HttpTools: A Toolkit for Simulation of Web Hosts in OMNeT++

Kristján Valur Jónsson
Laboratory for Dependable Secure Systems
Reykjavik University
103, Reykjavik, Iceland
kristjanvj04@ru.is

ABSTRACT

Simulation studies have proven useful in networking research, as large-scale trials are often difficult to carry out in a reliable manner on deployed systems. This also applies to research in Web-based systems and applications, which are already an important part of our daily computing experience, and will surely proliferate in the coming years. However, support for high-fidelity simulation of Web browser and server nodes is currently lacking in the OMNeT++ community.

This paper describes a set of components, *HttpTools*, for simulation of Web hosts in OMNeT++. The toolkit consists of browser, server and global controller objects, which extend the INET framework and utilize its existing TCP simulation implementation for transport. We describe our components and report initial verification trials.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Discrete event*

General Terms

Simulation, performance

Keywords

Simulation,HTTP,Internet,Web applications,OMNeT++

1. INTRODUCTION

We have in the recent years seen a rapid increase in the number of deployed Web services and applications, as well as increased sophistication and complexity. Web applications are already ubiquitous, spanning the spheres of e-commerce, healthcare, finance and numerous other important areas. In particular, Web 2.0 [1] applications, powered by innovations such as *Asynchronous JavaScript and XML (Ajax)* [2], are sophisticated software packages, often rivaling their more traditional ”shrink-wrapped” desktop counterparts in speed and utility. We can thus surely expect an increase in the number of such systems in the coming years, which means that research in the area of Web applications is timely and relevant.

Simulation studies have already proven valuable in networking research, as large-scale trials are often difficult to carry out in a reliable manner on deployed systems. Furthermore, results obtained by observing real systems are often difficult to reproduce. It is therefore hard to obtain statistically significant comparisons of implemented solutions on real systems. Researchers in the networking field therefore commonly employ simulation tools for their studies to assess their various solutions and associated parameters in a repeatable and statistically significant manner. One of the most valuable tools in our opinion is the OMNeT++[1] [3] discrete event simulator. Support for high-fidelity simulation of Web browser and server nodes is though to the best of our knowledge currently lacking in the OMNeT++ community.

This paper describes our effort to create a set of components, *HttpTools*, for flexible Web traffic simulation in OMNeT++. The toolkit[2] consists of browser, server and global controller objects, which extend the INET framework [4] and utilize its existing TCP/IP simulation and infrastructure objects. *HttpTools* was developed to support our work in development of methods for distributed measurement of Web application behavior on end-systems, especially detection of anomalous behavior of Web clients. Realistic modeling of Web hosts is essential for our work, especially being able to simulate both a wide range of ”normal” conditions and injected anomalous code. In particular, we require models of individual nodes for this purpose, as opposed to the aggregate traffic generators which currently exist.

The *browser* and *server* components operate in conjunction to provide a realistic simulation of Web usage patterns. The *browser* component simulates a single browser node and generates periodic requests, emulating real user actions. We simulate usage patterns observed in the population, characterized by sessions of varying length, in which users periodically request pages from a number of Web servers. The parameters for session length, request rate and size, are determined by configurable random distributions. The browser component selects servers to communicate with at random using a site popularity distribution, managed by a *controller*

[1] `http://www.omnetpp.org`
[2] The code for the components, example scenarios and further documentation is available on our project Web site, `http://code.google.com/p/omnet-httptools`

component. It can also be run in a scripted mode, in which it makes a series of HTML requests to specific sites at predetermined times. The *Server* component simulates a single Web server. Page requests from browsers are answered by simulated HTML documents, containing a number of image and text resource references. The document can be randomly generated, according to statistics of Web page composition regarding size, number of referenced objects and their type. Similarly, the size of requested objects, e.g. images and CSS documents, are determined from statistical distributions. Alternatively, the server can be run in a scripted mode, in which it serves a number of pre-defined HTML documents and resources. Browsers which receive HTML pages with resource references open a connection, or number of connections, and retrieve each one in turn. The global *Controller* component supports the server and browser components by providing URL lookup services for the simulated servers. It enables browsers to request a random server, in accordance with a site popularity distribution, to support simulation of realistic browsing behavior.

