AN EXPERIMENT DESIGN FRAMEWORK FOR THE SIMULATOR OF WIRELESS AD HOC NETWORKS

by

Christopher J. Kenna

A Thesis

Presented to the Faculty of

Bucknell University

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science with Honors in Computer Science

August 21, 2008

Approved:

Felipe Perrone
Thesis Advisor

Xiannong Meng
Chair, Department of Computer Science

Acknowledgments

Before I begin, I would like to thank everyone for their support not only during the writing of my Honors Thesis, but throughout my career at Bucknell, especially:

- Luiz Felipe Perrone, for his help, understanding, and patience throughout the project. Since my first year at Bucknell, Felipe has served as a mentor, providing sound advice, and believing in me even when I questioned myself. Thanks for everything Felipe, you are a true friend.
- Bryan Ward, my colleague (class of 2011), for spending the better part of a day (or more) with me merging some of his code into my project. It was Bryan's idea to use DRb to distribute the simulations, and he provided some clever solutions to tricky situations. Good luck, Bryan!
- Christina Howard, for tolerating my elevated stress and shortened phone calls as deadlines approached. Thank you for your understanding and encouragement!
- My family and friends, who supported me while I wrote my thesis.

Contents

List of Figures

Abstract

A survey of literature has indicated a crisis of credibility in Mobile Ad Hoc Network (MANET) simulation studies. There are many opportunities for researchers to make mistakes that render studies unreliable, as they require a great deal of organizational methodology. I discuss a web-based interface I developed that seeks to enhance the believability of these studies. It consists of a series of carefully planned steps to guide the experimenter in creating reliable MANET simulation studies. The interface guides experimental design, helps with the execution of multiple, simultaneous experiments using Distributed Ruby (DRb), stores the results in a database, and creates mechanisms for the simple analysis of simulation output data. I discuss several common pitfalls of simulation studies in literature, and how the interface prevents users from falling into these traps.

Chapter 1

Introduction

1.1 Concepts in Simulation

Computer simulation allows a researcher to implement a model of an actual, physical system in software in order to gain insight into the system without actually interacting with it. This abstraction tool is used by researchers in many scientific areas, and is an invaluable tool in fields such as chemistry, physics, and, of course, computer science. According to Law and Kelton (2000), "Simulation is one of the most widely used operations-research and management-science techniques, if not *the* most widely used." Computer simulation usually provides numerous benefits when compared to actually running a live experiment. For instance, these benefits include a decrease both in cost, and in the amount of time an experiment has to be run. Furthermore, since simulations do not require any equipment other than a modern computer, they may be used to generate insight into systems that would otherwise be too costly to experiment with in real life. Finally, in some cases, a process that normally takes place over a long time period can be simulated faster than it would occur in real time by altering the speed at which the simulation executes.

For example, a researcher wishing to model data flowing between computers on a local area network (LAN) may not have the funding to purchase all of the equipment necessary to create a LAN. This has the potential to be expensive, as multiple computers, vast lengths of Ethernet cable, and numerous switches would be required to implement the infrastructure of an experimental LAN. While the actual infrastruc-

ture would be costly, a computer simulation programmed to model data flow between computers on a LAN can be implemented on a single machine. This simulation based solution is obviously more economical than running an experiment with a tangible network.

Another benefit of simulations is that they sometimes are able to simulate events significantly faster than they would occur in the physical world. This is because simulations are designed to strip away the unnecessary details of the system being studied. When a researcher is constructing a simulation to represent some real-world object he is studying, such as a LAN in our example, he attempts to reduce the system to only parts that are statistically relevant. In other words, the researcher attempts to simulate *only* the parts of the real-world system which will produce interesting and meaningful results. This abstraction technique results in a simulation that is much simpler than its real-world counterpart. As a result, the simulation does not perform every event that would occur in the real-world, and can sometimes execute in faster than real time. Getting back to our example, a researcher who wishes to simulate several days of network traffic relatively quickly may find this to be to his benefit. Obviously, this saves time, making life somewhat less stressful (hopefully).

Simulation may seem to be a wonderful solution to common problems facing researchers, but creating a simulation is nontrivial. To perform a computer simulation, a researcher must follow a series of steps, as illustrated in Figure 1.1. First, he designs a mathematical model that approximates the behavior of the process she is studying. Second, she implements the mathematical model in software so that it can execute on a computer. She may implement the mathematical model in an existing simulation framework; however, an ambitious researcher, or one who is performing research in an area that does not yet have a simulation framework, may elect to program a custom simulator in a high level programming language. Finally, the computational model is executed and the results it produces are analyzed with statistical methods.

1.2 Wireless Networks Overview

My project is defined in the context of the simulation of wireless networks, and, therefore, a brief introduction to the two most common types of wireless networks is important to understand later parts of the thesis. Many different types of wireless networks exist; however, two of the most predominant are infrastructure based networks, and ad hoc networks. These types of networks differ in the way the con-

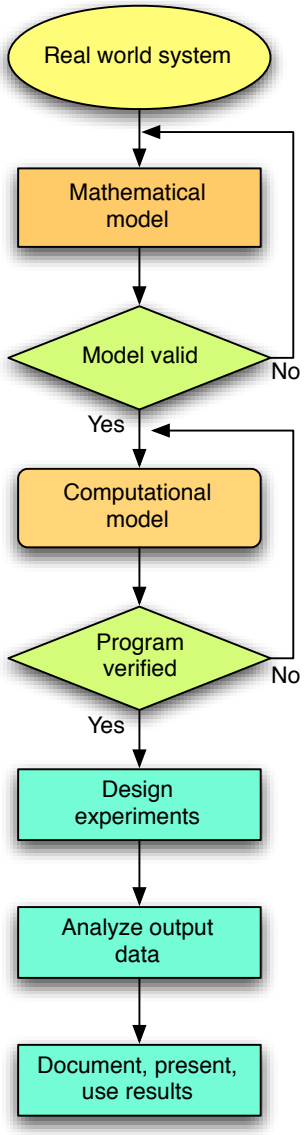

Figure 1.1: The simulation process.

nections among the elements of the network are established and maintained. The arrangement of the elements (hosts, network links, etc.) in a network is called the *network topology*.

1.2.1 Infrastructure Based Networks

Infrastructure based networks have one or more stationary access points that act as gateways for all of the computers, or *nodes*, in the network. This means that a network node *always* directly communicates with the access point. In order for two nodes to communicate, they must relay their traffic through the access point, which will in turn forward the message to the proper destination. The access point acts as the router for all hosts connected to it.

As a visual aid, if we imagine that each node has an "invisible" wireless link between itself and the access point, the topology of the network would be similar to that of Figure 1.2. The access point is in the center of the figure, and all hosts radiate out from it. The access point connects to other arbitrary networks, such as the Internet, or a campus network. Infrastructure based networks are carefully constructed and managed by humans, as the location of wireless access points impacts network performance.

1.2.2 Ad Hoc Networks

Ad hoc networks, on the other hand, are not managed by humans and do not rely on any preexisting infrastructure. By definition, an ad hoc network is decentralized and spontaneously forms when nodes move close enough to one another so that they are able to communicate. The network is ad hoc because predetermined algorithms take charge of establishing a routing system that passes messages from a source host to destination host based on network connectivity. The network is therefore self configuring. In an ad hoc network, nodes communicate directly with one another, and are also responsible for routing all messages. This stands in contrast to infrastructure networks, where the access point assumes this role. Because the network is self configuring, there is no well defined network topology. However, wireless links are typically formed between a node and its adjacent neighbors within the node's wireless range. Links are established and destroyed as hosts join and leave the network, respectively. The topology would then approximate a partially connected mesh.

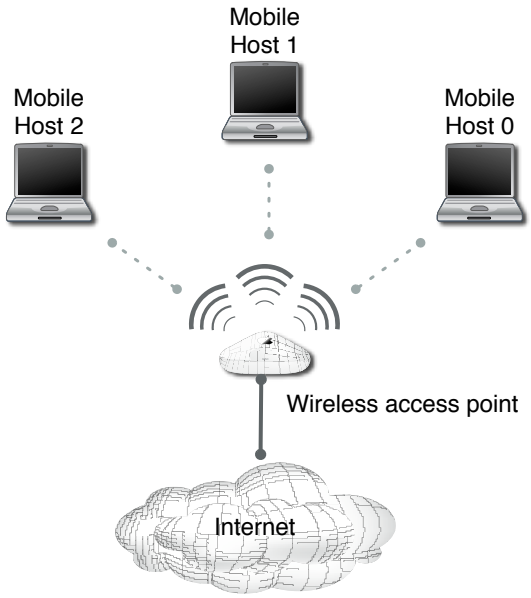

Figure 1.2: Architecture of an infrastructure based wireless network (access point) with connection to the Internet

A mobile ad hoc network, or MANET, is really nothing more than a special type of wireless ad hoc network. Similar to the ad hoc networks mentioned above, a MANET is self configuring. The difference is that in a MANET the hosts are completely free to move about randomly. The nodes may or may not roam about in an ad hoc network, but in a MANET at least some nodes are assumed to be in motion. The result is a constantly changing, arbitrary arrangement of the links in the network. Because of the fickle nature of the network links, any approximation of the network topology is quickly outdated when a node moves out of range of its neighbors. Figure 1.3 is an example of what a MANET could look like. The nodes are shown as laptops to emphasize the fact that they are free to move about. The wireless links between them are created and destroyed as they roam around.

