Hercules: An Environment for Large-Scale Enterprise Infrastructure Testing

Lan Wang, Charles Ellis, Wei Yin, Dung Dinh Luong
{lanwang, ceellis, weiyin, dluong}@memphis.edu
Department of Computer Science
The University of Memphis
Memphis, TN 38152

Abstract

Failures in Internet-based services often stem from problems in the underlying enterprise infrastructures that provide the services. We propose an environment called Hercules to test the reliability and performance of large-scale enterprise infrastructures. Hercules contains two major components: (1) a methodology to build a virtual testbed that can accurately emulate any infrastructure topology, as well as simulate failures, attacks and other types of stresses on the infrastructure to identify defects and bottlenecks; and (2) a realistic traffic model and a tool to automatically generate different traffic loads based on the model. Systems testers can use Hercules to evaluate whether an existing or proposed enterprise infrastructure provides adequate support to its targeted applications.

1 Introduction

Today's companies and organizations are increasingly dependent on the success of the distributed online applications that they deploy. These applications provide a multitude of functionality, ranging from delivering products and services directly to customers to facilitating internal communication. Given the importance of these applications, they usually undergo rigorous testing before their deployment. However, they are only one component of the big picture. If the underlying infrastructure (e.g. the application server) is unavailable, users will not be able to access the desired services provided by these applications no matter how robust the applications are.

What infrastructure support do enterprise applications need? In our view, they need support from at least four categories of infrastructure components: *hardware equipment*, *operating systems*, *middleware*, and *network connectivity*. Typical hardware equipment on enterprise networks includes servers, workstations, load balancers, switches, routers, and firewalls. Operating systems run on some of the hardware equipment, e.g. servers and routers. Middleware includes the non-OS software between the applications and hardware, such as application containers and messaging service. Network connectivity among the hardware equipment supports the communication between end users and applications. Figure 1 shows a typical enterprise infrastructure.

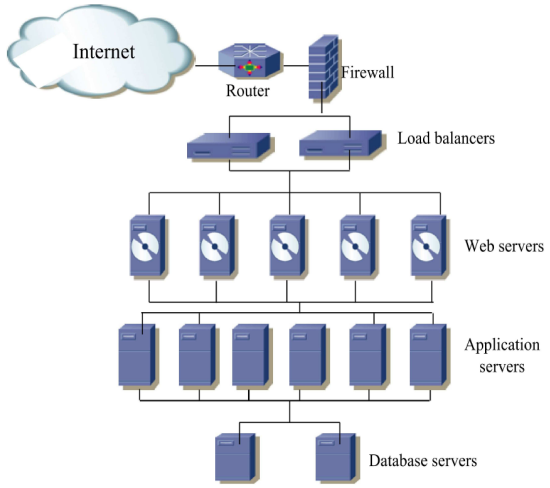

Figure 1. A typical enterprise infrastructure

Given the complexity of large-scale enterprise infrastructures and the interdependencies between components, infrastructure failures could occur during the deployment and operation of an application. In fact, many failures in the delivery of online services stem from the issues of the underlying infrastructure such as server failures and configuration errors. Therefore, the main objective of infrastructure testing is to **uncover a particular infrastructure's reliability and performance issues** in order to ensure that the infrastructure will provide *adequate support* to the applications.

In practice, testbeds usually have a much smaller scale and complexity than the deployed infrastructure due to the cost of setting up and managing the testbed. Moreover, the testing traffic does not reflect the diversity and burstiness

of user behaviors as most existing testing tools simply replay some pre-recorded sequences of user actions on different client machines. To address the deficiencies of existing infrastructure testing tools, our project aims to *construct a prototype testing environment and develop the associated tools* to evaluate the reliability and performance of large-scale enterprise infrastructures.