The remainder of this paper is organized as follows: First, we discuss related work. Next, we describe the *HttpTools* components individually. We then discuss some applications of the HttpTools components and present basic validation results. Finally, we conclude with a summary of the current status and some thoughts on future work.

2. RELATED WORK

The *HttpTools* server and browser components are based on the `TCPBasicClientApp` and `TCPSrvHostApp` from the *INET* [4] framework. They have though been heavily modified to serve our purpose. The `httpclient` and `httpserver` components from the OMNeT++ *sockets* example provided additional inspiration. Both projects are considerably simpler than our implementation. In particular, we model a more complex user behavior and server document model, in addition to scripted browsing and pre-defined Web sites. The controller object, which provides server lookup, is also unique to our implementation and allows a great number of nodes to be simulated with minimum effort. A Web server simulator for OMNeT++ does exist[3], but does not appear to be actively developed at this time.

Several studies of Web traffic have been conducted, e.g. [5, 6, 7, 8, 9, 10]. Realistic modeling of any specific real-world results is though not considered to be within the scope of this work; rather our aim is to create a flexible architecture which can be applied to modeling a wide range of situations. Several measurement studies, e.g. the ones cited, were studied in preparation for this work and used to determine the parameters of interest.

Numerous Web traffic generators exist[4], e.g PackMime-HTTP [11] for the ns-2 simulator, SWING [12] and SURGE [13]. Similar components for the OMNeT++ simulator do however not exist to the best of our knowledge. The purpose of the aforementioned models, and others known to us, is primarily to investigate the effects of network traffic on the network infrastructure itself. Hence, a common approach is to model several nodes as an aggregate. We, however, require a more fine-grained approach, in which we can control each individual node. In fact, most of our research considers only the end-to-end interactions between browser and server nodes, and may even abstract from the network by employing the OMNeT++ direct message passing feature to eliminate the effects of the intermediary network links and nodes. The browser and server models can of course be employed to model an aggregate of nodes by applying aggregate measurement parameters. We believe that our modular approach provides greater flexibility than the Web server models we have researched.

[3]WebServer by Waldemar Kubassa, `http://metis.weia.po.opole.pl/~d18616`

[4]A summary of HTTP traffic generators can be found at `http://www.icir.org/models/trafficgenerators.html`

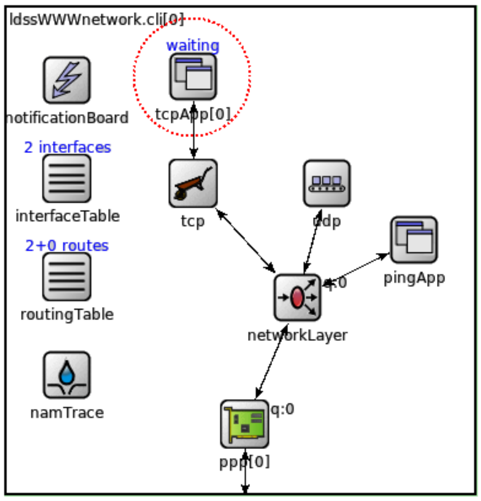

Figure 1: The `StandardHost` and its submodules. `HttpBrowser` and `HttpServer` components plug in as *tcpApps*.

3. THE HTTPTOOLS TOOLKIT

This section introduces the *browser*, *server* and *controller* components of *HttpTools*. A scenario utilizing the components consists of several browser and server nodes, in addition to a single global controller object. An example scenario is discussed in Section 4.

The server and browser modules utilize the TCP/IP modeling of the *INET* framework for transport and plug into its `StandardHost` module, as shown in Figure 1. The modules additionally support the OMNeT++ direct message passing feature, when the effects of the network infrastructure can be disregarded. A simple container, `DirectHost`, was created for this purpose, and can be used as a light-weight alternative to `StandardHost`.