1.3 SWAN Overview

The Simulator for Wireless Ad Hoc Networks, or SWAN, is a project which aims to replicate a real MANET through computer simulation. SWAN started at Dartmouth

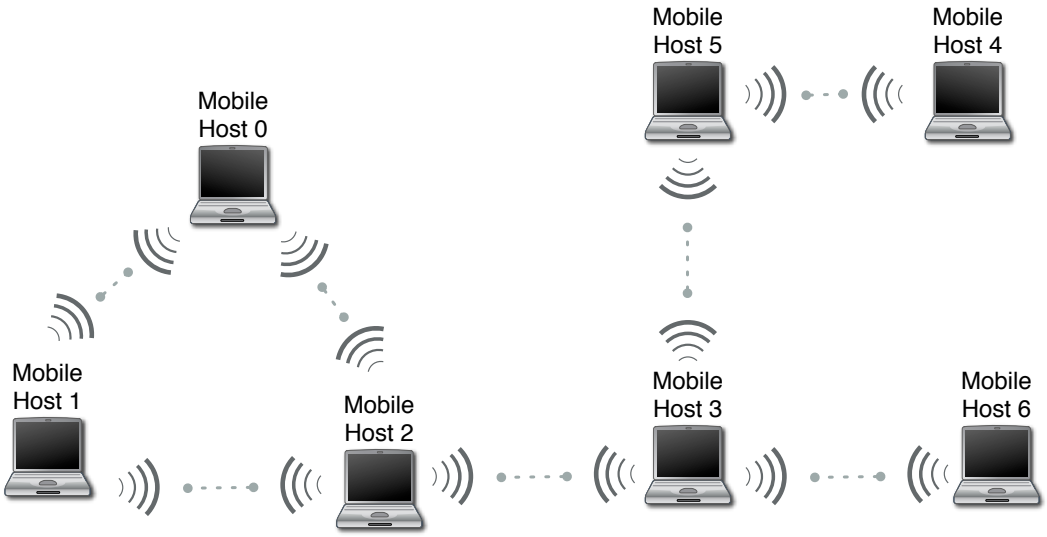

Figure 1.3: Instantaneous "snapshot" of MANET network architecture.

College in its Institute for Security Technology Studies in 2000. Presently, Luiz Felipe Perrone of Bucknell University is the maintainer and primary contact for SWAN.

The goal of SWAN is to produce a virtual environment to study the many parameters that can affect MANET performance. Using SWAN, models of physical process can represent environmental effects, such as the dispersal of a chemical plume, or variations in temperature and barometric pressure. These environmental effects interact with models of wireless hosts that can be constructed to react to some characteristic. In turn, this characteristic can be measured, and the hosts can respond to variations of the characteristic by communicating with one another. SWAN records statistics pertaining to these interactions for later analysis.

1.4 Creating Simulation Experiments

This section discusses the process of creating a simulation experiment. By "experiment" I mean the execution of a computer simulation model. Creating such an experiment is not always a straightforward task. In fact, large experiments, such as those executed by SWAN, are quite complex. However, simulations can be carefully constructed to minimize the amount of work that a researcher and computer simulation must perform in order to gain useful experimental results. We refer to these

techniques as *experimental design*.

1.4.1 Modeling and Simulation

Law and Kelton (2000) describe a *system* as the collection of entities that "act and interact together towards the accomplishment of some logical end" (3). The *state* of a system is the collection of variables that describe the system at a particular time. The collection of entities that make up this system depends upon the goal of the simulation study. For example, in SWAN, if the researcher wishes to find something out about the network as a whole, the system could be all of the hosts in a MANET. Or, if he only wants study the effects of some event on a particular host, the system could be only a few hosts in the MANET.

As discussed in Section 1.1, it is often not cost effective to study the actual, physical system. Because of this, it is necessary to build a *model* of the system. The model takes the place of the actual system, and is the representation that a researcher will study instead. The model may be either a physical model, or a mathematical model. A physical model is something that one could actually reach out and touch. In other words, it actually exists in the physical world. Someone researching automobile aerodynamics is likely to use a physical model. A researcher in this field may build a scaled down version of a car that is being considered for future production, and then place it in a wind tunnel to observe how it interacts with the air flowing around it. SWAN does not use a physical model, but rather, a mathematical model. A mathematical model represents a system in terms of a set of variables to represent its state. The variables are then manipulated to see how the mathematical model reacts. A simple mathematical model used to represent the velocity of a body in motion is shown below, where $\bar{\mathbf{v}}$ represents velocity, and $\Delta\mathbf{x}$ represents the displacement of the body during the time interval Δt.

$$\bar{\mathbf{v}} = \frac{\Delta\mathbf{x}}{\Delta t}$$

The purpose of constructing a mathematical model is so that a researcher can use it to answer questions about the real system that it is supposed to represent. If the mathematical model is simple enough, it may be possible to use mathematical techniques to achieve an exact, *analytical* solution. For instance, if the mathematical model above represents the velocity of a body in motion, and we know that

acceleration is the rate of change of velocity, then we can use calculus to derive the acceleration of the body. The acceleration $\mathbf{a}$, would be given by the formula below.

$$\bar{\mathbf{a}} = \frac{\mathrm{d}\bar{\mathbf{v}}}{\mathrm{d}t}$$

If the formula for $\bar{\mathbf{v}}$ is simple, the derivative is obtainable with only a basic knowledge of calculus. However, some analytical solutions may become very complex, or even infeasible. When a system is too complex to be studied analytically, one can resort to simulation.

Now that we finally understand what a simulation experiment is, we need to be able to talk about its different parts. In a simulation experiment, we call the input parameters of an experiment *factors*, and the output performance measures *responses*. These factors are nothing more than the variables that make up the state of the system. A factor can be either quantitative or qualitative, where a quantitative factor, assumes discrete numerical values, and a qualitative factor represents something that can not naturally be quantified. Similarly, the performance measures are the qualitative or quantitative results obtained after executing the simulation.

1.4.2 What is Experimental Design?

Careful experimental design lets a researcher create the specific set of factors that he or she will need to produce results for a particular circumstance with the least amount of simulating. Law and Kelton (2000) write that the main benefit of carefully designed experiments is that they let a researcher work much more efficiently than randomly selecting a series of factor configurations unsystematically just to see what happens.

To illustrate this point, consider a simple model with two factors. The value that each one of these factors is set to is called the *level* of the factor. If we would like to examine the two factors at three levels, a simulation that covers all possible combinations of these factors would require $3^2 = 9$ different simulation runs. We call each run for a given combination of factors a *design point*. The combinations are shown in Table 1.1.

A researcher would be interested in obtaining results for all of these possible combinations because it lets him get an initial estimate of how each factor affects the

Factor Combination (Design Point)	Factor a	Factor b
1	1	1
2	2	1
3	3	1
4	1	2
5	2	2
6	3	2
7	1	3
8	2	3
9	3	3

Table 1.1: All combinations of two factors with three levels each. $3^2 = 9$ design points total.

responses. Measuring the effect of a particular factor can be done in this way. In general, if we have k factors and n levels for each factor, and we wish to determine the effect of a particular factor, we can fix the level of the other $k - 1$ factors at some set of values and make simulation runs for each of the n levels of the factor we are interested in. This results in n total simulation runs. Repeating this in order to examine all of the other factors, one at a time, results in n^k simulation runs.

This strategy turns out to be inefficient for larger experiments. Furthermore, it assumes that there are no interactions between the factors. A better design, one that Law and Kelton (2000) call a more "efficient" design, is called a 2^k factorial design. With 2^k factorial design, we choose just two levels for each factor, and then perform simulation runs at each of the 2^k possible factor-level combinations.

There is no algorithm or prescribed methodology for setting the two factor levels. Law and Kelton (2000) write that the levels should be opposite of each other, while still retaining some degree of realism. They also caution that levels should not be so far apart that they mask potentially important aspects of the response.

Choosing a design to exercise the parameters one would like to research is difficult. 2^k factorial design simplifies this process by limiting the possible levels of each factor, in turn reducing the total number of simulation runs.

1.5 Problems with Simulation Studies

According to Merriam-Webster (2008), the scientific method is "principles and procedures for the systematic pursuit of knowledge involving the recognition and formulation of a problem, the collection of data through observation and experiment, and the formulation and testing of hypotheses." It is important that this method is followed, and that the details of the experimental setup are shared so that readers of any particular MANET publication are able to replicate any experiment they come across. This is what gives research its capacity for belief, or credibility, which is the most important characteristic of an experimental study. Unfortunately, this is not the case with experimental studies today.

Camp et al. (2005) and Kurkowski (2006) show that the vast majority of MANET simulation studies are unreliable. They believe that credibility in experimental research is contingent upon four factors. First, experiments must be repeatable. Second, experiments must produce results that are not specific to a single scenario. Third, the experiments must exercise the characteristic(s) of the system that we want to study. Finally, statistical analysis of experimental results must be carried out with proper statistical methods. For my Honors Thesis, I am particularly interested in two of these aspects, the first and the last.

1.5.1 Experiment Reproducibility

A reader ought to be able to repeat experiments found in any particular journal article, conference proceeding, or other document; however, the experiments are rarely repeatable because a vast majority of experimental factors remain unpublished for the sake of reducing the page count of a paper. If the results and conclusions documented in a publication cannot be replicated then the study lacks credibility. Both Kurkowski (2006) and Perrone et al. (2003) identified that making code and configuration files from the simulation study available to researchers would greatly improve the repeatability of an experiment, but the typical scientific journal article or conference proceedings paper does not allow adequate space for an experimental parameter value listing.

As a small example of the problem, Camp et al. (2005) found that when they surveyed 114 conference papers, only 34 mentioned what simulator was used in the simulation study. Kurkowski (2006) says that this directly compromises the repeata-

bility of the simulation. He also writes that researchers may use default factor levels when conducting a simulation study. This is dangerous because the factor levels in question could change between different releases or versions of the simulation software, thereby introducing an anomaly into future simulation results.

1.5.2 Output Data Analysis

According to Camp et al. (2005), the analysis of the data resulting from MANET simulations is the downfall of many studies. Confidence in statistics is greatly improved as the number of samples increases; therefore, performing an experiment only once, using too few samples, neglecting to compute confidence intervals, or using wrong methodology are common pitfalls that reduce experiment credibility.

Kurkowski (2006) noticed, for example, that considering the first set of results from a simulation experiment to be the way that the real system would function, and failing to mention (and possibly vary) the pseudo-random number generator's (PRNG's) seed for experimental runs are two common errors. In the first case, there is little confidence in the results of the simulation because it was only executed once. In the case that the PRNG's seed is never changed, the same output would be produced for every run of the simulator. This results in incredible statistics, and introduces a false confidence in the results of the experiment.

