A Framework for Large-scale Simulations and Output Result Analysis with ns-2

Matteo Maria Andreozzi, Giovanni Stea, Carlo Vallati
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Via Diotisalvi, 2 – 56122 Pisa, ITALY
{m.andreozzi, g.stea, c.vallati}@iet.unipi.it

ABSTRACT
Stochastic simulation is an important aid for the design and performance engineering of computer networks. The credibility of simulative results can, however, be seriously affected by human errors (e.g., inconsistencies in the parameter selection, poor initialization of random generators, bugs in the scripts used for post-processing), which become more and more likely and numerous as the dimension of the set of simulated scenarios (*simulation campaign*) increases. The occurrence of such errors can be limited by using reliable *automation tools*, i.e. tools which take care of the above mentioned tasks by using state-of-the-art methodologies. This work describes *ANSWER* (*Automated NS-2 Workflow managER*), a simulation workflow automation tool for the *Network Simulator* (ns-2), explicitly designed for facilitating large-scale simulation campaigns, i.e. those involving many factors. Our framework reduces the space for errors when defining scenarios, controls the execution of a large number of scenarios, and reduces the time overhead required for output data analysis.

Categories and Subject Descriptors
D.2.6 [**Software Engineering**]; Programming environments – *performance measures;*I.6.7 [**Simulation and Modeling**]: Simulation support system – *environments*; G.3 [**Mathematics of Computing**]: Probability and Statistics – s*tatistical software*.

General Terms
Measurement, Performance, Experimentation, Verification.

Keywords
Simulation tools, ns-2, statistical analysis, network simulation

1. Introduction
Stochastic simulation is of paramount importance in aiding the design and performance engineering of computer networks, all the more as technological progress make such systems more and more complex, and consequently less and less treatable through analytical techniques. However, several recent studies have begun to question the scientific credibility of computer network simulation studies, particularly those involving wireless networks (see for instance [1], [2], [3]). It is shown therein that many works, even some published in top-notch conferences and journals, often either lack the essential feature of scientific studies, i.e. *reproducibility*, or are performed according to non-rigorous statistical methodologies. For instance, random number generation, confidence intervals, selection of the initial warm-up period and of the simulation run duration are seldom treated with the necessary care, resulting in less credible analyses. The problem with simulation seems to be that there actually is a lot more to it than simply writing good code. In that respect, to quote [4]: "*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*".

For this reason, developing tools or software layers that support the networking researcher, automating - to the extent possible - the entire *simulation workflow*, is becoming of great practical importance. Henceforth, we refer to *simulation workflow* as the entire process which encompasses a simulation study, from defining the objectives to plotting the final results. A thorough description of the various steps involved in the simulation workflow can be found in [4], as well as in many good tutorials on the subject. These include some speculative steps, like defining objectives and metrics, choosing a suitable simulator, defining the scenarios, which are clearly outside the scope of automation tools. However, they also include quantitative aspects, like checking a scenario for consistency, selecting random generator seeds, determining the length of the initial warm-up period and the number of samples required, running independent replicas of the same scenario until a certain confidence level is reached, balancing the simulation load among many machines, storing the results in such a way that they can easily be analyzed a posteriori. The latter can be more easily (and more reliably) taken care of once and for all by offloading them to a software tool.

Closely related to the problem of credibility, in that this one too can be altogether solved or alleviated by using workflow automation tools, is the problem of *scale*. Quite often, simulation is aimed at proving a single research claim (e.g. "this congestion control schemes achieves a higher throughput than that"), one that can be supported by few graphs in the *performance evaluation* section of a scientific paper. Even when carried out according to state-of-the-art methods, this is work for a single scientist, which often involves producing and analyzing a limited amount of output results. The importance of these data and the need for accessing them usually drops down when the related paper is eventually accepted. When, instead, simulation is used to thoroughly evaluate the performance of a complex system (e.g., a new wireless standard), *large-scale* studies are required. These involve a large number of runs (even when reduction techniques, such as 2^k factorial analysis, are employed), often producing *very large* amounts of data, access to which may be required by third parties for *long times* (e.g., years), and are the results of *team* efforts more often than not. In such cases, without tools taking care of automating some aspects of the simulation workflow, it is simply impossible to get such an amount of work done in the first place. Moreover, the room for mistakes or configuration errors increases with the number of decisions required, i.e. with the size of the job. For instance, the apparently simple problem of devising a coherent naming framework for the output files, so as to facilitate arbitrary post-processing and to enable output data to be correlated to the input scenario, becomes non-trivial when the number of factors which are varied in a simulation study exceeds a few units. Furthermore, the very definition of simulation scenarios when (sets of) factors may vary conditionally depending on the values of other (sets of)

QoSim, 2009, Rome, Italy © 2009 ICST, ISBN 978-963-9799-45-5

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702

factors rapidly becomes a nightmare as the number of factors grows beyond a few units: for instance, when the scenario includes scheduling policies, the scheduler parameters depend on the scheduler type (e.g. a Deficit Round Robin scheduler needs integer-value *quanta*, whereas a Proportional Fair one needs real-valued *weights*). Checking the consistency of a scenario in these settings is likely to become unmanageable (and therefore error-prone) rather quickly.