3.1 Browser component

The purpose of the `HTTPBrowser` component is to simulate a browser operating on a single host. It operates in conjunction with the server component of Section 3.2 to provide a realistic simulation of Web usage patterns.

The activity of the browser is primarily characterized by the expected user behavior, as we are modeling an interactive application. Observed browsing patterns are characterized by periods of relatively frequent activity, divided by extended pauses. Users browse their favorite news sites, read e-mail and research a paper, generating bursts of re-

Parameter	Description
sessionInterval	The interval between activity periods
requestInterval	The interval between requests within activity periods
reqInSession	The number of requests per activity period
processingDelay	The delay in processing of each received HTML document before issuing requests for referenced resources
requestSize	The size of request, disregarding the TCP/IP headers

Table 1: Browser component parameters for random request mode

quests, while periodically returning to their daily chores of writing research papers. Activity periods, inter-request delays and request size distributions are thus primarily used in our solution to model the browsers behavior.

Browsers send GET requests to server objects, emulating real browsing behavior. The queried server returns simulated HTML documents, which can contain references to resources, e.g. images, CSS documents and scripts. The browser opens a connection, or number of connections, and retrieves each referenced resource. The simulated HTTP messages employed are further discussed in Section 3.4. The browser does not have any knowledge of the size of the requested object before it is returned, as it is solely determined by the server component, as further discussed in Section 3.2.

The browser component supports two modes of operation:

random request mode, in which the browser uses statistical distributions of usage patterns to generate requests to random Web servers.

scripted mode, in which the browsing behavior is determined by a list of predefined Web sites to visit at specific times.

3.1.1 Random request mode

The goal for this mode of operation is for the browser component to exhibit usage patterns, which are statistically similar to ones observed in real systems. Parameters are determined by statistical distributions, which can be derived from measurements. The parameters supported for the browsing simulation are shown in Table 1. All are defined in a XML configuration file, and assigned to the browser component at run-time using a XML initialization parameter and optional XPath-like section specifier. *Uniform*, *normal*, *exponential*, and *Zipf* random distributions are currently supported, in addition to a *histogram* object for empirical distributions. The browser selects a server to query in this mode by utilizing the global controller object to return a random server reference, as further described in Section 3.3.

Every browser in a simulation can be configured with different parameters, although this would quickly become unwieldy when setting up a large simulation. The flexibility to define groups of users, e.g. for light, normal and heavy browsing, is however useful in many cases.

3.1.2 Scripted mode

Predefined and repeatable actions by a subset of users in a simulation can be useful in a variety of situations. One example is an experiment in which we let the majority of browsers peruse HTML pages according to the nominal usage pattern, while a small subset behaves in a known manner, e.g. issuing anomalous requests consistent with a XSS attack. The general population here provides ”background noise”, behind which the attack may be hard to detect. Methods for detection of such events are of particular interest in our current research, which provided the inspiration for the creation of HttpTools. We thus support a scripted mode for our browser object, in which traces are replayed to inject browse events at specific times. We can of course mix browser nodes employing the random and scripted modes in a scenario.

The script file is specified by setting a *scriptFile* initialization parameter. The file format is:

```
{simulation time};{resource URL}
```

The *simulation time* parameter indicates the time at which the request for a specific resource, indicated by the *resource URL*, is made.

3.2 Server component

The purpose of the `HTTPServer` component is to simulate a Web server operating on a single host. It operates in conjunction with the browser component of Section 3.1 to provide a realistic simulation of Web usage patterns. The activity of the server is characterized by the requests it receives. It responds to requests by serving documents or resources.

A server receives a HTTP page request from a browser. If the browsers message is marked as bad, the server will respond with a 404:Not found message. Otherwise, the server will respond with a HTML response message. The body is a list of resource references, as further detailed in Section 3.4. The receiving browser is expected to issue requests for those resources, as discussed previously. The server in responds to each resource request with a message consistent with the object requested. The size of the HTML page or resource sent is solely determined by the server receiving the corresponding request.