Our prototype, Hercules, has two main components: (1) a methodology to build a virtual testbed that can accurately emulate any infrastructure topology and simulate failures, attacks and other types of stresses on the infrastructure to identify defects and bottlenecks; and (2) a realistic traffic model and a tool to automatically generate different traffic loads based on the model. We are taking the following steps to build Hercules. First, we are evaluating existing networking testbeds such as Emulab in their ability to emulate large-scale enterprise infrastructures. Second, using web traffic as a case study, we are studying how to accurately model application-layer traffic load including both users' traffic and servers' response traffic for enterprise infrastructure testing.

Our work is still in its early stages. Therefore, this paper serves as an overview of our project. We first identify the requirements for the testing environment in Section 2. Then we describe the design of Hercules and the associated tools in Section 3. We present some preliminary results in Section 4. We review the related work in Section 5 and conclude our paper in Section 6.

2 Requirements

When considering upgrading or replacing an existing enterprise infrastructure, decision makers often need to know whether the proposed infrastructure could provide the desired functionality and meet the performance expectations (under different traffic loads and failure conditions). To support that objective, the testing environment needs to meet the following basic requirements.

First, the testbed should accurately reflect the functionality, capacity and complexity of a particular infrastructure, so that testers will not over-estimate or under-estimate the capability of the infrastructure. It may be desirable to have a testbed at a scale close to the infrastructure under consideration, but the testing methodology should not require such a large testbed to produce meaningful results.

Second, tools should be provided to simulate stressful internal and external conditions in order to discover failures in the infrastructure as well as to evaluate the system performance under certain stressful scenarios. Examples of stressful internal conditions include configuration errors, insufficient memory, unreliable network connectivity and intentional shutdown of specific hardware or software components. Stressful external conditions include, but are not limited to, malformed messages, extremely high traffic loads and malicious software propagation (e.g. virus and worm outbreaks).

Third, in order to avoid vastly over-provisioning an infrastructure, the testing tools should be able to generate realistic traffic loads on the infrastructure under testing. This is actually much more difficult than simply generating high traffic loads, because it requires a deeper understanding of users' actions and an appropriate stochastic model to capture the most important features in user behavior.

3 Hercules

In this section, we sketch out the design of our Hercules testing environment. We first present our methodology to emulate distributed enterprise infrastructures. We then introduce our user session-level model for generating realistic traffic loads. We end this section by describing how Hercules can be used in typical testing scenarios.

We would like to emphasize that the goal of our work is not to build a physical testbed, but rather to develop the testing methodology and tools. We are not testing applications either, so we will provide a library of benchmark applications, each focusing on one or a few specific functions. These application should be simple and already tested thoroughly.

3.1 Emulating Enterprise Infrastructures

In our project, we need to develop the methodology to construct testbeds that accurately reflect the complexity and functionality of real enterprise infrastructures. The networking research community has been facing an even larger problem – how to simulate the Internet [3]. Detailed packet-level simulation in software, e.g. using the NS-2 simulator [1], is only suitable for small network topologies with a few tens of nodes. On the other hand, higher-level software simulations may not reveal the subtle behavior of network protocols. Fortunately, two new techniques, *network emulation* [14] and *network overlay* [11], are enabling networking researchers to run their experiments on real network testbeds that contain hundreds or even thousands of physical machines with a wide range of link speeds and delays. Figure 2 shows how a virtual testbed can be built over a physical testbed using network emulation.

While both network emulation and network overlay allow us to simulate any given network topology containing an enterprise infrastructure and their remote users, we consider network emulation a better choice for infrastructure testing mainly for two reasons. First, with network emulation, we can control precisely the speed and delay of each link in the topology. Second, an emulated network is not subject to interference from external traffic, so the results