1.6 Enhancing the Credibility of Simulation Studies

MANET simulation studies require a great deal of organizational methodology. Both Kurkowski (2006) and Perrone et al. (2003) call for detailed descriptions of published experiments to be available on-line so that researchers can further study a MANET model they find in the literature. Unfortunately, the vast configuration files required to perform a simulation study cannot be published in the small amount of space available in conference and journal articles. Finally, Camp et al. (2005) writes that those running simulations often err by leaving key parameters undefined, carelessly using default parameter values embedded in the simulator, or incorrectly analyze the output data.

Thesis Statement

My thesis is that in order to make MANET simulations more credible, one can rely on a system that guides the experimental design, helps with the execution of multiple, simultaneous experiments, stores the results in a database, and creates mechanisms for the easy visualization of simulation output data. My contribution to the MANET community is a web based application to achieve all these goals and a case study demonstrating its positive impact. Because researchers sometimes neglect one or more of these aspects of the established methodology mentioned in Section 1.5, few experiments described in current literature can be successfully replicated. My thesis addresses the two issues highlighted above and proposes a solution to avoid them. Subsequently, the contribution of my work makes steps towards enhancing credibility and reliability in MANET simulation studies.

To address these problems, I propose a web based framework which facilitates organization in MANET simulation. The web based framework presents the end user with a simple *wizard-style* interface accessible from any web browser. The interface consists of a series of carefully planned steps to guide the experimenter in creating reliable MANET simulation studies in the style of the scientific method. This guidance helps researchers automate tasks that would otherwise be confusing, error-prone, or difficult to those who are unfamiliar with MANET simulation.

The initial step in a SWAN simulation is experiment generation, thus, a user of the interface is guided through this process. All of the configuration parameters used by SWAN are stored in the database so they are easily accessible for later use. The framework then executes SWAN using the generated experiment. Finally, the results of the experiment are archived in the database. In doing so, both the configuration parameters and the results of experiments are at hand for quick reference at the time a publication is prepared.

When an experiment is complete, the framework mines the database of results and applies rigorous output analysis methodology to the data. A common pitfall of simulation studies is either accepting the first set of results as a generalized result for all scenarios of a simulation, or not using proper statistical analysis techniques with varying forms of output. My framework automates statistical analysis, thereby reducing the probability of human error. An article by Sanchez (2005) on experimental design has brought to light efficient ways in which to implement and analyze simulations. I implement some of these techniques, which are explained Chapter 2.

Chapter 2

Related Work

Some amount of work has been done in each of the areas I wish to address in my thesis. I noted that simulation experimenters frequently use a series of shell scripts to automate some or most of the simulation process, as the basic idea for creating a system to manage experiments usually starts as a series of scripts used to run and extract information from experiments. The aptly titled "Scripts for Organizing 'Speriments" (SOS) is a rather large set of scripts whose purpose is to ease the process of running a large number of experiments and working with the resulting data by utilizing a database. A scheduling tool developed at Massey University in James et al. (2007) attempts to make managing experiments easier with the use of a graphical user interface (GUI) driven scheduler. The group mentions extending this interface so that it can be accessed via the World Wide Web (WWW), and mentions James and Hawick (1998) which implemented a very crude form of this. Finally, dealing with experimental design, Camp et al. (2005) and Kurkowski (2006) explain the benefits of different types of experimental design, and use Law and Kelton (2000) as their primary source for information on 2^k factorial simulation. Perrone et al. (2003) tie these ideas up nicely by pointing out the need for good experimental design and a medium where experimental configuration parameters can be made public.

2.1 Automating the Experimental Process

2.1.1 SOS

The first attempt at organizing SWAN experiments was the "Scripts for Organizing 'Speriments" (SOS) project, formerly Scripts for Organizing Simulations. The purpose of the set of scripts is to ease the process of running a large number of simulation experiments and working with the resulting data. Griffin et al. (2002) recognized that building experiment files for every simulation run can be a "painful" task, and organizing the large set of experiments can be even worse. SOS was initially meant to be used with SSF compliant simulators. SSF created a standard for the construction of component based simulators, and describes itself as "a public-domain standard for discrete-event simulation of large, complex systems" (SSF Research Network 2002). SSFNet and SWAN are two of these simulators. SOS was later generalized to work with any set of experiments performed on a computer.

Basically, SOS allows a user to create a specification file with a list of factors and levels for the factors. SOS is then able to compute the cross product of these variables and run a given command for each design point, passing the levels as parameters. The user can specify a script to extract the relevant data from the results produced by the command in the previous step. SOS stores all of this information in a database.

SOS helped me realize the importance of a database when organizing experiments. However, custom extractor scripts have to be written because the tool tries to be generic. Furthermore, I felt that the organization of the database could be better, and that the generalization of SOS made it cumbersome, so I improve upon these points.

2.1.2 Massey University Scheduling Tools

James et al. (2007) wrote about a user-friendly simulation scheduling and management system that they developed to examine the issue of scheduling simulations. The system is used to schedule simulations that involve large and complex multi-scale parameter spaces. The authors wanted to have the ability to steer the simulation toward certain design points once the execution was underway. The scheduler lets a user manually cancel jobs that he deems unnecessary based on partial results obtained during

experiment execution. Thus, the parameter search space can be changed based on the most recent experimental results.

Similar to SOS, the initial solution to managing their experiments was a shell script that executed jobs with parameters drawn from a pre-set range. They later developed a more advanced system using Python which includes the GUI mentioned above. The authors wish to include the ability to use a simple web-services interface to allow the remote creation and steering of experiments. I wish to take this a step further and allow not only for the creation and steering of experiments in the web-based interface, but also incorporate experimental analysis, and "user-friendliness" through error checking and on line help.

James and Hawick (1998) allowed experimental results to be accessed and manipulated over the WWW, but the technology was rudimentary by today's standards. They used a typical web server to invoke Java programs which sent the processed experimental data back to a client who made a request to the server over the WWW. These articles generated ideas about how to provide scheduling tools and experimental analysis to a user of my web-based framework.

2.2 Experimental Design and Credibility

I read quite a few articles about the problems with the credibility of simulation experiments. Camp et al. (2005) and Kurkowski (2006) both wrote about the lack of reliability of MANET simulation studies. Perrone et al. (2003) present some modeling and simulation best practices, which address some of these concerns, and Sanchez (2005) presents an experimental design technique which can also help address credibility concerns. Her paper referred me to Law and Kelton (2000), which helped me study the topic more in depth.

2.2.1 Credibility

As discussed in Section 1.5, Camp et al. (2005) and Kurkowski (2006) address the lack of credibility in simulation experiments. They carefully examined approximately 114 papers from MobiHoc, a premier conference on wireless ad hoc networking, that use simulation to test verious hypotheses. The results they present are quite startling.

Only 34 bothered to identify the simulator used, and none of them stated that the code used was available to others. This means that the research would be difficult to repeat, and therefore lacks credibility. The group stated that the lack of believability in MANET simulation studies is a concern. Among other things, the group suggested that researchers spend more time setting up their experiments, as this is the most often overlooked or rushed step.

Perrone et al. (2003) reiterate these concerns and called attention to the need for adherence to well-established simulation techniques in their paper. Similarly to Camp et al. (2005), the group wrote that it would be unlikely that anyone would be able to repeat an experiment without a detailed description of it, such as the code used and the parameter configurations. As a possible solution, they made this kind of information available on a web page associated with their paper. They also mentioned that it would be helpful to devise tools to help guide the user through the experiment configuration, which I address in my work.

2.2.2 Design

Since both Perrone et al. (2003) and Camp et al. (2005) wrote that experimental design is a crucial step in establishing experimental credibility, I read Sanchez (2005), who extolled the benefits of carefully designed exeriments as well. Sanchez states that a carefully designed experiment allows the analyst to examine many more factors than would otherwise be possible. Both Sanchez and Law and Kelton (2000) wrote about techniques that can cut down the number of design points, and therefore simulation runs, in an experiment. Their techniques do so without sacrificing any information that larger experiments would provide. I discussed one of these techniques, called 2^k factorial design, in Section 1.4.2. 2^k factorial simulation is interesting because it allows a researcher to run fewer simulations, and hopefully still gain the same insight into how a system functions. I construct a framework where 2^k factorial simulations can be easily performed.

Chapter 3

Constructing a Simulation Framework

In the previous chapters I discussed the problems with MANET simulation studies. In particular, Section 1.5 detailed the two crucial downfalls in these studies. Recall that they are a result of poor experimental simulation guidelines, and, similarly, a lack of experimental methodology. I also discussed how poor experimental design was a factor, and introduced a methodological experimental design called 2^k factorial design that introduces a predictable and beneficial structure into the design of a simulation experiment. In this section, I discuss the design of a web-based framework that aims to solve these problems, and simultaneously incorporate good experimental design while helping the researcher work more efficiently.

3.1 Why Use a *Web-Based* Interface?

There is more than one way to write a tool to achieve all the goals outlined above, but the reason that I chose a web-based interface is portability. The interface is great because it centralizes all the logic of the program on one server. This makes the program very portable, because the client is essentially "dumb." The interface can be used anywhere that an Internet connection is present, on any operating system, from any browser.

The requirements to use the interface are met on any computer. The interface runs on a server that is connected to the Internet, and therefore users can log on to it from any computer that is connected to the Internet as well. The operating system of the client is independent of that of the server. The client can use almost any operating system on their computer, including Windows, Mac OS, GNU/Linux, BSD, and more. Finally, the browser that the client uses to directly interact with the server is flexible as well. Firefox runs on all of the operating systems mentioned above, and works well for interacting with the application, although each operating systems' native browser theoretically work equally well.