The component has two modes of operation:

random document generation mode, in which the server responds to requests for HTML documents by generating replies according to statistical distributions.

scripted mode, in which the pages and resources served are predetermined.

3.2.1 Random document generation mode

The goal for this mode of operation is for the server component to exhibit traffic patterns, which are statistically similar to ones derived from measurements of real systems. The component is initialized with a number of parameters, shown in Table 2, which define its behavior. The parameters are defined in a XML file and can be determined at run time by various random distributions, as was discussed for the browser component.

In this mode, the server responds to a HTML page request by assembling a response containing a random number of resources. The number of resource references and their type is determined by random distributions, according to the initialization parameters. Similarly, requests for resources are

Parameter	Description
pageSize	The size of the generated HTML document
numResources	The number of referenced resources per HTML page
textImageResourceRatio	The ratio of images to text resources on HTML pages
imageResourceSize	The size of image resources
textResourceSize	The size of text resources, e.g. CSS documents
replyDelay	The delay in processing of each received HTML request before sending reply

Table 2: Server component parameters for random document generation mode

answered by messages of a size consistent with the size distributions for the object in question.

Every server in a simulation can be configured with different parameters, although this would quickly become unwieldy in a large simulation. The flexibility to define groups or categories of servers will however prove useful in many cases.

3.2.2 Scripted mode

Repeatable response by a subset of servers in a simulation can be useful in a variety of situations, e.g. for testing purposes. This is provided by the scripted server mode. A Web site is defined in some detail using a *site definition* file. The definition file is specified by setting a *scriptFile* initialization parameter. Its format is:

```
[HTML]
[path/]{name}[.extension];{definition file}
[RESOURCES]
[path/]{name.extension};{size}
```

The *HTML* section lists the hosted HTML pages and their associated definition files[5], one for each hosted HTML page. The page definition file is normally named U_R_L.def[6], although any name can certainly be used. The format of the page definition file is the same as used for the generated HTML message body (see Section 3.4), a simple list of the referenced resources. A optional entry with URL as *default* can be used to specify a default page for the site, e.g. an error page. The *RESOURCES* section lists all resources available for the site, their type (image or text) and size in bytes. The server responds with a message containing the specified page or resource if the request can be satisfied. Otherwise, a 404:Not found reply is issued.

3.3 HTTPController

The purpose of the `HTTPController` object is to support the server and browser components. A single controller instance must exist in all scenarios in which the browser and server components are used. Server components register their URL, OMNeT++ module name and listening port with the controller upon initialization. The controller serves as a lookup service for OMNeT++ module names from the URLs used to identify the Web server instances. It also provides a random Web site lookup facility, used by the browser component in its random browsing mode. *Uniform* and *Zipf* distributions are currently supported for random server selection. The controller parameters are defined in a XML file, as previously discussed for the server and browser components.

The controller supports *popularity modification events* for individual sites. Such events are defined in a script file, assigned to the controller by a initialization parameter. This feature can e.g. be utilized to model a flash crowd event, as further discussed in Section 4. A popularity modification event at a time T_{init} causes the site to be added to a custom selection list. A random site request from a browser will then yield a hit from the custom list with $p_c = \sum p_i$, where p_i is the enhanced selection probability of site i on the list. A particular site i is selected from the list with $p = p_i/p_c$. Conversely, a site is selected from the general population, using its assigned random distribution, with $p_p = 1-p_c$. The special probability of a site can be *amortized* by a constant α for each hit, eventually reducing it to zero, at which time it is removed from the special list.

3.4 The HTTP messages

`HttpRequest` and `HttpReply` messages are used by the browser and server components to simulate HTTP requests and replies. Their fields are shown in Table 3.

