

In order to promote an understanding of the current state of development in the interfaces between experimenter and network simulator, in the remainder of this section, we explore three different cases. This discussion presents evidence of the need for support tools to protect users from making egregious mistakes in the configuration of simulation scenarios and in output data processing.

3.1 Case study: ns-2

The Network Simulator version 2 (*ns-2*) [3] is widely used in the networking community to perform a broad variety of simulation studies, ranging from the evaluation of Medium Access Control (MAC) protocols in personal area networks to Internet transport protocols. A development community has supported *ns-2* throughout the years by continually contributing with a large number of simulation models. A considerable number of contributed models eventually became part of the simulator distribution. Although the *ns-2* core and its modules are released under mixed licences, the users have open access to source code and face no limitations of use. *ns-2* is written in a mix of C++ and OTcl, the object-oriented extension of the Tcl scripting language, in a design that implements the concept of *split-level programming* [19]. The configuration of simulation experiments uses an interpreted language, which means that the simulator doesn't have to be recompiled when new model structures are defined.

A negative consequence of *ns-2*'s growth was that core's features received contributions from uncoordinated, independent research groups over time. Since no refactoring took place to impose a broad-scope design vision, additional development often had to resort to work-around solutions to make different components fit together. To illustrate the point, we look at how *ns-2* supports the simulation of wireless networks. The core structure of the simulator was originally created around the idea of a generic model for network node for which different protocols layers could be defined. This didn't turn out to be possible and a second kind of node model had to be introduced to allow for nodes constructed with models of wireless physical, MAC, and link layers. We can still observe the use of this pattern of patching the design to solve individual problems rather than rethinking the global structure of the simulator to provide a more general framework in [20, 21].

From the perspective of its users, *ns-2* poses several challenges. The documentation is sparse, particularly for most contributed modules, and often out of date. The user is expected to have at least basic knowledge of Tcl, a language of decreasing popularity. Worse yet, the user is often required to understand internals of the simulator such as how the event scheduler and classifier work, which contradicts the split-level programming model. There are few conven-

tions in the code in what regards the naming of variables and the use of units of measure. Managing the installation of contributed models requires the applications of patches, which is not always automated. The combination of these factors makes for a steep learning curve and long setup time. Novice users have to undergo a significant amount of studies and spend considerable time in trial-and-error experimentation. While this effort can be justified for larger research projects, it discourages the use of *ns-2* in education.

Another uninviting characteristic of *ns-2* is the complexity of scenario configuration. It is not always clear for the user where to find the parameters of interest in each component of the simulation model. At times, the parameters appear in Tcl files. In others cases, they might be defined as literals in C++ header files and changes in parameter settings require one to rebuild the simulator executable. (This showcases another violation of the goals of the split-level programming model.)

In configuring an experimental scenario, one must also consider the fact that the simulator works with a large number of default values for parameters, defined in file `tcl/lib/ns-default.tcl`. The contents of this file, which spans more than 1,000 lines of Tcl code, can change with different versions of the simulator. Comparing the results of the simulation of one same scenario executed on different versions of the simulator, we might observe significant changes. As discussed in [1], this can impact the reproducibility of the experiment. It should also be noted, that due to the overwhelming volume of configurable parameters, the user shouldn't be held responsible for configuring them all without assistance.

There is no provision in *ns-2* to expose to the users only the subset of parameters in a simulation model that is pertinent to the goals of their simulation study. One must wade through `tcl/lib/ns-default.tcl` and determine what is safe to change and what ought not to be touched. As a further complication, there is no indication of the acceptable ranges for each parameter in this file.

An interesting idea that can be used as a solution to the scenario configuration problem is presented in [22]. While the paper is focused on the development of a description language for web-based network simulation, its main contribution is portable to a broader context. It would be beneficial to use a hierarchy of documents in the Extensible Markup Language (XML) to encapsulate the entire set of parameter values used in the simulator and in the components of its library of models. The documents could include annotations that explain the function of each parameter and one could build XML Schemas to validate the specific values with respect to range and type.