3.2 Goals of the Framework

There are several goals that must be considered while constructing the web-based interface. The interface is a tool that should aid the researcher in performing simulations. That is, it is supposed to be a helpful tool that enables scientists to work more efficiently. To help the researcher do this, it would be good if the interface automated the typical work flow of a simulation experiment. It should also be easy to use and simply designed, as this also aids in building an efficient simulation process. Furthermore, the interface should incorporate the possibility for 2^k factorial experimental design, as discussed above. Finally, it should be programmed in such a way that it could easily be extended. First, one of the goals of my work was to design this framework in a way that it is easily maintained and extended by other people. I want future students to be able to extend parts of the interface with ease. This encourages further development of the interface.

3.2.1 Automating the Work Flow

Chapter 1.1 describes the simulation process in a general sense, but this section couples the concepts of simulation with SWAN. The process of performing a SWAN simulation usually entails a few steps, and they are very repetitive and prone to error. The first step is the construction of a model, usually mathematical, that approximates the system that we would like to study. The second step is to implement the mathematical model in software so that it can execute on a computer. SWAN is the tool that serves as the implementation of a mathematical model. Thankfully, we only need to design and implement the model once, and we are free to modify

and add on to it later. Sometimes we reuse existing models that have already been validated and verified and to stage experiments.

The repetitive part comes in the next few steps. They are configuring and executing experiments, running the simulator, and storing and analyzing results. As we saw earlier, it is a process that is prone to errors.

Experiment Configuration

Configuring the simulation is the most difficult part of the entire process. In SWAN and SSF based simulators, the way that an experiment is configured is through a file written in a special language called Domain Modeling Language, or DML (SSF Research Network 2002). The syntax of DML is quite straightforward, it consists of a series of keyword and attribute lists; however, it is easy to make mistakes in writing a DML file without some kind of assistance. For instance, it is easy to mismatch a square bracket or make some other small mistake that can mess up an entire experiment. Listing 3.1 shows an excerpt of a DML model description which contains many matching square brackets. One could imagine how easy it would be to omit or mis-match one of these. Quite possibly the most difficult part of configuring a SWAN experiment is the way that the different factors depend upon one another. That is, the level of one factor can dictate what other factors need to be set, as well as the range of levels available for that factor. Setting a factor to some level could mean that an additional section of configuration is required, or it could mean a limitation on the levels of other factors in the model. There is no way to know this information without a detailed understanding of the SWAN simulator. My tool eliminates all of this confusion.

The contents of a DML file are not as important for the thesis as are the concepts behind it. One DML file corresponds to one execution of a SWAN experiment, and, as defined in Chapter 1.4.2, one design point. Also recall from Table 1.1 that more than one design point is needed per experiment. It follows, then, that a researcher performing simulations will need more than one DML file. In fact, he will probably need tens, hundreds, or even thousands of DML files. Managing all of these can be quite unwieldy! Storing the simulation configuration in a database is a better alternative because a database can be easily searched, queried to extract only relevant information, and structured in a way that prevents erroneous values from being stored (such as inserting a floating point number where an integer should be). Furthermore, I dynamically generate the requisite DML files from configured experiments stored in

```
dictionary [
  equipment [
    cabletron [
      portmaster [
        machinespecs [
          _schema ".dictionary.schemas.machinespecs"
          powerconsumption 1600w
          mtbf "20 days"
          replacementcost "$500,000"
        ]
      ]
    ]
  ]
]
```

Listing 3.1: A DML file excerpt.

the database. Doing this makes the life of the researcher much easier, and lets me automate the running of SWAN later on. Finally, storing the configured experiments in the database saves space, file system clutter, and lets the researcher easily share them via the WWW for use in publications.

I also mentioned that the structure of a DML file lends itself to making syntax errors. By this I mean that the configuration requires a complex system of nested brackets, where each component of the SWAN simulator except for the root configuration is encapsulated within another component. Keeping track of what goes where is often difficult, and requires knowledge of the inner workings of the simulator. I found myself picking through the source code of the simulator in order to understand what factors applied to what sections of the simulator. This is quite an inefficient way to configure an experiment. This is another area which can introduce errors into a simulation, which I automate to reduce human error.

Finally, and most substantially, there are dependencies between components of SWAN that are not easily understandable or intuitive to a beginning (or even intermediate) researcher. SWAN uses a collection of models to simulate the mobility of wireless nodes. One such model is called "stationary mobility." Hosts associated with this model remain stationary throughout the simulation. One section of this model that must be configured is the deployment of the hosts in the simulation. Deployment controls how the hosts are distributed on the terrain model at the beginning of the simulation. One possible setting of this factor is "cluster" deployment. Cluster deployment adds hew hosts to the simulation terrain in a group, whose properties

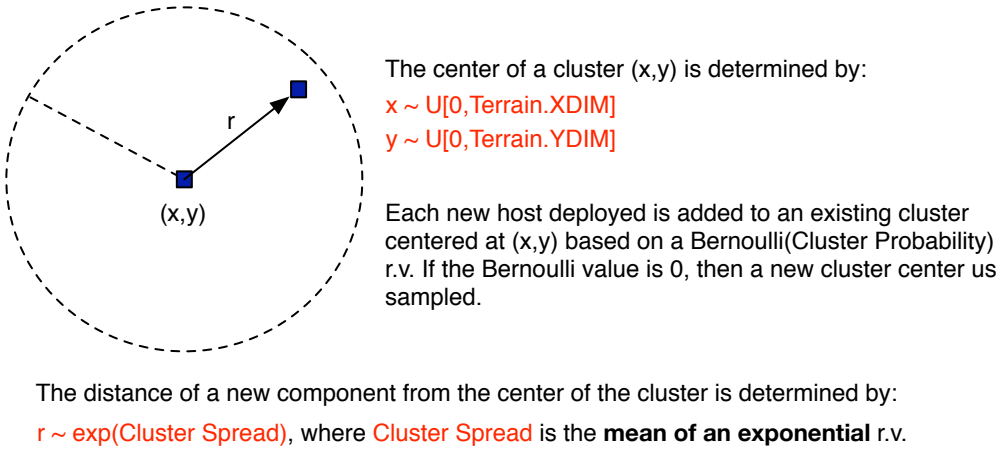

Figure 3.1: Explanation of additional factors to be configured when using a "cluster" deployment.

are determined by still more factors. This type of deployment requires that the user specify additional factors, such as the center of the cluster as an ordered pair, as well as a probability to use as a parameter in a Bernoulli random variable when placing the hosts (see Figure 3.1). Prior to the development of the web-framework, there was no way to know that additional factors must be configured except for detailed knowledge of SWAN. The interface I designed solves this problem by dynamically updating the configuration screens to add or subtract options on the fly.

A web interface makes it easy to implement several other functions in experiment configuration, such as validating ranges of values, and checking number types. Figure 3.2 shows the framework catching an error in a probability. An initial probability of 0.2 incremented by 0.81 one time results in a final probability of $0.2+1(0.81) = 1.01$, which is of course illegal because the end result is greater than one ($1.01 > 1$).

Executing Simulations

Executing simulations is also an automatable task, because we've already stored the configuration in the database. Using the settings for the model's parameters stored in the database, we can extract the relevant information and generate the appropriate simulations. We can then pass the simulation parameters to the simulator and run it, instructing the simulator to save the results to a location that we choose.

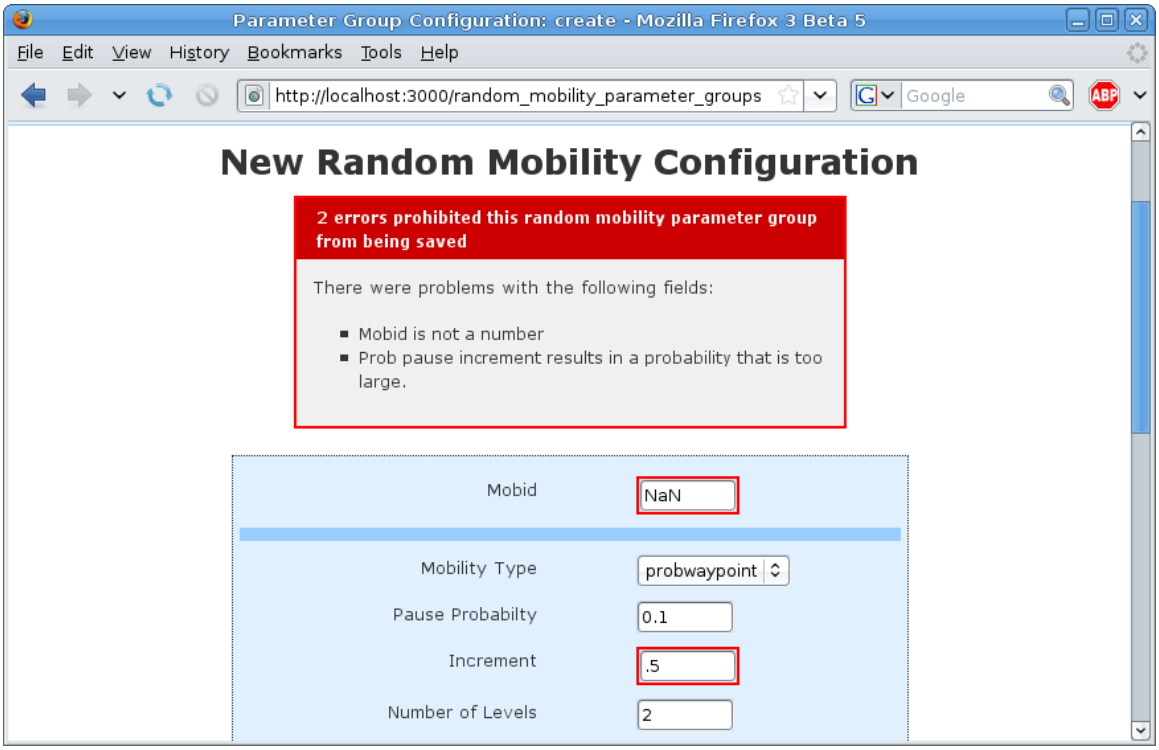

Figure 3.2: An example showing the framework catching an out-of-bounds probability.