Servers generate replies containing simplified HTML bodies as payload, either according to the random parameters or site definition scripts, as discussed in Section 3.2. The body is a list of the form:

```
{resource}[;{site};{delay};{bad}]
```

resource is the only required field and contains a reference to a resource object, by default simulated as stored locally.

site is by default omitted, signifying that the resource is local. This field is only used when referring to resources hosted by third party sites.

delay is a optional parameter which causes the receiving browser to insert a delay before requesting the resource.

bad is a optional parameter used as a convenience when modeling anomalous messages. The receiving browser will generate a request to the target site, marked as bad (see Table 3). The target site will then automatically answer with a 404:Not found. This feature can be used to simulate usage errors or malicious behavior, e.g. DDoS attacks as further discussed in Section 4.

4. APPLICATIONS

An example scenario which uses *HttpTools* is shown in Figure 2. It contains a number of clients (browsers) and Web servers, in addition to a single controller. Three specific servers, *the good*, *the bad* and *the ugly*, to quote the spaghetti western, are defined. A number of generic servers

[5]Our toolkit includes a Python script for creation of definition files and associated page definitions from a set of actual HTML pages.

[6]U_R_L stands for the resource URL string, with / replaced by _.

message type	field	Description
Common	targetUrl	URL of the intended recipient
Common	originatorUrl	URL of the originator. Only applicable if the originator is a server.
Common	protocol	The HTTP protocol version, http/1.0 or http/1.1
Common	keepAlive	True if the keep-alive header is simulated as set
Common	serial	Convenience field, which allows resources requested and sent to be serially tagged
Common	payload	The message body as a string
Request	requestString	The request line. Emulates the HTTP request, e.g. GET /resource /protocol
Request	bad	Indicates that the browser is issuing an invalid request. The server responds with a 404:Not found.
Response	resultCode	The numerical result code, e.g. 200 for OK or 404 for not found
Response	payloadType	The type of the returned object, page, image or text resource, as an integer

Table 3: HTTP message fields

are represented here by a city object for simplicity. A network infrastructure can be defined, as shown by the routers and links in the figure or, alternatively, direct message passing between browser and server nodes can be employed.

Our current research focuses on investigating distributed measurement methods to detect anomalous Web traffic and deduce Web-based vulnerabilities. Large scale trials of such methods require simulation for analysis and verification. The fine-grained approach of modeling individual nodes rather than the aggregate as commonly done in other tools, is motivated primarily by our need for a accurate individual node objects to model a peer-to-peer aggregation network.

The simulations presented here are a part of our initial work but were performed strictly as a demonstration of the HttpTools components. The parameters used were approximated from informal measurements of Web traffic and browsing behavior on a handful of workstations, and are shown in Tables 4 and 5. The scenario used for the trials is similar to the one shown in Figure 2, containing 10000 browser nodes and 1000 generic server nodes, in addition to the servers *www.good.com*, *www.bad.com* and *www.ugly.com*. Uniform server popularity distribution was used, unless otherwise indicated. Startup transients were eliminated by throwing away the results of the first 6 hours of each simulation run. Confidence intervals are 95%, where applicable. The direct message passing feature was used to eliminate network effects, essentially assuming the network to be of infinite capacity with regards to the HTTP traffic. 3 to 6 hour bins were used, in which access counts are summarized, maintaining the mean, standard deviation and SEM values for each one. Hit counts are scaled to number of hits per server in the bin.

Let us first consider some simple verification runs. Figure 3 shows a sample of page requests made by ten clients (browsers). The session length, pauses between page requests and the number of requests in each session are all parameters of the simulation, as specified in Table 1. All parameters are here assumed to be normally distributed, although other random distributions can be employed. Page requests issued by a browser are indicated by a X marker. The randomly chosen session and request times can clearly be seen from this graph. Figure 4 shows the average request rate per server in the population using a uniform server popularity distribution. 15 minute bins are used and 95% confidence bands are drawn from the maximum SEM value. We would expect the results to be normally distributed by the central limit theorem and the large number of samples,