A standard language for scenario description would enable a number of additional improvements to the simulation workflow. We can envision that the results of scenario development efforts, such as what is described in [9], would lead to the creation of a repository of valid scenarios spanning a variety of different purposes in network simulation. If each scenario is described in the standard language, one could use XSLT transformation documents [23] to translate the scenario into configuration files to drive particular simulators. In [22], we see a proof of concept in the translation of an XML language to *ns-2* configuration scripts.

In what regards tools for the collection of metrics during the simulation and their statistical analysis and visualization, the *ns-2* distribution doesn't offer very sophisticated solutions. The simulator produces logs with *packet traces* as ASCII text. The logs contain samples of metrics for all the packets that traverse *ns-2* elements like agents, queues, and channels. A large volume of such data is generated even by small simulation runs and requires non-negligible amounts of storage.

Most often, users rely on ad hoc solutions to parse and to process the log data. Home baked solutions might be of help, once they have been fully developed and debugged, but it's a time consuming process. While the use of graphical tools helps, they add to the overhead of learning to use yet another piece of software. Upon recognizing this deficiency, the community has started to develop tools to automate parsing, processing, and visualization. iNSpect [24] addresses the visualization of data generated by the simulation of wireless networks in *ns-2*.

It is also important to note that not all metrics of interest in a simulation experiment might appear on packet traces because they are not related to transmission events. For instance, the designer of a new ad hoc routing protocol might find it useful to periodically collect metrics like the number of timer expirations or the size of routing tables. It would be helpful to have mechanisms to allow for the collection and the analysis of this type of data; we present one such solution in Section 4.1. It would be important, as well, to produce output data that contains annotations that facilitate the post-processing and the dissemination of results. CostGlue [25] is good step in this direction.

3.2 Case study: ns-3

Although the Network Simulator 3 (*ns-3*) [26] might be viewed as a descendant of *ns-2*, it does not inherit from *ns-2*'s code base. It was constructed from scratch following a new design that takes into account lessons learned in the development of its predecessor, one that allows for the smooth integration of modules and expansions. *ns-3* provides new features and capabilities that enhance the quality and the relevance of simulated results to the study of real world scenarios. In addition to providing the developers with a cleaner programming framework, the modular design of *ns-3* helps the users validating the effectiveness and accuracy of simulations. The code base is growing fast and there have been several releases to support additional development and public testing.

The build script for *ns-3* uses Waf, an open source build system with functionality related to Gnu Autotools and written in Python, a modern, well-known, and powerful scripting language. Waf is central to *ns-3* in that it is used to not only build the simulator, but also run simulations. Waf ensures that all libraries are properly linked in, and sets up the runtime environment for the simulator. This ensures that simulations will be executed as the user intended for them to be, thereby reducing the possibility of human error, e.g. in managing the compilation dependencies causing partially updated executables, hence inconsistent simulation runs.

Similarly to *ns-2*, the interface that *ns-3* offers to the user follows an interpretation of the split-level programming model. There is a clear separation between the simulator code base and the scripts the user creates to stage simulation experiments. The simulator can be built to support Python bindings, which allows one the option to use either C++ or

Python in the creation of experiment descriptions.

Using the scripts that describe the simulation model, the Waf build system can determine the set of classes that must be linked with the simulator code to support the needs of the experiment. An interesting feature that results from *ns-3*'s implementation based on Waf and Python is the interactive shell that can be used for running preliminary tests with the simulator [27]. This feature will be useful in verifing assumptions about configuration scripts before the launching of production runs of the simulator.

The core architecture of *ns-3* has been changed with many design goals in mind. The first one, arguably one of the most important, is to make it more user friendly to developers. Another one is to capitalize on the recent advances in multi-core processors. A third one is to allow for emulation and virtualization of physical devices. This allows the experimenter to reuse kernel and application code in the simulator, thereby reducing the possibility of errors in defining models to simulate real applications or kernel modules [26].

While simulations results often provide much insight, one must be able to compare them to the behavior of real systems. For this reason, *ns-3* has been designed to allow simulations to be integrated with running networks. Virtual machines can be run on top of *ns-3*'s simulated devices or channels. Alternatively, *ns-3* emulated devices can push packets around a true physical network to measure real time performance. This allows the experimenter to simulate real applications.