Bryan Ward, a member of our research group, introduced the idea of using Distributed Ruby (DRb) to dispatch simulations to various computers in a cluster. Using DRb, we can divide the different design points in the experiment (a single simulation, or one execution of the SWAN simulator) across multiple computers. Distributed execution allows different points of the experiment space to be explored in parallel; this is the main benefit of the DRb approach. It also provides redundancy, in case one of the computers goes down. If one computer goes down, only the part of the simulation executing on that computer is lost. The lost parts of the simulation are then reassigned to another computer to run to completion. Another benefit is that all computers write their results to a networked file system, instead of storing results locally. This provides protection against losing results. Instead of losing days worth of time, we've only lost a few minutes or maybe hours at most.

Storing and Analyzing Results

The results of the experiments can easily be extracted from the simulator's output and stored in the database, just as the configurations of the experiments are stored there. This way, the database will store the configuration of each experiment and its corresponding output data. When one needs to recall the details of the experiment for publication, the database can be queried to provide the desired data. Furthermore, it is possible also to make the entire database available to the research community via the web using a tool like phpMyAdmin, and backups are easy to do with a tool like MySQL-Dump. Scripts can also be developed to mine the database for information and easily generate graphs using utilities such as gnuplot. Statistics can automatically be performed at the end of a simulation, and be stored in the database as well. All of these new features reduce the amount of human error introduced into the system at multiple steps, with an emphasis on preventing inconsistencies in the model configurations.

3.2.2 Ease of Use

One of the purposes of the interface is to eliminate the difficulty of manually configuring simulations via text files. It follows that the interface should be simple and abstract away the confusing configuration files. Pidd (1996) writes that many are dissatisfied with the interfaces to simulation software. The literature presented in Chapter 2 provides additional support for this idea. I believe that simple designs are

better because they do not require a great deal of time or effort to learn how to use. An increase in usability provides an increase credibility, because all of the complexity discussed earlier is abstracted away. Google is a company that profits from providing users with web services. As an example of ease of use, Google presents its search users with only three places to interact on the search page: the search keywords text field, the button to actually perform the search, and the "I'm feeling lucky" button, which directs the user to the first search result. Their search tool requires little or no instruction to understand. I have worked to make this framework equally easy to use.

I designed the pages in a way that lends itself to future customization. Above all, it is important that the interface maintains consistency throughout all its parts so a user does not need to relearn how to interact with different sections of the website. Therefore, I strove for uniformity. For instance, I keep the same navigation bar on each page so that a user will always know where to go to access certain tools. This separates the navigation bars, or "tools," from the actual workspace, and reduces clutter.

The interface also has a simple appearance, without too many options that could bewilder a novice user of wireless network simulation. The steps to progress throughout the simulation process are presented incrementally, with a concise description of what each step does. Hiding the remaining steps will keep researchers focused on the task at hand, and prevent them from making mistakes or clicking aimlessly through configuration parameters that do not yet have any meaning in the current context.

3.2.3 2^k Factorial Simulation

2^k factorial design was explained in depth in Chapter 1.4.2. Recall that 2^k factorial design enables a researcher to cut down on the amount of design points in an experiment by selecting a maximum of two levels for any given factor in a simulation. Also recall, from Chapter 3.2.1, that even a small number of design points can be a hassle to manage because of all of the configuration files that SWAN requires. Since our research group would like to examine how useful a 2^k factorial design would be for SWAN simulations, the framework had better be able to configure simulations that support this, and do it easily. I support this by letting a user select an initial level for a factor, followed by choosing an increment value for that factor and the number of levels the factor should have. This lets us support 2^k factorial design, as well as other experimental designs.

3.2.4 Design and Extensibility

Finally, I write the framework in a way so that it can be easily modified and extended to change its behavior or add features at a later date. There are two motivations for doing this. First, I want someone else to be able to implement additional features in the interface without having to dig through complex code. Second, I want to encourage further development of the interface after I am gone, or at least have the *idea* of a well-designed interface survive. Ruby on Rails is a free web application framework that uses the Ruby programming language to make web development fast, simple, and efficient. It is a good language to program the interface in because of this simplicity. This simplicity stems from several philosophies that the developers of the framework subscribe to. There are some silly names attached to these ideas, although they are by no means simplistic, trivial things to do. Two that I find to be most important are "Keep It Simple, Stupid" (KISS), and "Don't Repeat Yourself" (DRY). I think the names of these two principles are self explanatory. In the next two sections, I explain how they fit into what I am doing.

Keep It Simple, Stupid (KISS) Principle

The "Keep It Simple, Stupid" (KISS) principle is really nothing more elaborate that it sounds. I think that it is useful in both coding the application, and, as mentioned in Chapter 3.2.2 the design of the interface. I have already explained how the user can benefit from a simple design, but potential maintainers of this project can benefit from well written code. Ruby code is very easy to read, and Ruby on Rails developers have taken this a step further by making Rails applications even easier to understand. Since this is not a software engineering thesis, I only provide one example, but I think it illustrates my point well.

Listing 3.2 is an example of a piece of simple, readable code used in part of the application that checks to see if part of the user's configuration of SWAN is valid. Notice that it reads almost like an English sentence. From left to right, we want to validate the "numericaliy" (Rails' way of testing is something is a number) of a probability (here it's actually the same probability I discussed in 3.2.1, and shown in Figure 3.1). This simply means we want to make sure the user types in a number. Continuing, it should be greater than zero, and less than one. Finally, we are only going to do this if the deployment type is set to "cluster." There is a lot going on behind the scenes here, but on the surface, I it is simple and readable, and will help

```
validates_numericality_of :cluster_prob, :greater_than => 0,
        :less_than => 1, :if => :deployment_is_cluster
```

Listing 3.2: An example of some simple and readable code.

anyone maintaining this application in the future.

Don't Repeat Yourself (DRY) Principle

The "Don't Repeat Yourself" (DRY) principle is also quite easy to grasp. DRY simply means that information should not be duplicated, especially in code. There are various reasons for this. One is that any change in the logic of a program could result in the need to change multiple areas of code to update the logic. Writing the code without duplicate sections eliminates this problem, ensures that changing one part of the code does not unknowingly affect other areas of the code, and that related elements all change predictably.

Listing 3.3 shows how Rails makes it easy to add functions that are useful to the programmer of the framework. Rather than copying and pasting the lines of code that validate the "numericality" of increment values, and the number of increments to perform, here I augment the base class provided by Rails called `ActiveRecord` to add in a custom function called `validates_increment_and_numvals`. This function can be called from any model that needs these variables validated, rather than putting code similar to that contained in Listing 3.3 in every model that needs it.

```
module ActiveRecord
  module Validations
    module ClassMethods
      def validates_increment_and_numvals
        self.column_names.each do |col|
          unless /increment$/.match(col).nil?
            validates_numericality_of col.intern,
             :greater_than => -1
          end
          unless /numvals$/.match(col).nil?
            validates_numericality_of col.intern,
              :greater_than => 0,
              :less_than => self.numvals_max
          end
# Additional end tags removed in the interest of saving space.
```

Listing 3.3: Preventing repeated code by augmenting the core Rails classes.

Chapter 4

Case Study: Why Is This Tool Worthwhile?

In this chapter of the paper, I use an article by Camp et al. (2005) to demonstrate how the use of my interface enhances the credibility of SWAN simulation studies. This article describes several downfalls that undermine the credibility of these studies. I discuss the downfalls that the article points out, and show how I addressed these points while constructing the interface.

4.1 Work Flow

The configuration of an experiment consists of four steps, which I elaborate on below. They are:

1. Set up factors and their levels.
2. Validate the user's input.
3. Store the configuration in the database.
4. Generate the simulations.

The initial stages of setting up an experiment are shown in Figure 4.1. The first step in the process is configuring the simulation. The model configuration and construction stage prompts the user for the levels of various factors in the simulation. The responses to these requests are saved in the database after they pass some tests to ensure they are valid. The interface then generates each design point in the experiment, and saves these back to the database in the experiment definition step. Next, the interface runs the experiment across a small cluster of eight computers.

Running the simulation across the cluster of computers is a bit more involved than simply running the simulation locally. The process is illustrated in Figure 4.2. To distribute the execution of experiments, we execute the algorithm below. Steps three to six are not necessarily executed in the order presented, because we typically have multiple clients interleaving their connections to the server in unpredictable ways.

1. Start an experiment distribution server which contains two queues. One has simulations waiting to run, let's call it queue A, and the other has simulations which are currently in progress, queue B.

2. Start clients on each of the computers we wish to run simulations for us.

3. Clients connect to the server and request work to do. The server sends the client a simulation (or design point), which the client executes. The server moves this simulation from queue A to queue B.

4. Clients execute the simulation. Upon completion, they send the location of the results file, as well as the complete configuration (DML) file back to the server. The results file is accessible to the server because they share a common, network mounted file system; however, if we wanted to have clients across the globe, we could simply transmit the entire results file back to the server, as is done for the configuration file.

5. The server receives this message from the client. If the simulation is in the queue B, the server deletes the simulation from the queue, extracts the results from the appropriate file, and saves the results and simulation configuration to the database.

6. Steps three to six repeat until there are no more simulations in queue A. If there are no simulations in this queue, the server dispatches those in queue B to the clients. This is done so the server does not wait forever in case a client that requested a simulation goes down.