This work describes *ANSWER* (*Automated NS-2 Workflow managER*), a software automation tool for the *Network Simulator* (ns-2) [5], explicitly designed for facilitating large-scale simulation experiments and publicly available on the Computer Networking Group web site [6]. Ns-2 is probably the most used among a vast number of competitors, due to its open source nature. It is continuously enhanced and extended thanks to the contribution of a large community of researchers. Today, it includes a large number of network protocols, applications, algorithms, in varied environments, both wired and wireless, from large-scale Internet routing to wireless sensor networks. Several works describing simulation workflow automation tools have appeared lately, some of them designed for ns-2, e.g. [7], [8], [9], [10], and [11]. Most of them ([8], [9], [11]), in one way or the other are devoted to animating, visualizing converting or analyzing ns-2 (wired or wireless) *traces*, i.e. ASCII logs of packet transmission events. However, as observed in [12], writing traces requires a huge amount of disk space, which entails additional simulation time and high post-processing overhead. Worse yet, there are many potential simulation output data which are not related in any way to packet traces, e.g. size of routing tables, etc. Therefore, the above tools, albeit useful to the ns-2 user community, are of little help in automating large-scale simulation studies. A software package which enables, up to some extent, simulation workflow automation is Akaroa2 [10]. It allows a user to perform multiple replications in parallel on different processors, with a central process receiving observations of the relevant simulation parameters. The central process estimates the mean value of each parameter and, if the required simulation accuracy is reached, terminates the simulation. While the above tool takes care of some important aspects of the simulation workflow automation, it provides no help to the user as far as managing large amounts of output data is concerned.

The *ns2measure* tool [12], [7], developed at the University of Pisa, represents a good starting point for the ANSWER framework. It provides a set of libraries that enhance the ns-2 data collection capabilities, as well as modules to wrap the execution of a single replica of a simulation scenario, so as to automate independent replications. The ANSWER tool described in this work adds new features to [7]. It provides a graphical user interface to analyze the ns-2 output results and an automated way to configure and run large-scale *simulation campaigns*, i.e. sets of simulation where a number of parameters vary, thus generating different scenarios. More specifically, it is composed of: i) a simulation description language that can be used to design simulation campaigns incorporating many factors in a simple way; ii) a *launcher* module that generates single scenarios from the description and feeds them as an input to the simulator taking care of the statistical issues (e.g., seeds for the random generators), and iii) a *drawer* module, i.e. a graphical interface for output result analysis. The description language is derived from XML, which makes it easier to exchange simulation data among research groups, thus increasing the interoperability and verifiability of the results. The launcher minimizes the possibility of human errors, thus increasing the credibility of the results. Finally, the drawer is a powerful GUI that can be used to correlate large output data sets, enabling an analyst to find relationships that are much harder to find without the aid of automated tools.

The work that most resembles ours is *SwanTools*, [1], which has been developed, independently and almost concurrently, for the SWAN simulator. The architecture of SwanTools is similar to that of ANSWER; it is composed of tools that aid the end user to improve the credibility of the simulation studies. A first set of tools can be used to design and automatically execute the experiments, while a web interface is used for output result analysis. Like ANSWER, SwanTools incorporates a simulation description language (*Domain Modeling Language, DML*). Unlike ANSWER, which stores results in files, SwanTools uses a database, which allows for independent storage of data. SWAN and ns-2 have different architectures, to the extent that porting either tool to the other simulator is probably a prohibitive task. However, we consider the fact that they have similar architectures and underlying concepts a point of strength, and an implicit validation of the ANSWER architecture presented in this contribution. Future work, possibly in collaboration with the developers of SWANTools, will consider incorporating the database features into ANSWER.

The rest of the paper is organized as follows: in Section 2 a brief description of the *ns2measure* module is provided as background. Section 3 describes our ANSWER framework with all its components. In Section 4, a usage example is shown. Finally Section 5 reports conclusions and highlights directions for future work.

2. The ns2measure package for ns-2

This section describes the *ns-2measure* module [7], distributed under the GNU Public License (GPL), which is required in order to run ANSWER. However, it is completely hidden by the latter, so that the user does not have to become familiar with it.