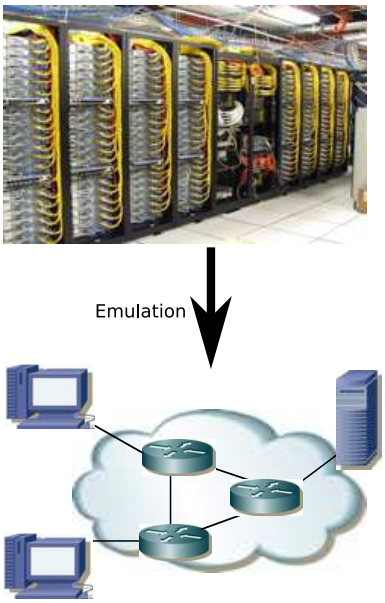

Figure 2. Building a virtual testbed using network emulation: the top picture shows the physical testbed at University of Utah's Emulab (source: http://www.emulab.net/), and the bottom figure shows a virtual testbed that can be configured over the physical testbed.

obtained from network emulation are reproducible, while those from overlay networks are not. Both are important factors for accurately evaluating an enterprise infrastructure. For more information about the pros and cons of these two techniques, please refer to Section 5.

On the other hand, the current Emulab software for *end-users* are not particularly suitable for systems testing, as its users are mostly networking and operating systems researchers. Therefore, one of our contributions will be to *develop tools that allow testers to specify their infrastructure at a high level and automatically create OS images and experiments for them*. Furthermore, we will investigate how to simulate infrastructure failure conditions, such as configuration errors, software failures, hardware shutdowns and unreliable links. It may require us to add new capabilities to the *core* Emulab software.

Note that existing research-oriented emulation testbeds may not provide all the hardware equipments used by a particular enterprise infrastructure. Therefore, we expect that an enterprise interested in adopting our testing methodology and tools will build their own emulation testbed. In fact, a dozen universities and companies have set up their own "Emulabs" using their own hardware with the software developed by the U. Utah's Flux research group.

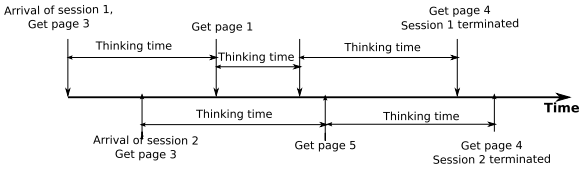

Figure 3. Session-level model for generating user traffic loads.

3.2 Modeling and Generating User Traffic

The second component of Hercules is a session-level model for generating user traffic loads (see Figure 3). Our model includes several features that accurately simulate actual user activities during their interactions with an enterprise application. We will develop a tool called *SessionSim* based on this model. Below we describe our model and the design of *SessionSim*.

First, we model the arrivals of new user sessions using appropriate stochastic processes. We consider a user session to correspond to an active period of communication between a user and a server through a particular type of application. For example, when a user starts to browse a website, a new web session is started. Note that there may be many active user sessions for a server at the same time. We will include several commonly used session arrival models in our tool *SessionSim*. For example, Paxon and Floyd showed that Telnet session arrivals follow a time-varying Poisson process [10]. Furthermore, the session arrival model can be estimated empirically using actual logs obtained from servers.

Second, within each active user session, we model the delay between a user's request and the receipt of the previous response. We call this delay "thinking time". Typically, the thinking time depends on the specific response that a user gets from the server, for example, the user may need more time to read longer text than would others. We can also estimate the distribution of the "thinking time" using server logs. In general, we expect that use of the empirical models would lead to more realistic traffic loads. Therefore, we will develop techniques to derive empirical models and parameters wherever possible based on server logs and measurements.

Third, we model application-specific user requests and responses. For example, in a web session, the users should request for web pages and follow hyperlinks in the web pages. Section 4.3 provides more details about how to use a webgraph to simulate a user browsing through a web site.