In any network simulation there are two major points of configuration. First, one must define how the nodes in the network are interconnected, that is, specify a network *topology*. Second, one must also define the values of parameters in the configuration of each network device. The topology of the network is defined the simulation script, but the values assigned to the nodes are controlled by the *ns-3* attribute system. This system allows for output of simulated values, both user specified and default, thereby making simulations easily repeatable. The user can also define many of these values from the command line with Waf to experiment with things without changing source. Providing the user with a pre-configured safe and known set of values for the system parameters, upon which she has full control via C++/Python and command line, reduce the likelihood of misconfiguring the simulator and makes it easier to reproduce simulation studies. We then consider this feature as one of the main strengths of *ns-3* regarding the credibility and repeatability of results.

An interesting experimental feature of *ns-3* may turn out to enhance considerably the usability of this simulator: `ConfigStore` can use a GTK-based front end that gives the user a GUI for viewing and changing attributes of the network nodes before a simulation experiment is executed. Although, at this point in time, the GUI does not allow the user to visually define the topology of the network, it is conceivable that it may include this functionality in the future.

As far as parsing and understanding the simulation output are concerned, *ns-3* provides a logging module that can report on the execution of the simulation according to configurable levels of detail. Logging can be enabled or disabled on a per component basis. Such output is valuable for debugging purposes and also for understanding the results. For debugging purposes, ASCII *traces* similar to those of *ns-2* can be generated to fully document the execution of the simulation. Furthermore, *ns-3* can output packet capture traces in `libpcap` format, which can be analyzed with familiar software packages such as Wireshark and `tcpdump`. This feature gives the user access to a standard interface that can be effective in the analysis of trace files.

Finally, it is interesting to note that the use of Python in *ns-3* can have a positive impact in the creation of automation tools *ns-3*. Python is widely used in web applications, whether through the Common Gateway Interface (CGI) or through web development frameworks such as Django `<djangoproject.com>` or TurboGears `<turbogears.org>`. This characteristic will enable one to create for *ns-3* an interface for simulation configuration, execution, and analysis that provides less-experienced users with a safer environment for the development of credible studies similar to that described in [15].

We expect that in due time, support tools and libraries that have recently been developed for *ns-2* will exist also to support the users of *ns-3*.

3.3 Case study: SSF Simulators

The *Scalable Simulation Framework* (SSF) is an object-oriented standard for the construction of high performance simulators [28]. The standard is powerful but concise, offering five base classes for the definition of events, processes, entities, and communication channels. Different simulator *kernels* have been constructed using the SSF application programming interface (API): DaSSF [11], iSSF [29], MaSSF [30], and PRIME [12]. Primarily, these kernels take care of the mechanics of discrete-event simulation with an emphasis on high performance. When the simulation is executed over multiple processors, the kernels also handle the distribution and the synchronization of components of the simulation model.

Network simulators based on the SSF standard consist of collections of *components* written in a high-level language, such as C++ or Java. These components are developed from the classes offered by the kernel and create models of protocols, network nodes, and other entities in the simulated environment. Two examples of such simulators are SSFNet [11] and the Simulator for Wireless Ad Hoc Networks (SWAN) [31].

A common characteristic of SSF-based simulators is that experimenters build simulation models using the Domain Modeling Language (DML) [32]. This language allows one to instantiate and to configure each of the models that were previously constructed using a high-level language. The use of DML reflects the same philosophy of split-level programming used in *ns-2*. However, SSF-based simulators have managed to achieve a clearer distinction between the implementation of simulation models and their configuration. Protocol models are completely contained in compiled code, while their parameters are assigned levels in DML configuration. The substantial difference here arises from the fact that *DML is not a programming or scripting language*. Two important practical consequences arise from this design. First, it protects the integrity of the simulator's code base by not allowing an unsophisticated user to compromise its correctness by carelessness or lack of skill. Second, the design enables one to construct automation tools that help the user to create configuration files under constraints that ensure an added level of credibility to the simulation experiments.