Ns2measure, released as a patch for ns-2, addresses two problems: the collection of samples of metrics and the statistical analysis of output data. As for data collection, it 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 (which, as already said, form ns-2 *traces*). Data collection is performed efficiently, avoiding frequent I/O not to slow down the simulation. The format used in the data log entries simplifies the extraction of information for a posteriori analysis and for generating graphs. 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, avoiding the frequent I/O that would ensue from constructing 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).

As for automating statistical analysis, ns2measure 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 i) in the Tcl scenarios for selecting independent substreams of random numbers (which is seldom done rigorously), and ii) in some post-processing scripting language for computing confidence intervals (which is seldom done at all). However, ns2measure still leaves it to the user to define simulation scenarios. When a *large* set of such scenarios, which differ from one another by few Tcl lines instantiating a single factor, are to be generated, things rapidly get out of control of the simulation user, jeopardizing the credibility of the whole campaign.

3. ANSWER Architecture

ANSWER is composed of two separate modules and incorporates an XML-based description language. An XML file describes a *simula-*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702

tion campaign, i.e. a set of simulation scenarios that can be obtained by varying a number of *factors*, its cardinality being equal to the product of the number of values of all the factors. By *factor* we mean a parameter that is varied in a simulation scenario, e.g. the number of mobile stations in a wireless network. The first module, called *launcher*, parses the XML description file, it creates single simulation scenarios from that by instantiating the variable factors, it feeds each scenario as an input to ns-2 and controls its execution. The second module, called *Drawer*, organizes the simulation results in order to make it easier to plot them through a graphical web interface. The *Drawer* module too takes the XML description as an input in order to get the complete description of the simulation campaign with all its factors and the corresponding values. A possible workflow with the utilization of the ANSWER tools suite is illustrated in Figure 1. The three main components of ANSWER are described in more detail in the following subsections.

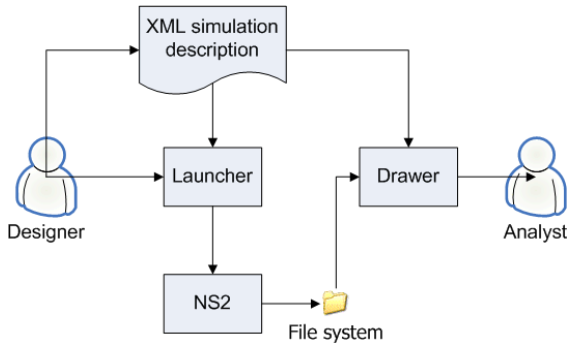

Figure 1. Simulation workflow with ANSWER.

3.1 XML code description

In order to describe and specify a campaign simulation setting, the XML description language has been selected due to its well known support in all operative systems and due to its interoperability between major programming languages. The choice of XML improves the verifiability, repeatability and, therefore, credibility of the simulation work since it allows others to check the results, even using different simulators. Furthermore, when the simulation work is performed in a team, the XML description can be used to exchange the results, so that analysts performing data post-processing can use the XML document as metadata.

The standard XML has been enhanced by defining a syntax capable of describing a simulation campaign. The syntax is based on a new set of tags, which can be divided into two semantic groups. The first one is composed of *simulation management* tags, describing global aspects of the simulation campaign: simulator execution path, statistical parameters and metrics to be collected during simulations. The second group of tags contains the tags that describe all the possible values for the simulation factors which will be varied during the execution of the simulation campaign.

The first group of tags is illustrated in Table 1, along with a brief description of their meaning. A feasible configuration requires at least the path of the ns-2 executable, the name of the base scenario that contains the network description and the default values, the path for the output directory and the statistical parameters about confidence intervals. These last tags are used by the launcher tool to control the execution of independent replications of a given scenario in order to get results with the desired statistical characteristics, as described in the following sub-section. The second group of tags, or *factor tags*, describe the simulation factors. This group is basically composed of two types of tag: <*param*> and <*instance*>. The first one represents the name of the factor that will be varied during the simulation campaign, according to the values specified into its *instance* sub tags. The instance attribute *value* stores a value for the corresponding <*param*> parameter. A nested *param* tag can also be included into an *instance* tag. This means that this parameter will be varied only in correspondence of that instance value.

The *param* tag contains also three more tags. A <*name*>, i.e. a human-readable name for the parameter; a textual <*description*>, describing its role in the simulated scenario; a <*tclname*>, which stores the name of the correspondent Tcl name that must be used by the launcher tool to configure its name within ns-2.