Our *SessionSim* includes a controller and different types of clients (see Figure 4). Each client simulates a particular type of user sessions, e.g. a web client simulates web sessions. In order to generate realistic traffic that reflects traffic from a diverse group of users, we will distribute the client

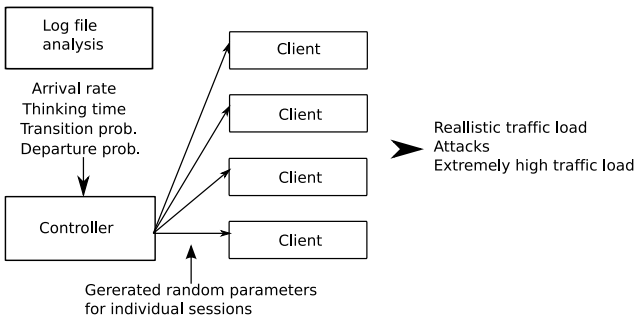

Figure 4. *SessionSim* **design.**

instances to many machines with different link speeds and delays. Each machine will simulate new session arrivals at a fraction of the aggregate session arrival rate so that the aggregate rate will match the tester's specification. The controller distributes model parameters to the client instances. In addition, the controller will monitor the aggregate traffic load and adjust the model parameters if necessary. Note that our *SessionSim* can also generate extreme traffic loads for stress testing, as long as the model parameters, e.g. session arrival rate, are set sufficiently high. It will also include clients that simulate different types of attacks, e.g. Ping-of-Death and TCP SYN flooding attacks.

3.3 Typical Testing Scenarios

Having presented the major components in Hercules, we give two examples to show how Hercules can be used by a company that is considering upgrading its application server software to provide better web service to their customers. We assume that the company has set up an emulation testbed. We also assume that they have maintained web server logs that record past users' requests (this assumption is not mandatory).

First, the testers will use our software to create two virtual testbeds, one with the new application server software and the other with the existing software. Second, they use our analyzer to analyze the web server's access log files and generate a web graph with the associated parameters (see Section 4.3). The controller in the *SessionSim* will disseminate these parameters to the web clients in the virtual testbed. Finally, they can run *SessionSim* to generate web traffic and collect performance statistics such as response time, memory and CPU load to compare the two application software.

Besides the testing scenario described above, Hercules can be used to test whether an existing infrastructure is capable of handling the growing number of users in the future. The tester first conducts a growth trend analysis to predict future traffic load, e.g arrival rate of new user sessions. Then the test can be carried out the same way as described above. If the estimated performance does not meet the expectation (e.g. the response time exceeds a predefined threshold), the given infrastructure needs to be upgraded to accommodate future demands.

4 Preliminary Results

We have performed some preliminary tests in Emulab, as a proof-of-concept of our infrastructure testing methodology. Moreover, we have developed a detailed model for simulating web users.

4.1 Emulation Testbeds

We have investigated the capability of two existing emulation testbeds – Emulab [2] and the Wisconsin Schooner router testbed [13]. Emulab has 160 Dell PowerEdge 2850 PCs each with a 3Ghz 64-bit Xeon processor and 2GB RAM, as well as 192 lower-end PCs. The Schooner testbed currently has 80 PCs and over 30 Cisco routers. It is actually set up using Emulab's software.

Although these existing testbeds lack certain common infrastructure hardware such as load balancers and firewalls, we can still use them to develop our testing methodology and tools. In the next section, we show how we use Emulab to test a simple web service infrastructure.

4.2 Testing Web Application Server

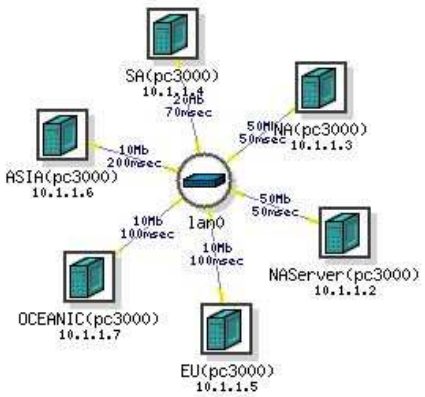

Figure 5. Test topology

Our experiments are designed to evaluate the feasibility of using emulation testbeds for infrastructure testing. Figure 5 shows the topology for a web application server test we have created in Emulab. We emulate a scenario where a web server serves users from all over the world. The web server runs FreeBSD (operating system), Apache (web server software) and Tomcat (application container).