The DML syntax is at the same time simple and power-

ful. DML contains mechanisms for the hierarchical structuring of model descriptions and for the reuse of fragments of description code. DML documents can have their syntax validated automatically by schemas written in DML itself, much in the same way that XML Schemas can be used to validate the syntax of XML objects. SSFNet [11] makes use of this feature to associate units with numbers that appear in a DML model configuration. When a *data rate* parameter must be assigned a value, a DML schema may require that the value is followed by its unit of measure, as in `5.5Mbps`, for instance. The schema checker will verify whether the units used are appropriate (syntax) and the simulator will parse the value into the appropriate numeric value (semantics) to be used in the experiment. Mechanisms such as these are of paramount importance to the creation of reliable simulation experiments, as they protect the user from inadvertently introducing errors in the configuration of models. Syntax validation schemas, however, do not address an important issue related to component-based simulators. As observed in [15], any simulator that allows users to build a model by picking and choosing from a library of previously constructed components is open to the possibility of mismatches.

As the component library of component-based simulators grows, it becomes important to provide a mechanism that can verify the *compatibility* and the *interdependence* of the components selected in the construction of a simulation model. An example given in [15] illustrates the case of incompatibility between components: one might create a model of wireless node by accidentally mismatching PHY and MAC layers from two different variants of the IEEE 802.11 standard. We offer a second example to illustrate the interdependence of models. We learn from [33] that when the random waypoint model is used to describe wireless node mobility, it is beneficial to sample the initial position of nodes space from a triangular random variate. This pairing tends to reduce the simulation time until the distribution of nodes in space converges to a state compatible with the distribution that emerges from the random waypoint model. Based on this knowledge, one might *require* that whenever the random waypoint model is selected for mobility the initial node deployment follows a triangular distribution.

Problems of compatibility and interdependence of components of a model can be detected with the use of an additional tool for *consistency checking*. The task goes beyond what can be accomplished by schemas and schema checkers. Assuming that every model created for the component library would have an associated description file, one could build a consistency checker that works similarly to software installation and update systems for Linux, like Debian's Advanced Packaging Tool (`apt-get`). These tools have been effective to manage packages in operating systems installations and guarantee that the dependencies and requirements of tools are always satisfied. In the simulation domain, such a tool would produce significant advances in ascertaining the correctness of models.

At the same time that we recognize the strengths of DML, our experience with its use in SWAN has exposed a significant problem of usability. Even though the DML grammar is simple, it leads less experienced users, particularly, undergraduate students, to create configuration files that don't violate syntax but which cause parsing errors at run time. Since the kernel of the simulator provides terse messages for this kind of error, the user had difficulties debugging model definition files. This observation indicated the need for the creation of a different mechanism for the creation of DML files, which we discuss further in Section 4.2.

4. SIMULATION SUPPORT

Having gone through an objective analysis in the strengths and the deficiencies of existing network simulators in Section 3, this section focuses on three tools that support the simulation workflow. What the cases we discuss here have in common is the general goal of creating means to ease the burden on the user. Each in their own way, the cases we discuss lead to simulation results that are more credible and to experiments that are repeatable by third-parties.

4.1 ns-2 Extensions

The *ns2measure* module [34], released as a patch for *ns-2*, addresses two problems described in Section 3.1: the collection of samples of metrics and the statistical analysis of packet traces. The solution to the data collection problem provides a general mechanism for specifying which events are to be logged. This allows one to record data about any type of event rather than just data related to packet transmission events. The recording of data avoids frequent I/O so as not to slow down the simulation. The format used in the data log entries simplifies the extraction of information for posterior analysis and the generation of graphs. The package also includes tools that free the user from having to create their scripts for staging simulations experiments and analyzing output data.

The data collection subsystem is based on the implementation of a C++ class called `Stat`, which processes and organizes samples from an arbitrary number of different metrics. When the user instruments the *ns-2* C++ code with calls to a `Stat::put()` method, samples of a metric are passed to a `Stat` object. The samples are processed into a different histogram for each metric and only the final outcome is written to file, which avoids the frequent I/O that would ensue from the construction of raw packet traces. The `Stat` class provides support for three types of data: metrics averaged over time (e.g. throughput or loss rate), metrics reflecting stochastic values over continuous-time (e.g. number of packets in a queue), and metrics reflecting stochastic values over discrete-time (e.g. end-to-end delay for a flow of packets).