For instance, assume we want to simulate a system where a parameter A takes two values, X and Y. When A is equal to X, another parameter B becomes meaningful. When A is equal to Y, yet another parameter C becomes meaningful (while B has no meaning in this case). The resulting XML code is shown in Figure 2. As shown in the next subsection, simulation scenarios are built by performing the Cartesian product of all *instance*s at the same *param* level recursively. The result is used by the launcher to feed the ns-2 simulator.

Table 1. Generic simulation tags.

Tag name	Tag description
<name>	Human readable name for the simulation campaign. It is used to tag the results displayed by the *Drawer* tool.
<description>	Short textual description of the simulative campaign (e.g., objectives, etc.).
<ns_path>	Path to the simulator executable.
<base_scenario>	Tcl file describing the network topology.
<min_run>	Minimum number of execution runs.
<max_run>	Maximum number of execution runs.
<output_dir>	Directory for output files.
<check_metrics>	Metrics to check.
<check_conf_level>	Desired confidence interval.

```
<param>
        <name>A</name>
        <instance value='X'>
                <param>
                        <name>B</name>
                        <instance value='Z'>
                </param>
        </instance>
        <instance value='Y'>
                <param>
                        <name>C</name>
                        <instance value='J'>
                </param>
        </instance>
</param>
```

Figure 2. A simple XML example.

3.2 Launcher

The *launcher* tool is responsible for providing a correct input for the ns-2 simulator based on the XML campaign description. Its main role is to interface with ns-2, relieving the user of the burden of preparing home-made scripts or using manual commands, all operations that are known to be error-prone. Unlike home-made scripts, the *launcher* does not need to be modified to cope with different simulation campaigns. Indeed, it takes the XML description as an input and uses it to produce the correct data input for the specific simulation campaign execution on ns-2. Our first version of the launcher tool fully supports the ns-2 simulator, and it can be easily modified to interact with other network simulators. A framework module acting as a simulator-adaptable intermediate layer between the researcher and the simulator promotes the interoperability among different systems.

The *launcher* operates in two main steps: it first parses the XML configuration file and then it checks its consistency with the XML extended for simulation language . The consistency check is success-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702

ful if all required *simulation management* tags are specified into the parsed XML file, and if *factor tags* have valid values.
Starting from a correct input file, the launcher tool internally builds a tree data structure representing all the scenarios stemming from the combinations of the factors included in the simulation campaign. This tree is built as follows: a new node is created for each <*instance*> tag and it is connected with the node representing the instance where it is nested, if such node exists. If the instance tag belongs to the top level XML file structure, it is connected to the root of the tree.
For instance, considering again the example shown in Figure 2 the resulting tree will be the one reported in Figure 3.

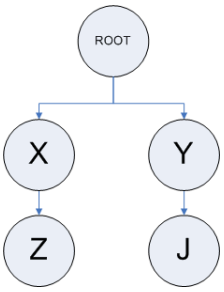

Figure 3. A simple tree.

By navigating the tree, the launcher produces all simulation scenarios which it feeds as an input to ns-2. In fact, each possible path from the root to a leaf represents a simulation scenario, and every node along the path contains a particular parameter value. By examining all the possible paths from the root to the leaves of the tree, the *launcher* builds the list of all resulting simulation scenarios (*simulations list*). The launcher then uses the values stored into the generic simulation tags to guide multiple executions of the ns-2 simulator. Specifically, it takes the ns-2 executable path from the <*ns_path*> tag and the base *network* topology file from the *base_scenario* tag. For each single scenario it instructs ns-2 to run replications, using different and independent initialization seeds for all the involved random number generators. The launcher keeps running independent replications of the same scenario until either of the following occurs (see Figure 5):

- the average values of the collected metrics, i.e., those stored in the *check_metrics* tag, are within the confidence interval specified in the *check_conf_level* tag;
- the *max_run* limit for replications number is reached,

whichever occurs first. The *min_run* parameter forces the execution of a minimum number of replicas as specified inside this tag even if the confidence interval has already been obtained for the specified metrics. At the end of each replica, the results are stored in files named after the collected metrics and located in the *output_dir* directory.
Note that, with launcher, a running simulation campaign can be interrupted at will and resumed directly from the breakpoint without losing the previous data, with a resolution equal to that of a single simulation run. In fact, launcher checks for partial status information before starting a new simulation run.