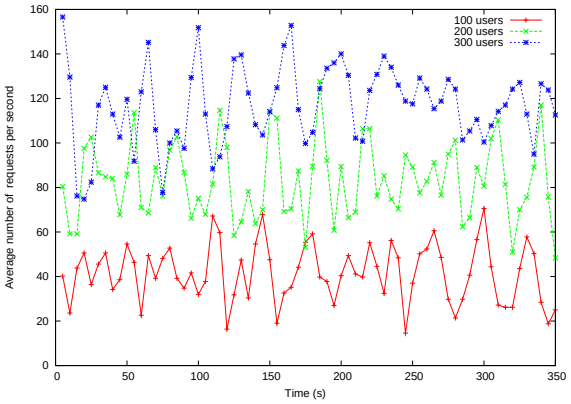

Figure 6. Service rate of the application server.

A real enterprise network would use more powerful application software than Tomcat (e.g. WebLogic or WebSphere), but we do not have these commercial software and they are not necessary here as our purpose is only to demonstrate the feasibility of our methodology. The users (called virtual users) from each continent are simulated in a different client machine with a link delay and bandwidth specific to that continent. We use 5 Windows XP client machines to simulate users from five different continents: North America, South America, Europe, Asia and Oceania, with 30%, 15%, 20%, 15% and 20% users, respectively. We use OpenSTA [9] to automatically create virtual users and each user runs a pre-recorded script repeatedly against the Tomcat Manager application. Note that once our *SessionSim* is developed, we can use it to generate more realistic traffic.

We carried out the experiments using three test cases with 100, 200 and 300 virtual users, respectively. Fig. 6 shows the average number of HTTP requests per second processed by the server. The service rate of the server varied over time (for every test case) and it seems to have a cycle of around 20-30 seconds. This is probably because the pre-recorded script that each user repeats lasts 26 seconds. We can also observe that as the number of users increases, the server processed more requests.

Test case	Average (ms)	Std (ms)
100 users	682	732
200 users	702	787
300 users	869	2737

Table 1. Average and standard deviation of server response time.

Table 1 shows the average and standard deviation of the HTTP response time. These two values changed only slightly from 100 users to 200 users, suggesting that the server had enough capacity to handle 200 users. However, both numbers (especially the standard deviation) increase noticeably for 300 users, indicating that the server was heavily loaded in this case.

This experiment shows that we can use Emulab to emulate a web service infrastructure along with its wide-area users. We found that it is quite easy to change the topology and testing cases. In the future, our *SessionSim* tool will automatically generate the scripts that the virtual users in OpenSTA (or other existing software) can use.

4.3 Web Traffic Simulation

In Section 3, we gave an overview of our model for simulating user behavior. However, the details of the user behavior depend on the specific applications. In the rest of this section, we describe our model for simulating web traffic.

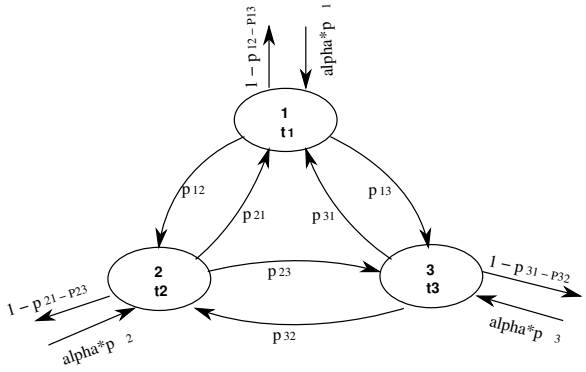

Figure 7. An example of a web graph containing 3 webpages.