The software framework for automating statistical analysis allows a user to execute a number of independent replications of the same simulation scenario and to compute means and confidence intervals on the chosen metrics. This framework relieves the *ns-2* user from having to write code in the Tcl scenarios for selecting independent substreams of random numbers and for computing confidence intervals. The framework also allows one to have simulation objectives guide the actual running of replications. For example, it is possible to specify that a new independent replication of a given scenario should be generated and executed until the confidence interval of the required measure is below 10 percent of the sample mean. This way, a user does not need to "guess" the right number of replications that satisfies the stated goal.

The framework consists of two programs: the *analyzer* and the *loader*. The *analyzer* takes as an input a configuration file, in which the user defines a minimum/maximum number of runs, and a set of relevant measures, together

with their required confidence intervals. At the end of each replication, the *analyzer* computes confidence intervals, as specified in its configuration, using the samples collected and stored into a file by the `Stat` object. If the confidence intervals are small enough for all the relevant measures or the maximum number of runs is reached, *analyzer* saves the output measures and terminates. Otherwise, it instructs the *loader* program to run a new replication. In doing this, *loader* automatically initializes the *ns-2* Random Number Generators (RNGs) so that the new replica is independent from already completed replications.

Both the patch and the utilities are distributed under the GNU Public Licence (GPL). Users who are less proficient with *ns-2* reap important benefits from *ns2measure*: they are allowed to concentrate on the goals of simulation study rather than struggle to use the simulator and to deal with data collection and analysis.

Finally, we note that *ns-2* lacks resources to create a permanent storage for the results of simulation experiments and to facilitate their dissemination. Such a feature is especially important to enable collaborating researchers to share simulation results and to overcome the restrictions of space related to their publication. Functionality to support these goals can be found in CostGlue [25] and in SWAN Tools [15], which we discuss next in Section4.2. Much of the functionality that is required to meet these goals is centered on the use of a database management system. Currently, *ns2measure* is being extended to save simulation results in a database that can be accessed from a web interface.

4.2 SWAN Tools

SWAN Tools [15] is a web-based framework for the automation of the entire simulation workflow with SWAN [31]. The framework was developed as a Ruby on Rails application and it addresses issues that can put in jeopardy that credibility of simulation studies. The benefits that SWAN Tools provides for the more experienced user come from guidance in the configuration of models, in the management of multiple and potentially large simulation experiments, in the application of sound statistical methodology for output analysis, and in the organization and presentation of results. We note that these functionalities also have a strong impact in the usability of SWAN by less experienced users, undergraduate students, in particular.

The simulation workflow is represented in the architecture of SWAN Tools as five different components. The *Experiment Configuration* component sets the stage for the simulation study by requiring the user to define data that pertains to all its simulation runs. From a single integer seed, it generates the multiple seed values that are required to initialize the PRNGs in the simulation model. It requires the user to match the experiment with the code base of the simulator in order to guarantee the reproducibility of the experiment. When the appropriate version of the simulator is not known to the framework, this component requires the user to upload it as a compressed archive.

The *Model Specification* component guides the user through the definition of all parameters that can be configured in the model built for the simulation study. It presents the user with an information-rich interface that helps the less experienced user understand what the parameters role in the model are. From the parameter values entered by the user, the component generates syntactically correct DML specification files. The current implementation of this component works with a pre-defined composition of the wireless nodes' protocol stack; future extensions should allow the user to pick and choose from a library of protocol models.

The *Simulations* component generates all the design points for an experiment and dispatches individual runs of each simulation to different processors, when executing on a cluster or network of computers. This component parallelizes the execution of the simulation study according to the MRIP approach. As simulation output data is generated, it is returned to the system, which archives them in the database. The *Results* component provides access to this database via web browsers. The *Plotter* component allows one to use another web browser interface to request the construction of graphs, which include confidence intervals for each data point, by default. The graphs are generated by the system's backend in a format specified by the user.