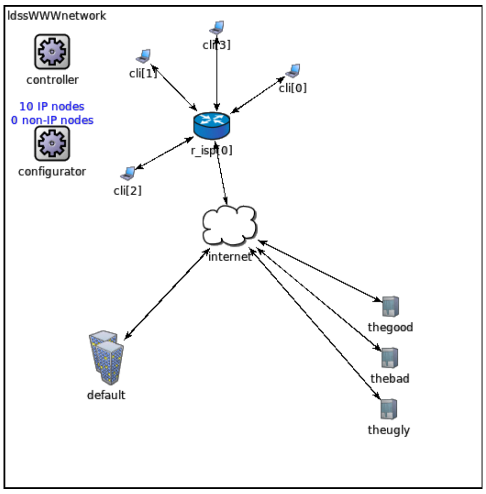

Figure 2: A sample scenario using *HttpTools* modules.

Parameter	Value
sessionInterval	normally distributed, $\mu = 3600$, $\sigma = 1800$ seconds
requestInterval	normally distributed, $\mu = 600$, $\sigma = 60$ seconds
reqInSession	normally distributed, $\mu = 10$, $\sigma = 5$

Table 4: Browser parameters for trial runs

Parameter	Value
pageSize	exponential, mean 10000 bytes, min=1000
numResources	uniform $[0, 20]$
textImageResourceRatio	uniform $[0.2, 0.8]$
imageResourceSize	exponential, mean 20000 bytes, min=1000
textResourceSize	exponential, mean 10000, bytes, min=1000

Table 5: Server parameters for trial runs

as indeed seems to be the case. The requests per minute, across the server population, can be approximated as con-

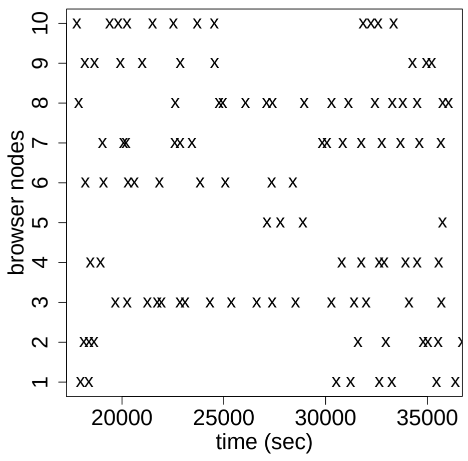

Figure 3: Page requests made by a population of ten browsers

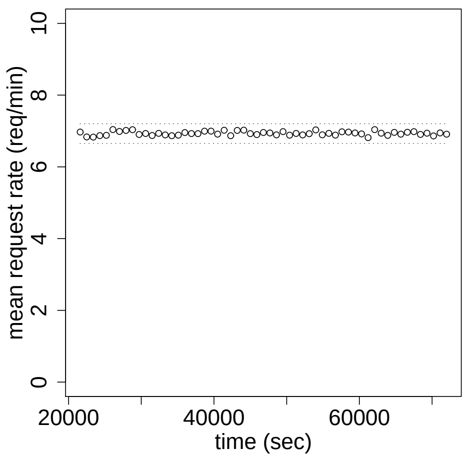

Figure 4: Average requests per server per minute using uniform server popularity distribution

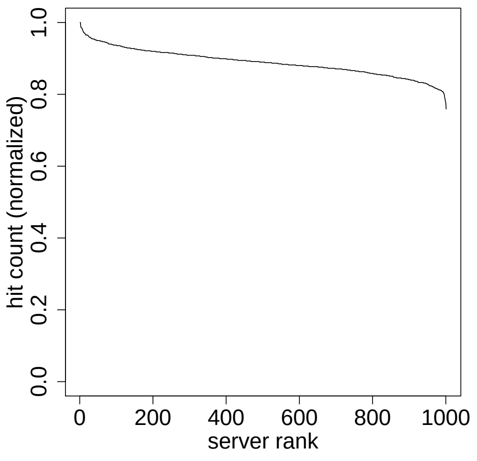

Figure 5: Ranking of Web servers when using uniform server popularity distribution