We have developed a continuous Markov-chain model for simulating user requests in web sessions. Figure 7 illustrates our model. This example contains three web pages numbered from 1 to 3 corresponding to the three circles. We call this graph a web graph. First, our model assumes that new web sessions arrive at an aggregate rate of α. A user may start his/her web session by browsing any one of these pages. Therefore, the arrival rate of a new web session at page i is $p_i\alpha$, where p_i is the popularity of page i, given that $\sum_i p_i = 1$.

Next, the user spends a certain amount of thinking time reading a page after the page is downloaded. The thinking time for page i is t_i. The user then decides which page to download next. This is modeled by the transition probability from page i to j is p_{ij}. Moreover, the user may decide to leave the web session to do something else. We model this behavior using a termination probability. The termination probability for page i is $1 - \sum_{j:j\neq i} p_{ij}$.

All the above parameters can be estimated using access log files of web servers. In practice, the number of parameters involved in the web graph will not be proportional to

the square of the number of web pages because only few pages are linked to a particular page.

Type	Value
Total number of urls	6338
Http, static hypertext pages	20%
Http, static non hypertext pages	4%
Http, dynamic pages	64%
Https, static hypertext pages	1%
Https, static non hypertext pages	2%
Https, dynamic pages	9%

Table 2. Page makeup of the studied website.

We used *GNU wget* to discover the structure of an existing enterprise web site. *GNU wget* is a web crawler that recursively downloads web pages, images and other files and gives statistics related to each web page. Table 2 shows the total number of discovered web pages and the percentages of different pages. We will develop a tool similar to *GNU wget* to automatically build a web graph directly from an enterprise web site. In addition, the tool will analyze web access logs to estimate the transition probabilities and thinking time of each page. Without the web access logs for a particular web site, we can use the page size of that web site to determine the approximate thinking time of each page. We will need to analyze the correlation between thinking time and page size using other available web access logs first.

5 Related Work

In this section, we present related work in three areas: (1) large-scale network emulation; (2) network traffic characterization and modeling; and (3) traffic generators developed for application/infrastructure testing.

5.1 Large-Scale Network Emulation

Network emulation is a technique pioneered by the Flux research group at University of Utah led by Prof. Jay Lepreau [14]. It allows many researchers to share a single testbed that contains a large number of machines. The testbed machines are usually located close to each other (typically placed on many racks in the same room), but users can create arbitrary virtual topology using the same physical testbed – the mapping between the virtual topology and the physical topology is automatically done by the emulation software. The testbed can simulate arbitrary links by spacing the packets according to the specified link speed and delay. Moreover, users can run custom operating systems and software on the testbed machines by creating custom images. Before a user's experiment starts, the custom images are loaded onto the testbed machines. When the experiment is finished (or after a certain period of inactivity), the custom images will be "swapped out" so that the testbed resources can be used by other researchers. The *Emulab* at University of Utah [2] is the first and most widely used emulation testbed environment.

An *overlay network* is typically constructed over a wide-area network, e.g. the Internet, using many physical machines in geographically diverse locations. Similar to Emulab, users create different virtual topologies over a common physical topology. However, unlike Emulab, users do not have much control over the network speed and delay between the nodes, as the packets generated by their experiments are transported over a real wide-area network and are subject to the congestion caused by external traffic when their experiments are run. *PlanetLab* [11] and *RON* [12] are two popular overaly testbeds built over the Internet.

Our work follows the network emulation approach (as we have explained in Section 3). However, we will adapt this approach to better suit infrastructure testing, since current emulation software has been designed for networking and operating systems researchers.

5.2 Network Traffic Characterization and Modeling

There has been a considerable amount of measurement-based research on characterizing Internet traffic. Most notably, Leland et al. demonstrated that real Ethernet traffic does not follow the traditional Poisson-like model, rather it is self-similar [6]. Paxon and Floyd [10] studied wide-area traffic and confirmed the failure of Poisson modeling for packet-level traffic. However, they found that session arrivals seem to follow a time-varying Poisson model. Later research has confirmed the findings in both papers (see [15] for a comprehensive survey of the work in this area).