3.3 Drawer

The *drawer* tool is a graphical web interface that can be used to aggregate and analyze the results produced by ns-2 through the launcher. The *drawer* has been developed using the PHP scripting language, and is meant to be used on a Linux machine within the Apache web server. The drawer takes as an input the XML configuration file describing the simulation campaign, which is required in order to correctly reconstruct all parameter names and preconditions, and the output files containing the simulation campaign output metrics, produced according to the ns2measure file format. In fact, these files contain comma separated values. The last two values on a row are respectively the metric value and its confidence interval while the former values represent the instances of the simulation factors according to which that metric was collected. A simple example is shown in Figure 4. Now, the correct interpretation for those factors (including a human readable description) is included in the XML file. Therefore, in order to present human-readable results, the XML file is needed as well.

```
X,10,0.001
Y,15,0.002
```

Figure 4. A simple output structure.

As a first form, the *drawer* tool presents the user with an interface where she can select one of the metrics collected during the simulation runs. Then, it displays a second form where the user selects the simulation scenarios for which the selected metric has to be plotted. In the next and last form the user inputs the labels to be inserted into the graph, along with other style options as label position, font, and so on.
The graphs are drawn by invoking the open source *gnuplot* tool through a PHP shell execution command. After the requested graphs have been stored locally at the web server, the *drawer* tool shows its last form, where the resulting image is shown through the web browser. At this point the user can download images produced by the *gnuplot* software, choosing among PNG, EPS, CSV or gnuplot source scripting formats. The *drawer* tool supplies the chosen file format for download by using internal format conversion routines. Figure 6 shows a screenshot of the *drawer* interface.

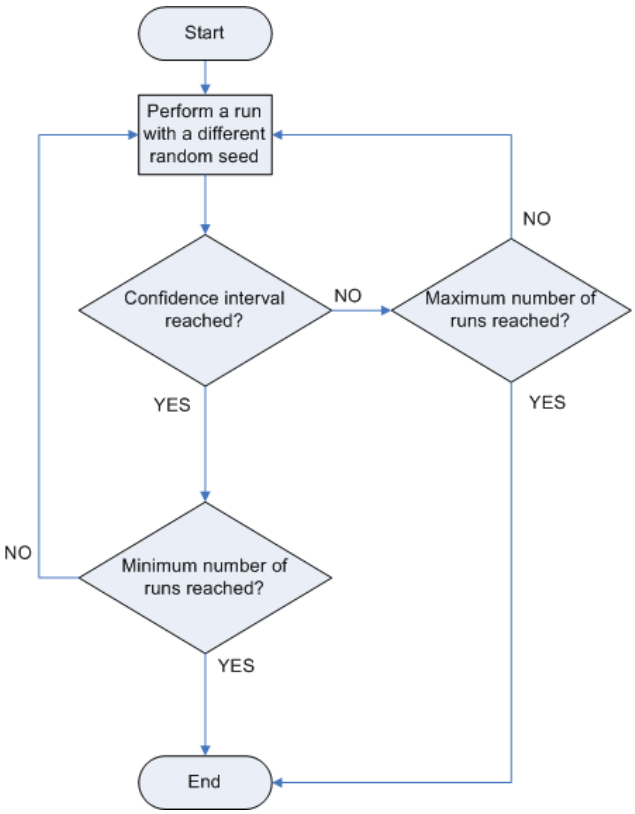

Figure 5. Launcher flow chart for a single scenario.

3.4 Portability

As can be seen from Figure 1, our framework drives the ns-2 simulator in order to obtain results that can be analyzed through a graphical web interface. Obviously, the core of that system is represented by the simulator itself. Although it is probably the most common network simulator, ns-2 is not the only one. A new release, called ns-3,

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702

is currently under development. The latter is going to be very different from ns-2: for instance, Python will be used instead of Tcl for configuring scenarios.

The ANSWER architecture has been designed with a specific attention to modularity, which makes it relatively easy to adapt it to other simulation environments. In particular, the launcher can be modified to work with other simulators with relatively little effort: if the target simulator allows the definition of a base scenario in a file and accepts the factors through the command line, our module can work with it without any modifications. Otherwise, the script can be easily extended modifying just a few functions that implement the interface with the simulator.

Adapting the drawer to other simulators appears to be trickier, due to its reliance on the structure of the output results. If a simulator uses the same output file structure as ns2measure, the drawer can be used to analyze the results without any problem. For instance, ns-3 developers foresee a metrics collection system similar to that of ns2measure [13], which would make ANSWER easily portable to ns-3. Otherwise if the simulator has a different output structure the drawer needs modifications; however, the changes are limited only to the functions that load the results into memory.