Although SWAN Tools is restricted to work with one specific underlying simulator, it automates much of the simulation workflow and enhances the credibility of networks simulation studies. In its current state, the tool does not address two extremely important issues, which are covered by functionality of the tool that we discuss next.

4.3 Akaroa 2

The survey presented in [1] gives evidence that two very important issues have been neglected in published studies of network simulation. Both are intimately related to the statistical rigor of the simulation study and determine the degree to which its results are credible, as discussed in [35].

The first issue is the length of the simulation run, that is, the end time of the simulated clock. This value determines how many samples of the metrics of interest will be collected in the simulation, and thus it is directly related to the confidence intervals of these metrics. Once a user specifies a confidence level, the number of samples collected determines whether the estimated mean is contained in the confidence interval and also determines the width of the confidence interval. The second issue is whether or not the metrics are sampled after the warm up period of the simulation models so as to avoid initialization biases.

The *Akaroa 2* simulation package [36] uses automation to address both these issues. The user specifies the confidence level for the metrics estimated and the system determines how long simulation runs need to execute in order to meet that goal. It also collects samples of metrics using data deletion to ignore those obtained before steady-state was reached. Once enough samples have been accumulated to guarantee the coverage of their confidence intervals, *Akaroa 2* stops the simulation runs. To enable faster progress, *Akaroa 2* makes use the MRIP approach to distribute simulation runs to various processors and communicates with each simulation to collect and organize their output data. *Akaroa 2* can be adapted to work with any simulator that accepts C++ linkage and the required interface code already exists for *ns-2* and OMNeT++

Without automation, the experimenter has to resort to trial runs to evaluate when the model enters steady-state. Also, it's up to the user to determine if the length of the simulation is sufficient to meet the stated level of confidence. In spite of these strengths, *Akaroa 2* does not automate other steps in the simulation workflow.

5. LESSONS LEARNED

Our experience in the development of support tools for network simulation and the analysis of the literature reported in this paper have yielded important conclusions. In this section, we give a summary of key points which can advance the usability and the credibility of network simulators.

From the observations we made in this paper, we conclude that various tools have to a certain degree provided solutions to specific problems in the simulation workflow. Although, we didn't find the entire set of solutions in one same tool, it seems clear that they exist as components that could be aggregated into a simulation system that produces credible results for beginners and experienced users alike.

We identified that opportunities for automating the process appear from the start to the end of the simulation workflow. Building the simulation model from sub-components is an activity that can be aided by a model consistency checker that embodies the knowledge developed in scenario development research. This consistency checker would rely on data stating the compatibility constraints of each model sub-component. We learned of efforts in the development of standard languages for model description that can be translated into configuration files for different network simulators. The use of such languages would allow for published experiments to be reproduced by third-parties.

Once a model has been built, validated, and verified, it needs to be configured with an exhaustive list of parameters values that reflect the goals of the experiment. It is important to hide the overwhelming number of configurable parameters behind an abstraction for the sake of the novice user, who should see only the parameters that are of interest for a chosen type of study. At the same time, it is important to make it possible for experts to have full access to model configuration options. Even these constituents, however, will benefit from a clear, clean, and safe interface. Model parameters would be better placed outside the simulator, in a configuration file that can contain annotations on the nature and the range of each parameter. Hard-coding parameters in the simulator compromises both usability (by requiring recompilation) and credibility (by making results depend on simulator version). The split-level programming model, when carried out to its full extent, can offer assurances that in the configuration of a model, the user does not introduce errors in the simulator.

It's important for the tool to provide guidance to the user with experiment design. The simulation system can ask the user for the parameter values to experiment with and use a well-established design method to generate the specifications for all the simulation runs that will be required to complete the study. The tool can automatically provide seeds to the PRNGs in each replication of a stochastic simulation run and distribute them across different processors using the MRIP approach. Alternatively, there exist resources today to automate also the partitioning of large models for parallel execution. We have shown also that current simulation support tools can have a strong impact on the quality of the results produced by determining on their own the length of transients in steady-state simulation and the length of distributed simulation runs. These tools guarantee the statistical quality of estimates produced through simulation.