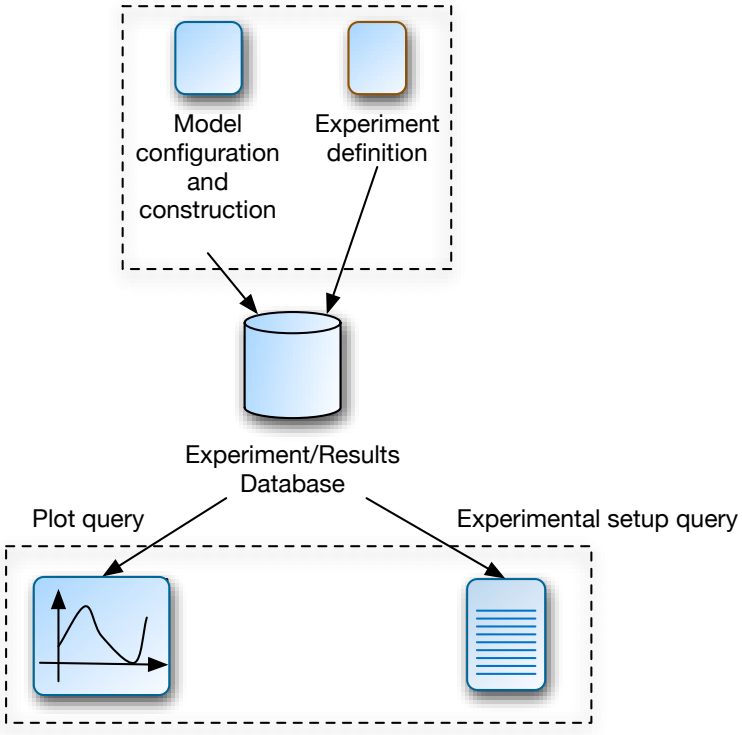

Figure 4.1: Simulation work flow using the web-based interface.

7. When queue B is empty as well, we know that all simulations have executed, and that all results are in the database, so the server and all clients exit.

Now that the results are stored, the user can use the interface to perform a number of useful tasks, such as viewing a complete configuration file for any one design point, and obtaining statistics in a format compatible with a variety of tools such as gnuplot. In Chapter 5, I discuss how features such as the automatic plotting of results can also be incorporated. In the following sections, I elaborate upon how each of these steps makes simulation more credible.

4.2 Simulation Setup

Section III.A of Camp et al. (2005) discusses simulation setup. They mention how this is the phase that is most often skipped or overlooked, which is dangerous because

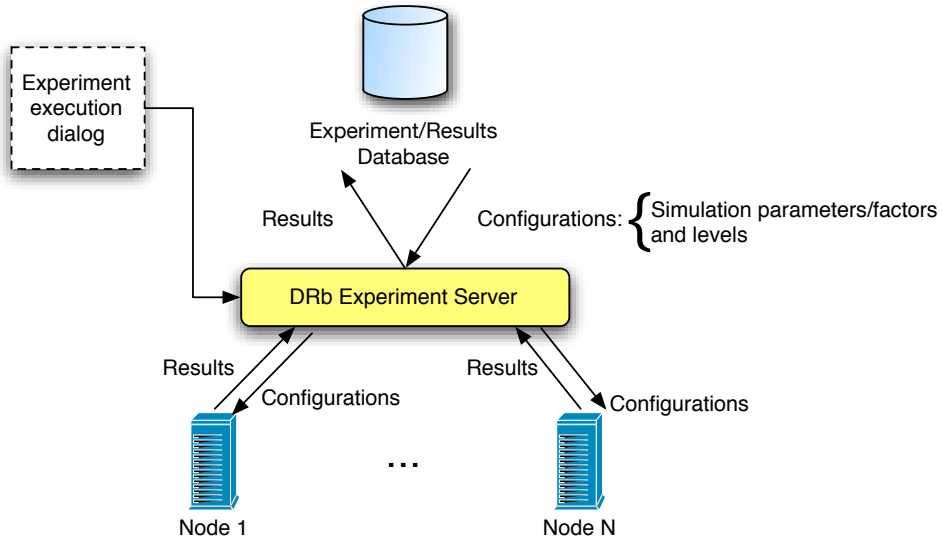

Figure 4.2: Distributed execution of SWAN experiments.

then every subsequent step in the simulation process is unreliable. The authors argue that variable definition, or determining the levels of the factors, is often a problem in simulation studies. The large number of variables that must be configured to run a simulation study often makes it difficult to determine the default settings of these variables. Camp et al. note that researchers will sometimes leave key parameters undefined.

To remedy this problem, I validate each variable that the user sets in the web interface. Each submodule configuration is checked for mutually exclusive and dependent factors. That is, the configuration options will appear or disappear as the configuration of the submodule progresses. Figure 4.3 shows a screen that uses a drop-down list to prompt the user for a "Deployment" type. Upon selecting a deployment type, the screen is updated with additional parameters that need to be configured based on the user's choice. This ensures that all the necessary parameters are set, without relying on the dangerous default values that Camp et al. mention in their paper. In case the user does not know what a particular factor does, I provide help links that he can click on. Clicking the link opens a pop up link with documentation specific to that factor. The documentation for the "Deployment" help link is shown in Figure 4.4.

In addition to checking for the errors above, I also make sure that the user enters sensible values at the prompts. These errors include type errors, such as entering

Figure 4.3: Configuration page with help link.

a floating point number or text where only an integer should go, and checking that the numbers do not overrun boundaries, such as a probability that exceeds one. See Figure 4.5 for a screen shot of this error checking in action. Finally, Perrone et al. (2003), express the need for a medium where experiment configurations can be permanently stored, as there simply is not enough space to publish this information in journals and conference in-proceedings. Since this is a web based interface, simply including a short URL in a publication is enough to direct a reader to the complete experiment configuration. In this section, I have already implemented solutions for a number of problems that Camp et al. and Perrone et al. presented.

One final point that adds to the credibility of simulation experiments using my tool is the consideration I gave to experimental design (or design of experiment, DOE) when constructing it. The ability to produce experiments in 2^k factorial DOE, as well as other DOE styles, lets the researcher use proven DOE methodologies to add believability to her simulations.

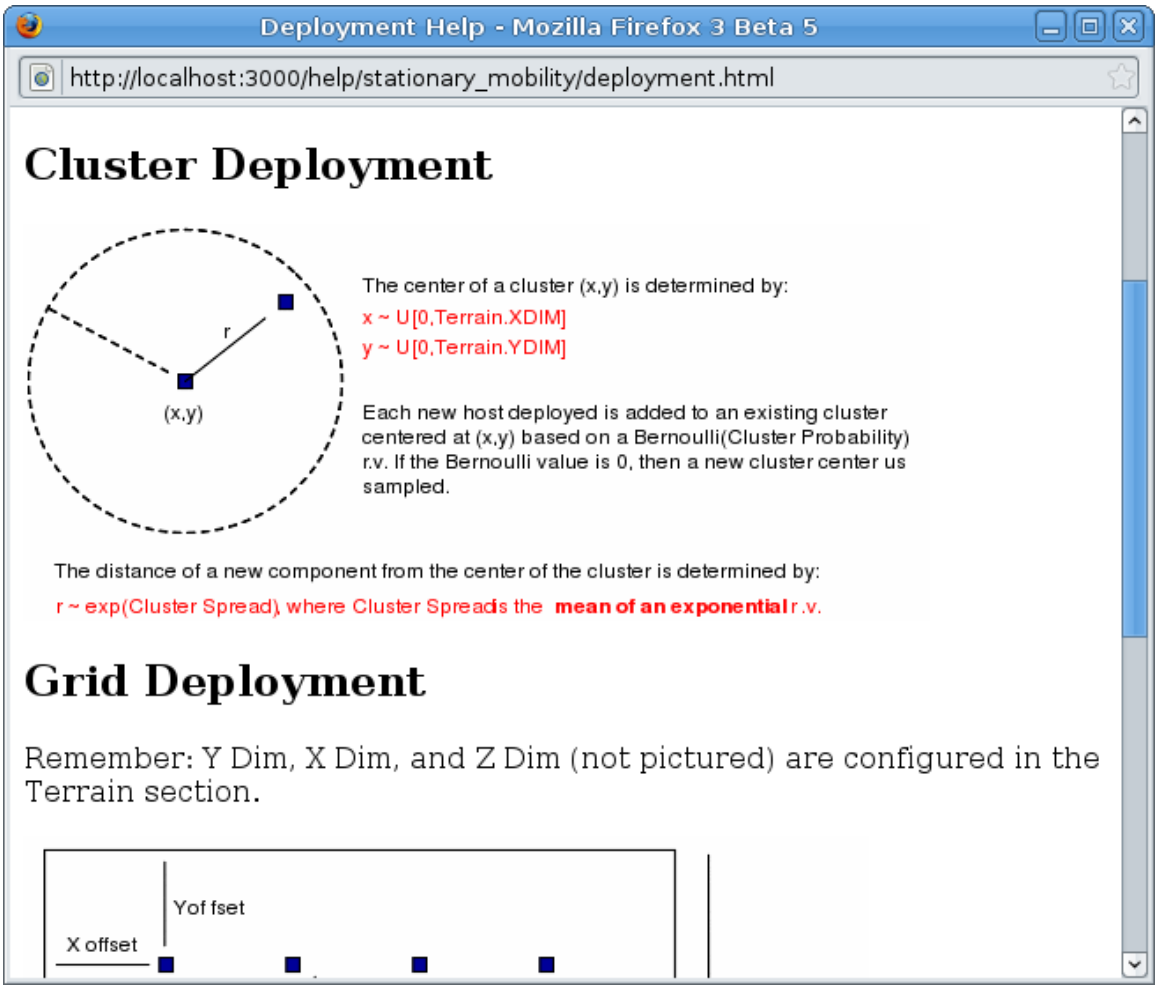

Figure 4.4: Convenient documentation pop-up window.