4. Usage Example

As previously said, ANSWER is designed to help a user to manage large simulation campaigns aimed at evaluating a complex system. In these cases it is reasonable to assume that the number of the simulation scenarios is large. ANSWER helps one to drive neatly the simulator to obtain the desired results and to easily analyze the results. Moreover a large number of simulation runs can be described synthetically through a single XML file. In particular, XML allows one to represent complex scenarios without sacrificing readability. In this section a simulation campaign is described in order to demonstrate the power of the XML description. The case study is a centralized wireless network with a Base Station (BS) where a scheduler allocates the resources. The comparative evaluation of scheduling policies is the objective of the simulation campaign. Note that a scheduler is defined by multiple factors, which are sometimes interdependent. For instance, one such factor is the name of the scheduling policy itself, Deficit Round Robin (DRR), Proportional Fair (PF), Earliest Deadline First (EDF) and Max C/I. However, different scheduling policies require different configurations, and, accordingly, different factors. For instance, DRR schedulers depend on integer *quanta*, the PF schedules has real-valued *alpha* parameter, the EDF scheduler need as input the value of the offset used to mark the packets and, finally, the Max C/I wants no parameters. This conditional variation of parameters is neatly accounted for in the XML file by using nested tags. The above example is described in Table 2 and Table 3. Table 2 illustrates the factors and the relative values while Table 3 shows the pre conditions required to make a factor meaningful. The factors without preconditions are factors that are always meaningful at each simulation scenario and therefore each case needs a value for that. For example, the simulator needs the value of the number of mobile stations and the scheduling policy to build a feasible scenario.

Table 2. Example parameters variation.

`Scheduler`	Deficit Round Robin (DRR), Proportional Fair (PF), Max C/I (MAXCI), Earliest Deadline First (EDF)
`Quantum`	190,1000 and 10000 bytes
`Alpha`	0.1, 0.3, 0.5, 0.7 and 0.9
`Offset`	20 and 40ms
`Packet expire`	40, 80 ms
`MSs number`	70, 80 and 90

Table 3. Example parameters pre-condition.

`Scheduler`	No pre-conditions
`MSs number`	No pre-conditions
`Quantum`	Scheduler == DRR
`Alpha`	Scheduler == PF
`Offset`	Scheduler == EDF
`Packet expire`	Scheduler == DRR or Scheduler == PF

In a full factorial simulation campaign all these parameters should be varied. However, it would be useless to simulate the PF scheduler varying the quantum because the latter has no influence on a PF scheduler. Therefore, in order to avoid redundant runs, a home-made shell script that launches the ns-2 instances should be written with some care, and probably with a lot of nested *if* clauses. In that case, it would be poorly verifiable and manageable. For instance, adding a new factor once the script is complete could easily lead to rewriting large parts of it. Using our script, the first step is to edit the XML file in order to specify the desiderate campaign. The XML example shown in Figure 8 describes the simulation campaign for assessing the performance of the schedulers. As can be seen, the nested structure of XML tags easily represents conditional parameters, whose existence is tied to specific values of other parameters. The first tag provides statistical information and descriptive data, like the number of runs or the confidence intervals. Under the *param* tag a new parameter is included with the correspondent values specified by the *instance* tag. Inside a value, a new set of parameters can be specified. In the example, if the selected scheduler is DRR, PF or EDF, one or more factors become meaningful, otherwise (i.e., in the MAXCI

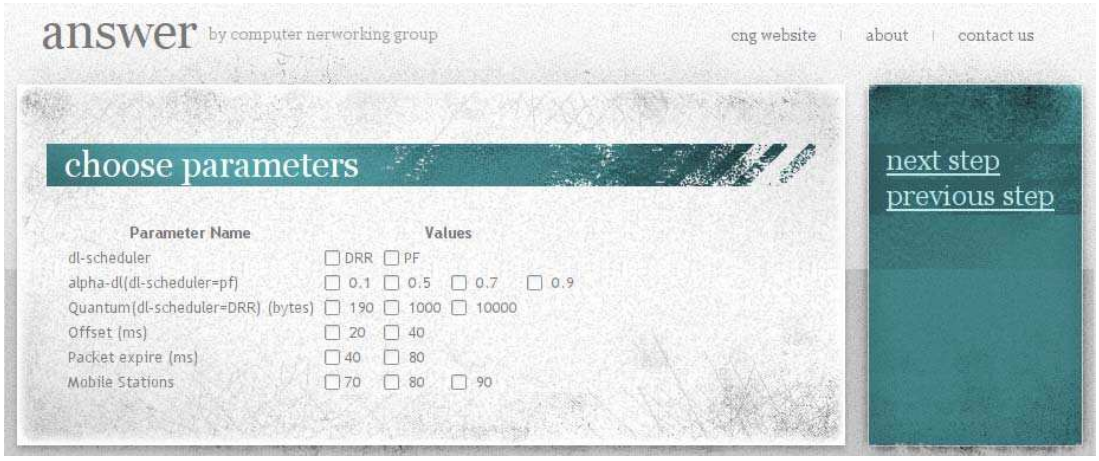

Figure 6. A screenshot of the drawer interface.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702

case) there is no additional parameter. Adding a new value or a new parameter only requires inserting new tags, without modifying any program or script. The simulation scenarios are created by the launcher module that performs the Cartesian product of the factors at the same level. For example, in this case the performance of each scheduler is evaluated for every number of mobile stations and if the scheduling policy is the DRR the simulations are performed for each quantum paired with expiration time value. Figure 7 illustrates a sample of the scenarios resulting from the XML presented here.