stant ($\mu = 6.9$, $\sigma = 2.5$), as expected since the requests are controlled by the browser population, which is expected to generate on average requests at a constant interval. The narrow confidence band indicates that the server popularity is indeed uniformly distributed – each server receives about the same number of hits. The normalized server popularity ranking, shown in Figure 5, further demonstrates the uniform popularity distribution. Note that the hit counts, used as a ranking metric, appear to be normally distributed, as would be expected. Figure 6 shows the server popularity ranking for a Zipf-distributed server access probability with $\alpha = 1.0$. The Zipf distribution ($p_n \approx 1/n^{\alpha}$) is commonly used to approximate popularity of objects such as Web sites. Informal observations of published rankings indicate that the Zipf distribution with $\alpha = 1$ is a reasonable approximation of Web site popularity in many cases. The simulation results, shown in Figure 6, correspond exactly to a $1/n$ function. The average request rate, 7.6 req/min, is similar to the previous run shown in Figure 4, as expected since the same pseudo-random sequence is used to generate the request events. $\sigma = 38.5$ is however significantly higher than in the previous case, demonstrating a greater variation in the individual server hit counts as would be expected.

We will now consider a slightly more complex scenario, modeling of a Web-based *distributed denial-of-service* (DDoS) attack, which misuses browsers as a distributed attack infrastructure. Similar attacks are described in [14].

First, let us briefly discuss a similar occurrence, *the flash crowd*, before presenting the DDoS attack modeling. The term was originally coined by Larry Niven[7] to describe the effects of instantaneous transfer of people between locations, using the fictional concept of transfer booths. It has been employed to describe the effects of a sudden avalanche of requests to a Web site. This is also known as the *slashdot effect*, originally used to describe the phenomenon of a small site being overwhelmed by requests as a result of being referenced by the popular technology news site *Slashdot*. It is now commonly used to refer to the same effect caused by other popular sites, which post links to items hosted on various servers. A flash crowd is thus caused by a sudden increased interest in a site. This may be inconvenient due to the inevitable reduced level of service, but is nevertheless a non-malicious event. A DDoS attack similarly causes a increase in the victim server request rate, with the associated waste of resources, e.g. CPU cycles and bandwidth. The two events may appear similar at the time of initiation, but there are subtle differences, which we will investigate in the course of our future work. This motivates the modeling of the flash crowd and DDoS attack scenarios described in the remainder of this section.

Let us first consider a flash crowd modeling scenario. We use the same scenario as previously, but introduce a popularity increase event (see Section 3.3) for *www.good.com* at time $T_{initiate}$. The popularity of the site is increased by a value p_{good}. Modeling the gradual reduction of the sites popularity to the nominal level is achieved by specifying a value α, the