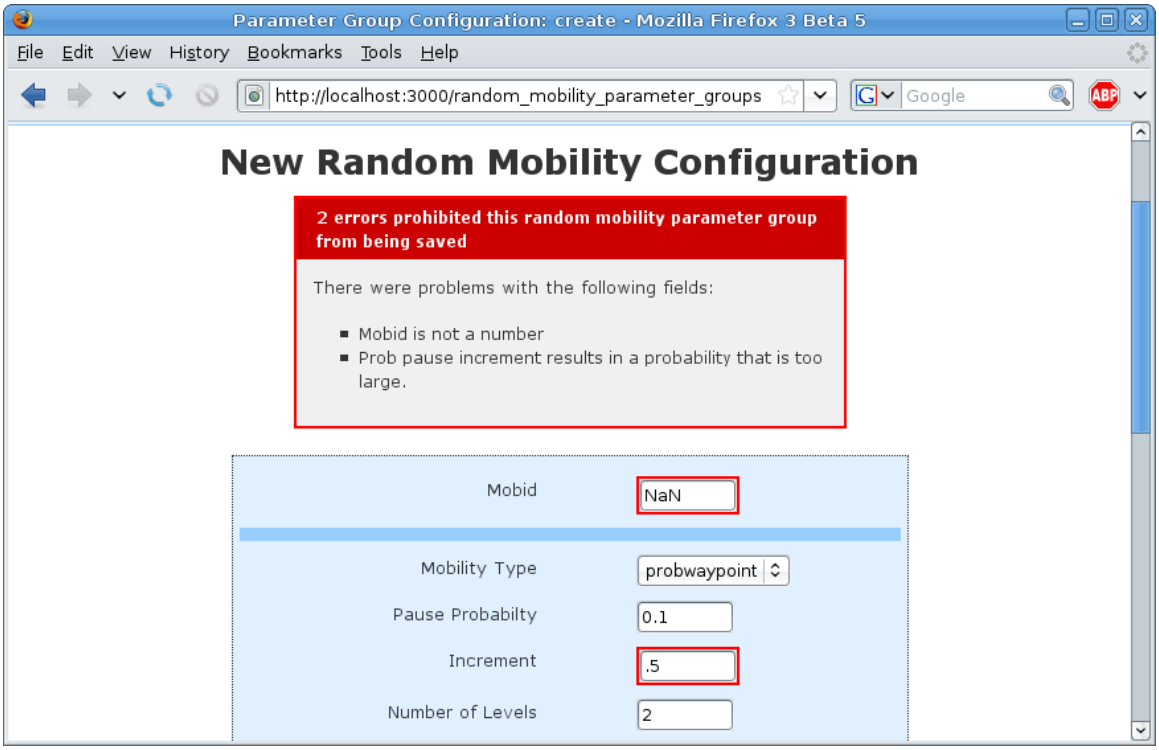

Figure 4.5: Error checking example.

4.3 Setting the PRNG Seed

Camp et al. (2005) devote an entire section of their paper (III.B.1) to the importance of setting the Pseudo-Random Number Generator's (PRNG's) seed. The group notes that if the user does not set the seed, the simulator will produce identical results each time it is run on a design point. This creates a false confidence in the statistics because they are all the same, and discredits the simulation. I remedy this problem and restore some credibility to the simulation by requiring the user to set an initial seed when he creates an experiment, ensuring that the default value is not used each time.

4.4 Output Analysis

Three sections of Camp et al.'s paper discuss output analysis. The authors claim that because many researchers spend so much time fixing the other problems I mentioned, output analysis is often rushed, and therefore the Achilles heel of many simulation studies. Researchers should have more time to do output analysis because of the time they will save by using my tool; however, I still managed to automate most (if not all) of the output analysis to reduce human error, and boost experiment credibility.

4.4.1 Single Set of Data

The authors write that taking the first set of data from a simulation and accepting the results as truth is not a believable way to conduct research. Obviously, a single result is rarely, if ever, representative of the population statistics. Using only a single run to calculate statistics on does not account for randomness in the simulator and does not provide confidence in statistical means.

Fixing this problem is quite straightforward. Multiple runs of the simulation must be performed, and, of course, the PRNG seed must be set correctly. The credibility restoring step here is that the web-interface, not the researcher, ensures that each simulation is run multiple times. The number of times to run each simulation is easily changeable. As for setting the PRNG seed correctly, I already addressed this issue in Section 4.3.

4.4.2 Statistical Analysis

Camp et al. say that using incorrect statistical formulas are another source of error in simulation studies. The group points out that statistical methods will change depending upon data type. Using standard statistical formulas for mean and variance without ensuring that the data is independent and identically distributed is a common pitfall. I ensure that statistics are correctly calculated by automating the processing of results. This eliminates human error in doing these calculations. Additionally, I ensure that the data is independent and identically distributed by performing independent replications of the data (see Section 4.4.1.

While thinking about the results processing, I realized that humans may make mistakes while transferring results to other analysis tools like spreadsheets or plotters. Figure 4.6 shows the web-interface displaying a table of statistics extracted from the result data of a simulation. There are some very long floating point numbers, up to ten digits in length, that are difficult to transcribe or even copy and paste to a desired format. I provide download links for this table in a variety of popular formats to reduce these types of errors.

4.4.3 Confidence Intervals

Confidence intervals are essential when establishing statistical significance; however, Camp et al. (2005) found that 98 of the 112 papers they surveyed in for their article failed to include them. Confidence intervals allow the researcher to say that the population mean lies between a set of two numbers with a certain percentage of confidence. They are necessary because they account for the randomness in the simulator. Using my tools, a researcher is able to calculate confidence intervals because I store all experimental results in the database, and ensure that the data is independent and identically distributed. All it takes is a function that queries the database, retrieves the samples, and produces a confidence interval for a predetermined confidence level. This can be easily implemented in almost any programming language, including Ruby.

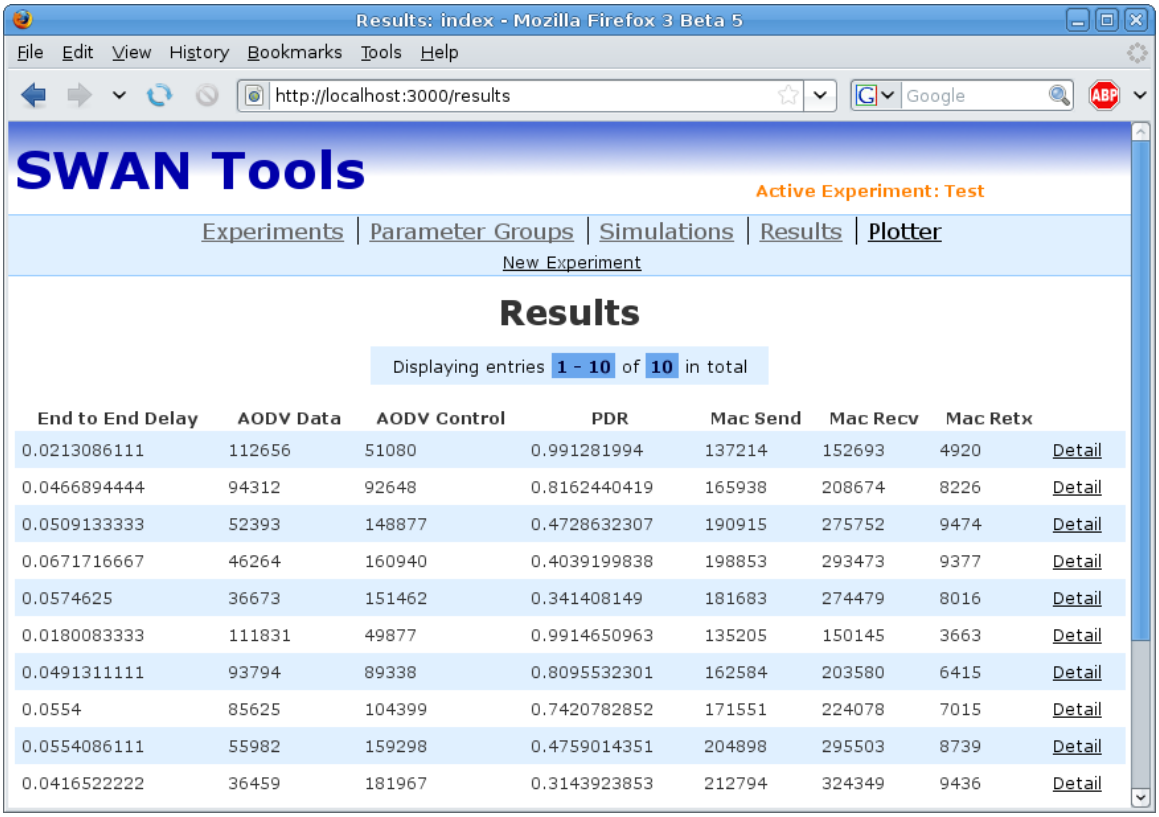

Figure 4.6: Automatically generated, error free statistics.

4.5 Publishing and Experiment Reproducibility

Camp et al. point out three problems with the publication of simulation studies. First, the trustworthiness of the studies is compromised because of inconsistencies in the publication of results. Second, published results are not directly comparable to one another, which they argue limits research advancements. Finally, simulations are not easily repeated, which is another large hindrance to research advancements.

While I cannot impose a standard for the publication of results, I can address the last two points. By saving all experiment configurations and results in a database and providing the web-interface to this data, my tool provides an easy way to access the configuration of our SWAN experiments. As first suggested by Perrone et al. (2003), an author can provide a link to the particular experiment she is writing about so that both the configuration of the experiment and the results are available to readers.

As far as experiment repeatability goes, the web-interface provides two different and convenient ways for researchers to reproduce experiments. For case one, consider Section 4.1 where I mentioned that the simulations are executed by clients that connect and request a design point (simulation parameters) from an experiment distribution server. It would be relatively easy to develop a "read-only" experiment distribution server that runs a custom protocol to serve specific experiment request. Then, a client could connect to our server and request a particular design point (or group of design points) to simulate. As long as they had the SWAN simulator (which is open source) there would be absolutely no configuration done by the end-user.

Case two is similar to case one, except that it requires the researcher wishing to reproduce the simulations to manually fetch the configurations. Referencing Section 4.1 once again, recall that the DML configuration file for each experiment is stored in the database. Figure 4.7 shows some experiments in the database with a link to the DML file for each one on the far right. Clicking on the DML link sends the DML file to the user's browser as a downloadable text file, as in Figure 4.8.