After the simulation runs have completed, the resulting output data can be analyzed using the graphical web interface. As described in the previous section, this web interface takes as input the raw simulation data and the XML file to analyze the results. At this point the analyst can easily access the simulation results and make graphs with them.

```
-nodes 70 -psDiscipline pf -pfAlpha 0.1 -pkt-expire 40
-nodes 80 -psDiscipline pf -pfAlpha 0.1 -pkt-expire 40
-nodes 90 -psDiscipline pf -pfAlpha 0.1 -pkt-expire 40
…
-nodes 70 -psDiscipline pf -pfAlpha 0.9 -pkt-expire 80
-nodes 80 -psDiscipline pf -pfAlpha 0.9 -pkt-expire 80
-nodes 90 -psDiscipline pf -pfAlpha 0.9 -pkt-expire 80
-nodes 70 -psDiscipline drr -drr-quantum 190 -pkt-expire 40
-nodes 80 -psDiscipline drr -drr-quantum 190 -pkt-expire 40
-nodes 90 -psDiscipline drr -drr-quantum 190 -pkt-expire 40
…
-nodes 70 -psDiscipline drr -drr-quantum 10000 -pkt-expire 80
-nodes 80 -psDiscipline drr -drr-quantum 10000 -pkt-expire 80
-nodes 90 -psDiscipline drr -drr-quantum 10000 -pkt-expire 80
-nodes 70 -psDiscipline maxci
-nodes 80 -psDiscipline maxci
-nodes 90 -psDiscipline maxci
…
-nodes 70 -psDiscipline edf -edf-offset 40
-nodes 80 –psDiscipline edf -edf-offset 40
-nodes 90 -psDiscipline edf -edf-offset 40
```

Figure 7 – Part of a sample scenario for evaluating schedulers.

5. Conclusions and Future Work

In this paper we described ANSWER, a simulation workflow automation tool which is useful for both improving the credibility of simulations and making large-scale simulation campaigns manageable. Our experience in performing large-scale simulation studies confirms that ANSWER is effective in reducing the possibility of human errors due to script errors and cuts the time spent to obtain results. In particular the visual interface for data analysis has proved valuable in reducing the post-processing analysis time, thus speeding up the entire process. Finally the use of a flexible and easily extendible simulation description language based on XML has improved the productivity of our team.

There are several directions in which the ANSWER framework can be extended. We are currently working on incorporating database storage of the simulation results, which would open new possibilities for output data analysis. As far as the XML document is concerned, a possible extension is to enhance the set of controls that are performed through the definition of a *Document Type Definition* (DTD) that describes the structure of the XML document with its semantic rules. Through a DTD document the launcher module will be able to verify not only the syntax of a XML scenario, but also the *correctness* of its structure. With this new feature, the user would be sure not only that a document is well formed but also valid. It is our belief that moving progressively the scenario definition from Tcl to XML would on one hand increase the interoperability of ns-2 with different simulators, and, on the other, increase its verifiability and reuse. This is therefore another direction of enhancement of this work.

Furthermore we are working on extending the launcher tool to take advantage of parallel computing in order to distribute the load according to the Multiple Replication in Parallel (MRIP) approach [10], so as to speed up the simulations. Algorithms for 2^k factorial analysis can also easily be incorporated into the launcher tool, so as to further reduce the number of generated scenarios. A graphical web interface to launch and remotely control the simulations is currently under design, with the aim of further improving the usability of our framework and of reducing its learning curve.

We are currently using ANSWER for didactic purposes, exploiting it in a fifth-year undergraduate course of Advanced Networking Systems. Our objective is to facilitate a steeper learning curve of the students, enabling them to focus their attention on the simulation objectives rather than on tweaking ns-2 and generating self-made scripts to launch simulation batches or to post-process output data.

6. References

[1] L. F. Perrone; C. J. Kenna; B. C. Ward, "Enhancing the Credibility of Wireless Network Simulations with Experiment Automation," Networking and Communications, 2008. WIMOB '08. IEEE International Conference on Wireless and Mobile Computing, , vol., no., pp.631-637, 12-14 Oct. 2008.