Our user session-level model is consistent with the above observations, i.e. we randomly generate the session arrivals using one stochastic process and the individual user requests within each session using another stochastic process. These two processes reflect different aspects of user behavior and thus should be modeled separately. On the surface, our model is similar to the one proposed in [8]. However, they model individual TCP connection arrivals, while we model user requests, which may or may not correspond to individual TCP connections. For example, if a web session uses persistent TCP connections, multiple user requests can be transported using a single TCP connection. Moreover, our web session model reflects how users randomly select and follow a path of their own choosing on a website, making the generated traffic more realistic.

5.3 Traffic Generators for Testing

There are many existing software tools for generating traffic loads for application and infrastructure testing, such as Mercury LoadRunner[7], OpenSTA [9], and Grinder [4]. However, they typically replay some pre-recorded sequences of user actions, so the testing traffic does not reflect the diversity of user behaviors. In addition, most testing tools generate rather deterministic traffic patterns, which does not take into account the randomness and burstiness of real traffic. Although they are called automated testing tools, the automation requires the testers to use certain scripting languages to create scripts to generate more sophisticated testing traffic. We want to free the testers from having to write the codes manually. In other words, our SessionSim will include tools to process log files from production services to estimate parameters for our traffic model. It will then automatically create scripts to be run in a chosen traffic generating software. In this way, we can have realistic traffic generated based on our traffic model and at the same time the common facilities provided by the popular software packages.

Hardware traffic generators also exist (e.g. IxLoad [5]), but they are more geared toward stress testing the infrastructure. As far as we know, our work is the first comprehensive effort to develop the methodology and tools for enterprise infrastructure testing.

6 Conclusion

We have presented an overview of the proposed Hercules testing environment for evaluating the performance and reliability of enterprise infrastructures. Hercules consists of two major components: (1) the methodology and tools to build flexible virtual testbeds using network emulation as well as to simulate infrastructure failures; and (2) a stochastic session-based model to generate realistic user traffic for testing purposes. As a proof-of-concept, we used Emulab to emulate a web service infrastructure providing services to users all over the world. We have also developed a detailed web session-based model for generating web traffic.

Our next step is to develop the proposed tools. More specifically, we will modify the existing Emulab software to make it more suitable for systems testers. We will also build *SessionSim* based on our model and verify whether the traffic it generates is consistent with that to/from enterprise networks. We will also evaluate how well Hercules performs in typical testing scenarios.

References

[1] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejae, P. Sharma, S. Shenker, K. Varadhan, H. Yu, Y. Xu, and D. Zappala. Improving simulation for network research. Technical Report 99-702, University of Souther California, Mar. 1999.

[2] Emulab testbed. `http://www.emulab.net/`.

[3] S. Floyd and V. Paxon. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, Aug. 2001.

[4] Grinder. `http://grinder.sourceforge.net/`.

[5] Ixia IxLoad. `http://www.ixiacom.com/products/aptixia_ixload/`.

[6] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, Feb. 1994.

[7] Mercury LoadRunner. `http://www.mercury.com/us/products/performance-center/loadrunner/`.

[8] C. Nuzman, I. Saniee, W. Sweldens, and A. Weiss. A compound model for tcp connection arrivals for lan and wan applications. *Computer Networks*, 40(3):319–337, 2002.

[9] OpenSTA. `http://www.opensta.org/`.

[10] V. Paxon and S. Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, June 1995.

[11] Planetlab. `http://www.planet-lab.org/`.

[12] Resilient overlay networks. `http://nms.csail.mit.edu/ron/`.

[13] The schooner router testbed. `http://www.schooner.wail.wisc.edu/`.

[14] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

[15] C. Williamson. Internet traffic measurement. *IEEE Internet Computing*, 5(6):70–74, 2001.