Both of these cases demonstrate that reproducing an experiment configured using the interface nearly trivial. Access to the configuration files is readily available so others can verify the results of the experiment. This is a huge step in establishing experiment credibility.

This section examined six different flaws that Camp et al. (2005) found with a survey of 112 different publications, from 2000 – 2005. In this section, I introduced

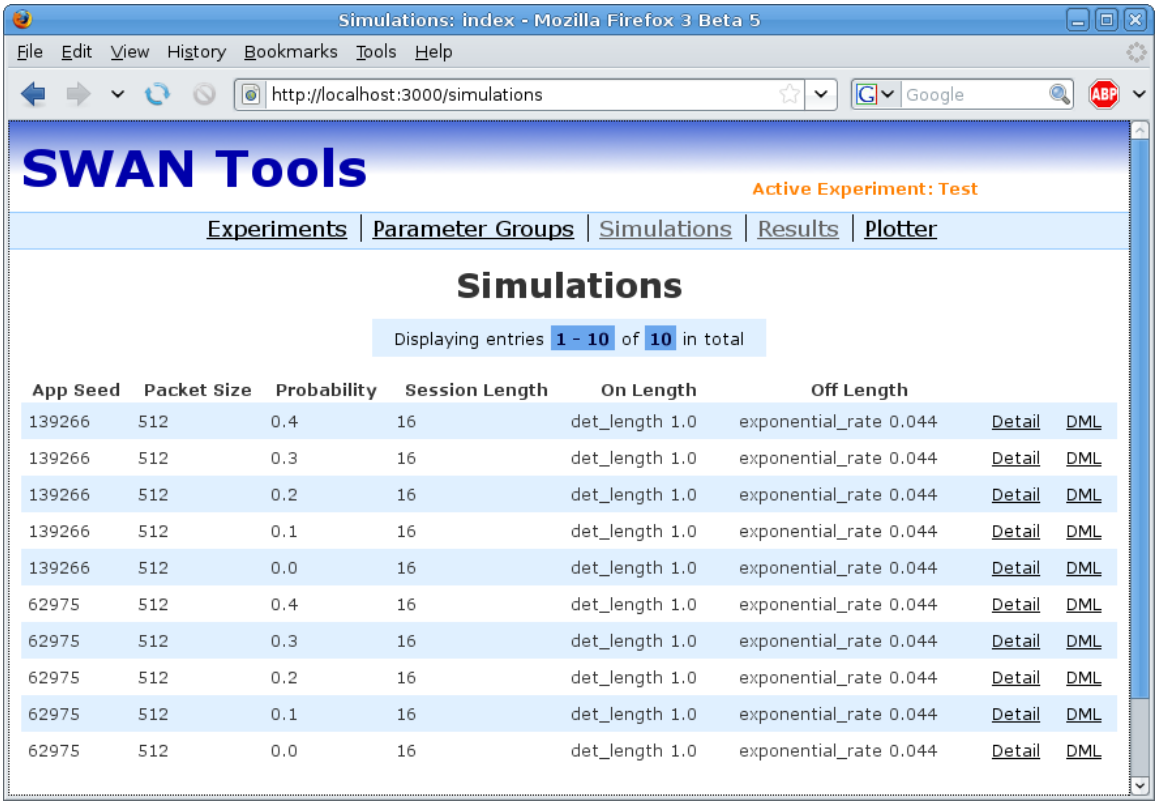

Figure 4.7: DML Download and overview of configured simulations on the simulations page.

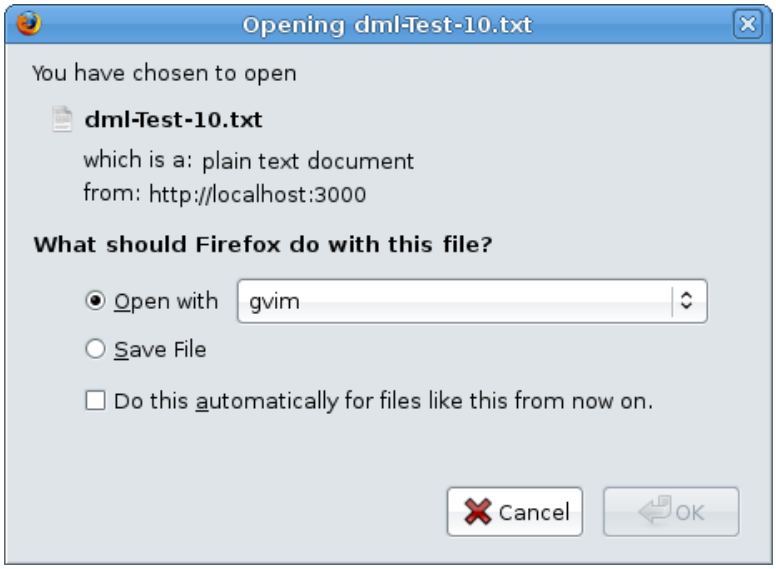

Figure 4.8: Clicking the DML link to download DML configuration file.

each of these, and commented on how my tool addresses each of them. By carefully validating user input, I am able to ensure that the simulation is correctly configured. One very important step emphasized by Camp et al. is to make sure the PRNG seed is set for each simulation, which I do. Concerning output analysis, my tool provides automated and accurate statistical analysis on data from multiple runs of the simulator. This lets the researcher provide confidence intervals for his statistics, which is very important when producing publications. Finally, the interface makes it easy for others to repeat simulations by storing the experiments' DML configuration files in the database. Those wishing to repeat experiments can simply download them, and in the future maybe even connect using a custom client to automate the process of fetching configurations and running simulations.

Chapter 5

Conclusion and Future Work

The web-interface I created adds reasonable, error preventing constraints on what a researcher can do when configuring and running a SWAN simulation. Because the interface guides the researcher in making sensible choices, validates all choices, and automates many error prone steps, SWAN simulations are made more credible. Restating some of the highlights of Chapter 4, I focused on an article by Camp et al. (2005) which outlined numerous flaws with simulation studies. I constrained the researcher's options and validated what he entered to check that the simulation was correctly set up. The simulation runs distributed across a cluster of computers, which we can capitalize on to provide easily reproducible experiments. Once this is done, the interface provides automated and accurate statistical analysis on data returned by the simulator. This lets the researcher provide confidence intervals for his statistics, which is very important when conveying the significance of the research. In these publications, the researcher can simply add a link to the location of the complete experiment configuration and results. Each of these points add credibility to specific areas of SWAN simulation experiments. I believe that all of them together constitute a significant shift towards more credible experiments.

There are a number of areas that I feel could be extended, and features that can be added to the interface. For one, the documentation that accompanies the experiment configuration can be expanded. The idea was even thrown about that the help pages could be automatically generated from the comments in the C source code of the simulator. Not only could the documentation be expanded, but more of the submodules that the user configures to create a SWAN experiment can be added to the interface. Moreover, there currently are areas of the DML file that are

not changeable via the web interface. As a result, the factors in these areas are not configurable through the interface. Finally, I would like to see a tool developed that can validate model parameters across submodules.

Some minor changes and improvements would be a customizable experiment result viewer. I imagine a list of check boxes for each factor in the database that, when checked, would make the corresponding data column appear on the results page. Also, the addition of more downloadable data formats could be added with relative ease. Related to experiment results is the topic of visualization or plots. A plotting utility that interactively generates gnuplot style graphs like those used in Perrone et al. (2003) would be a challenging, yet realistic endeavor.

Finally, and perhaps the most interesting implication of this research, is the potential to write a client and server that interact to fetch and execute simulations from the database automatically, for the purpose of demonstrating experiment reproducibility. As discussed in Section 4.1, we currently coordinate the execution of SWAN experiments across multiple hosts using a distributed client and server. At the time of this writing, the client and server only spawn and run to calculate results for a single experiment. I envision a server that is able to accept connections from a client, and send it the configuration for either an entire experiment or discrete simulations based on commands issued by the client. The client can then use this information to run its own SWAN simulation(s), and the results generated by the client can be compared to those published. This would do wonders for experiment reproducibility, and increase experiment credibility even more.

Bibliography

Camp, T., S. Kurkowski, and M. Colagrosso (2005). MANET simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev. 9*(4), 50–61.

Griffin, T. G., S. Petrovic, A. Poplawski, and B. Premore (2002). SOS: Scripts for organizing 'speriments readme. Available via <www.ssfnet.org/sos/README>. [Accessed: Mar 2008.].

James, H. A. and K. A. Hawick (1998, Feb). A web-based interface for on-demand processing of satellite imagery archives. In C. McDonald (Ed.), *Computer Science '98: Proceedings of the 21st Australian Computer Science Conference (ACSC'98)*. Springer.

James, H. A., K. A. Hawick, and C. J. Scogings (2007). User-friendly scheduling tools for large-scale simulation experiments. In *Proceedings of the 2007 Winter Simulation Conference*, pp. 610–616. IEEE.

Kurkowski, S. H. (2006, October). *Credible Mobile Ad Hoc Network Simulation-Based Studies*. Ph. D. thesis, Colorado School of Mines.

Law, A. M. and W. D. Kelton (2000). *Simulation Modeling and Analysis* (3rd ed.). McGraw-Hill.

Merriam-Webster (2008). Scientific method. Available via <www.merriam-webster.com/dictionary/scientific method>. [Accessed Feb 2008.].

Perrone, L. F., Y. Yuan, and D. M. Nicol (2003). Modeling and simulation best practices for wireless ad hoc networks. In *Proceedings of the 2003 Winter Simulation Conference*, pp. 685–693. IEEE.

Pidd, M. (1996). Model development and HCI. In *Proceedings of the 1996 Winter Simulation Conference*, Washington, DC, USA, pp. 681–686. IEEE.

Sanchez, S. M. (2005). Work smarter, not harder: Guidelines for designing simulation experiments. In *Proceedings of the 2005 Winter Simulation Conference*, pp. 69–82. IEEE.

SSF Research Network (1999 – 2002). Scalable simulation framework. Available via <www.ssfnet.org>. [Accessed Mar 2008.].