In what regards simulation output data, we have seen developments on the use of database management systems to enhance the organization and the permanence of results. When the results of a simulation run are processed and inserted into a database, one can link them to the associated simulator configuration files. This provides guarantees that the experiment can always be reproduced and that its results will be available for comparison. We have seen the importance of providing the user with tools for collecting and processing simulation output and also for annotating data with information that helps its posterior analysis. The automation of these activities not only relieves the user from having to create their own processing code, but also guarantees the statistical correctness of the processed results.

The combination of all these features into a single system to support network simulation would be invaluable and could have a big impact on the quality of the science produced.

With a layered design, an automated simulation system could provide more guidance for users who would benefit from it and, at the same time, offer different interfaces for users who have a high level of expertise. In either case, we believe that by placing knowledge of simulation best practices *into the system*, one would enable the simulation tool to produce a significant impact on the quality and the credibility of published simulation studies. In closing, we note that the resulting tools would be just as useful for classroom use as they would be for research.

6. REFERENCES

[1] T. Camp, S. Kurkowski, and M. Colagrosso, "MANET simulation studies: the incredibles," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 9, no. 4, pp. 50–61, 2005.

[2] K. Pawlikowski, H.-D. J. Jeong, and J.-S. R. Lee, "On credibility of simulation studies of telecommunication networks," *IEEE Communications Magazine*, vol. 40, January 2002.

[3] "ns2, version 2.33," available at <www.isi.edu/nsnam>, [Accessed Oct. 10, 2008].

[4] "GloMoSim global mobile systems simulation library, version 2.0," available at <pcl.cs.ucla.edu/projects/glomosim>, [Accessed Oct. 10, 2008].

[5] Scalable Networks Technologies Inc., "Qualnet, version 4.5," available at <www.scalable-networks.com> [Accessed Oct. 10, 2008].

[6] Opnet Technologies Inc., "Opnet modeler," available at <www.opnet.com> [Accessed Oct. 10, 2008].

[7] A. M. Law, *Simulation Modeling and Analysis*, 4th ed. McGraw-Hill, 2007.

[8] D. McNickle, G. Ewing, and K. Pawlikowski, "Transient deletion and the quality of sequential steady-state simulation," in *Proceedings of the 21st European Conference on Modelling and Simulation*, Prague, Czech Republic, June 2007.

[9] S. Kurkowski, T. Camp, and W. Navidi, "Constructing MANET simulation scenarios that meet standards," in *Proceedings of the 4th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2007)*, 2007, pp. 1–9.

[10] S. M. Sanchez, "Work smarter, not harder: Guidelines for designing simulation experiments," in *Proceedings of the 2007 Winter Simulation Conference*, Washington, D.C., U.S.A., December 2007, pp. 84–94.

[11] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski, "Towards realistic million-node internet simulations," in *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, Las Vegas, NV, U.S.A., June 1999.

[12] J. Liu, "Immersive real-time large-scale network simulation: a research summary," in *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS'08) NSF NGS Workshop*, Miami, FL, U.S.A., April 2008.

[13] G. Riley, "The Georgia Tech network simulator," in *Proceedings of the ACM SIGCOMM Workshop on Models, and Tools for Reproducible Network Research (MoMeTools '03)*, New York, NY, U.S.A., 2003, pp. 5–12.

[14] K. Pawlikowski, "Akaroa2: Exploiting network computing by distributing stochastic simulation," in *Proc. of the 1999 European Simulation Multiconference*, Warsaw, Poland, 1999, pp. 175–181.

[15] L. F. Perrone, C. J. Kenna, and B. C. Ward, "Enhancing the credibility of wireless network simulations with experiment automation," in *Proceedings of the IEEE International Workshop on Selected Topics in Mobile and Wireless Computing (STWiMob 2008)*, Avignon, France, October 2008, pp. 631–637.