[2] S. Kurkowski, T. Camp, M. Colagrosso. MANET simulation studies: the incredibles. Proc. ACM SIGMOBILE MC2R, vol. 9, no. 4, Oct. 2005, pp. 50-61.

[3] 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.

[4] L.F. Perrone, C. Cicconetti, G. Stea and B. Ward, “On the Automation of Computer Network Simulators”, in Proc. of SIMUTOOLS 2009, Rome, 3-5 March 2009.

[5] http://nsnam.isi.edu/nsnam/, ns-2 simulator home page.

[6] http://cng1.iet.unipi.it/wiki/index.php/ANSWER

[7] C. Cicconetti, E. Mingozzi, G. Stea. Measurement module for ns-2. http://info.iet.unipi.it/~cng/ns2measure/, last update May 2006.

[8] S. Kurkowski, T. Camp, and M. Colagrosso. A visualization and animation tool for NS-2 wireless simulations: iNSpect. In Proc. *MASCOTS*, 2005.

[9] J. Malek. Trace graph – Network Simulator NS-2 trace files analyser. http://www.tracegraph.com/, last update Jan. 2006.

[10] G. Ewing, K. Pawlikowski, and D. McNickle, Akaroa2: Exploiting network computing by distributing stochastic simulation. . In: Proceedings of the European simulation multiconference (ESM'99), SCS, Warsaw. pp. 175-181.

[11] D. Savić, M. Pustišek, and F. Potortì, “A tool for packaging and exchanging simulation results," in Proc. of VALUETOOLS'06, Pisa, Italy, October 11-13, 2006.

[12] C. Cicconetti, E. Mingozzi, and G. Stea 2006. An integrated framework for enabling effective data collection and statistical analysis with ns-2. In Pro. WNS2‘06, Pisa, Italy, October 10, 2006.

[13] T. R. Henderson, S. Roy, S. Floyd, G. F. Riley. ns-3 project goals, in Proc. of WNS2’06, Pisa, Italy, October 10, 2006.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702

```
<?XML version="1.0" encoding="UF-8"?>
<simulation>
        <!-- Management informations -->
        <name>Scheduler</name>
        <description>Scheduler evaluation</description>
        <ns_path>path/to/ns</ns_path>
        <base_scenario>base.tcl</base_scenario>
        <min_run>5</min_run>
        <max_run>5</max_run>
        <output_dir>savefile</output_dir>
        <check_metrics>metrics_to_check</check_metrics>
        <check_conf_level>0.95</check_conf_level>
        <!—Scenario description -->
        <multicell>
                <param>
                        <name>NMS</name>
                        <description>Number of mobile stations</description>
                        <tclname>nodes</tclname>
                        <instance value="70"></instance>
                        <instance value="80"></instance>
                        <instance value="90"></instance>
                </param>
                <param>
                        <name>Scheduler</name>
                        <description>Type of scheduling policy</description>
                        <tclname>psDiscipline</tclname>
                        <instance value="pf">
                                <param>
                                        <name>pf-alpha</name>
                                        <description>Alpha parameter for PF</description>
                                        <tclname>pfAlpha</tclname>
                                        <instance value="0.1"></instance>
                                        <instance value="0.3"></instance>
                                        <instance value="0.5"></instance>
                                        <instance value="0.7"></instance>
                                        <instance value="0.9"></instance>
                                </param>
                                <param>
                                        <name>expire</name>
                                        <description>Packet expire</description>
                                        <tclname>pkt-expire</tclname>
                                        <instance value="40"></instance>
                                        <instance value="80"></instance>
                                </param>
                        </instance>
                        <instance value="drr">
                                <param>
                                        <name>quantum</name>
                                        <description>Quantum for drr scheduler</description>
                                        <tclname>drr-uantum</tclname>
                                        <instance value="190"></instance>
                                        <instance value="1000"></instance>
                                        <instance value="10000"></instance>
                                </param>
                                <param>
                                        <name>expire</name>
                                        <description>Packet expire</description>
                                        <tclname>pkt-expire</tclname>
                                        <instance value="40"></instance>
                                        <instance value="80"></instance>
                                </param>
                        </instance>
                        <instance value="maxci"></instance>
                        <instance value="edf">
                                <param>
                                        <name>offset</name>
                                        <description>Edf offset</description>
                                        <tclname>edf-offset</tclname>
                                        <instance value="20"></instance>
                                        <instance value="40"></instance>
                                </param>
                        </instance>
                </param>
        </multicell>
</simulation>
```

Figure 8. XML example.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5702
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5702