[16] S. Kurkowski, W. Navidi, and T. Camp, "Two standards for rigorous MANET routing protocol evaluation," in *Proceedings of the 3rd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2006)*, 2006, pp. 256–266.

[17] A. Varga, "The OMNeT++ discrete-event simulation system," in *Proceedings of the European Simulation Multiconference*, Prague, Czech Republic, June 2001, pp. 319–324.

[18] S. Wang, C. Chou, C. Huang, C. Hwang, Z. Yang, C. Chiou, and C. Lin, "The design and implementation of the NCTUns 1.0 network simulator," *Computer Networks (Elsevier)*, vol. 42, no. 2, pp. 175–197, June 2003.

[19] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Hemy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. You, "Advances in network simulation," *Computer*, vol. 33, no. 5, pp. 59–67, May 2000.

[20] N. Baldo, F. Maguolo, M. Miozzo, M. Rossi, and M. Zorzi, "ns2-MIRACLE: modular framework for multi-technology and cross-layer support in network simulator 2," in *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, 2007, pp. 1–8.

[21] L. Paquereau and B. E. Helvik, "A module-based wireless node for ns-2," in *Proceedings from the 2006 workshop on ns-2: the IP network simulator (WNS2 '06)*. ACM, 2006, p. 4.

[22] R. Canonico, D. Emma, and G. Ventre, "An XML description language for web-based network simulation," in *Proceedings of the IEEE Symposium on Distributed Simulation and Real-Time Applications (DS-RT'03)*, October 2003, pp. 76–81.

[23] W3C Consortium, "XSL Transformations Version 2.0," Available at <www.w3.org/TR/2007/REC-xslt20-20070123/> [Accessed October 10, 2008], January 2007.

[24] S. Kurkowski, T. Camp, N. Mushell, and M. Colagrosso, "A visualization and analysis tool for ns-2 wireless simulations: iNSpect," in *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, Atlanta, GA, U.S.A., September 2005, pp. 503–506.

[25] D. Savić, M. Pustisek, and F. Potortì, "A tool for packaging and exchanging simulation results," in *Proceedings of the First International Conference on Performance Evaluation Methodologies and Tools (Valuetools '06)*, Pisa, Italy, October 2006.

[26] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, "ns-3 project goals," in *Proceedings from the 2006 workshop on ns-2: the IP network simulator (WNS2 '06)*, 2006.

[27] "ns-3 reference manual," available at <www.nsnam.org/docs/manual.pdf>, [Accessed Oct. 10, 2008].

[28] J. Cowie, "Scalable simulator framework reference manual," 1999, available at <www.ssfnet.org> [Accessed March 22, 2008].

[29] D. M. Nicol, J. Liu, M. Liljenstam, and G. Yan, "Simulation of large-scale networks using SSF," in *Proceedings of the 2003 Winter Simulation Conference*, New Orleans, LA, U.S.A, 2003, pp. 650–657.

[30] X. Liu, H. Xia, and A. A. Chien, "Network emulation tools for modeling grid behavior," in *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.

[31] J. Liu, L. F. Perrone, D. M. Nicol, C. Elliott, and D. Pearson, "Simulation modeling of large-scale ad-hoc sensor networks," in *Proceedings of the European Simulation Interoperability Workshop 2001 (EURO SIW 2001)*, University of Westminter, London, UK, June 2001.

[32] J. Cowie, "Domain modeling language reference manual," available at <www.ssfnet.org/SSFdocs/dmlReference.html> [Accessed June 8, 2008].

[33] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002.

[34] C. Cicconetti, E. Mingozzi, and G. Stea, "An integrated framework for enabling effective data collection and statistical analysis with ns-2," in *Proceedings of the 2006 workshop on ns-2: the IP network simulator (WNS2 '06)*, Pisa, Italy, October 2006.

[35] K. Pawlikowski, "Steady-state simulation of queueing processes: a survey of problems and solutions," *ACM Computing Surveys*, vol. 22, no. 2, pp. 123–170, 1990.

[36] K. Pawlikowski, D. McNickle, and G. Ewing, "Project Akaroa," available at <www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa> [Accessed June 8, 2008].