[7]Larry Niven, *"Flash crowd"*, Three trips in time and space, 1973

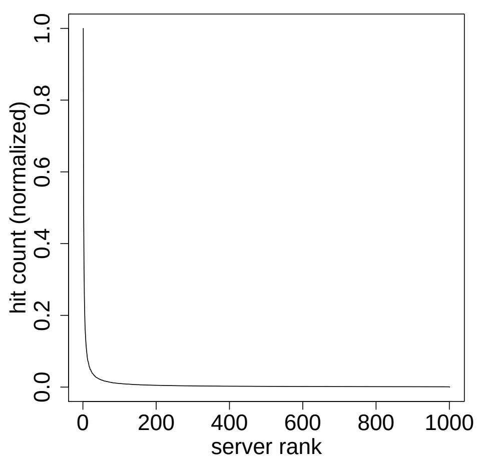

Figure 6: Ranking of Web servers when using Zipf-distributed server popularity

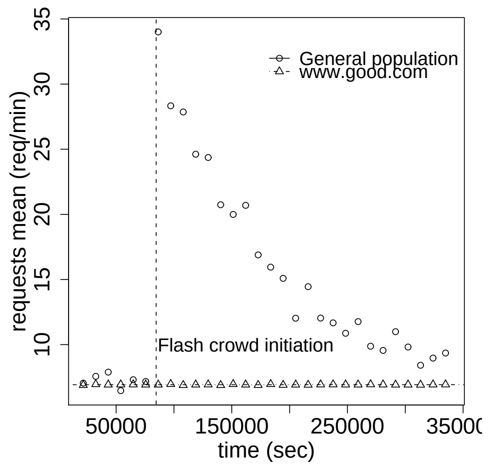

Figure 7: Flash crowd event initiated at T=24h for www.good.com

amortization constant, which reduces p_{good} on each hit for www.good.com. The elevated popularity status is canceled once p_{good} is reduced to zero. The results are shown in Figure 7. A popularity modification event for www.good.com, $p_{good} = 0.004$ and $\alpha = 0.000001$, is introduced at $T = 24h$. The results are similar to observed flash crowd events[8], a sudden traffic increase, followed by a gradual reduction over a period of time.

We now model two Web-based DDoS attacks, similar to those reported in [14]. A malicious node, *www.bad.com*, is activated at time T_{bad}. It serves a HTML page, containing a number of image references to a victim site, www.good.com. Requesting the images from www.good.com by a number of unwitting accomplices (users browsing www.bad.com) significantly increases its load. An attacker may attempt to mask the attack by varying its signature – the actual images requested, their order and request intervals. This is modeled by generating random number of requests and requests delays in our www.bad.com model. This attack can be hidden from the browsing user, e.g. by loading the images into 1 pixel frames, all but invisible. A second malicious server, *www.ugly.com*, is activated at T_{ugly}. This perpetrator employs an effective DDoS technique which sends a series of very long random URL strings to the victim, again using unwitting browsing users as accomplices. Parsing, validating the URL and returning a 404: Not found reply places significant load on the victim server. The www.ugly.com model generates HTML pages containing references to non-existent resources on www.good.com. The references are marked as *bad* when generated, meaning that they will generate a 404 reply from the victim. This is though simply done for convenience in the simulation. This attack too can be hidden from the browsing user, e.g. by using embedded JavaScript code to attempt to load the bogus resources into iframes. The modeling of www.bad.com and www.ugly.com was accomplished by subclassing the server component and rewriting about ten lines of code for each one. Only the random document generation method of the server class had to be redefined.

Figure 8 shows the result of our modeling. The number of 200:OK replies increases dramatically for www.good.com at 12 hours, when www.bad.com is activated. www.good.com starts to issue 404:Not found when www.ugly.com is activated at 24 hours. The volume of requests can easily reduce the level of service of a moderately sized server. Uniform server hit probability was employed, so the probability of any browser hitting a malicious site is expected to be 0.2% on each browse event generated. Each hit produces between 100 and 200 requests to the victim server (uniformly distributed), which will in all probability not be detected by the browser user. Even if detected, the particular user has already contributed to the attack. Further, it is very unlikely that the browsing user will be able to discern or notify the victim site of the attack or its perpetrator. At best, we can hope that the browsing user will avoid the malicious site in the future. Our future work will in part focus on detecting attacks of this kind in a distributed manner, enabling them to be detected early and hopefully traced back to the originator.

5. CONCLUSIONS AND FUTURE WORK

We have presented a set of components for simulation of Web traffic, consisting of browser, server and controller components. The browser and server components plug into the existing INET `StandardHost` module. Both components can be run in a random request mode, using statistical distributions to generate requests and replies. Parameters are configurable, allowing modeling based on real-world measurement results. The random request mode can be used to simulate the behavior of a large number of nodes with minimum configuration and setup. The components can also be run in a scripted mode, in which the browse events and Web sites are predefined. The controller object enables the browser population to locate servers based on their URLs, as well as to request a random one. Statistical popularity distributions are employed to randomly select a particular server from the population. The popularity distribution of the modeled servers is thus centrally controlled, allowing fast setup of simulations with a large number of nodes. Server

[8]See e.g. `http://www.neti.gatech.edu`, news section, where slashdot events for the NETI@home site are reported.

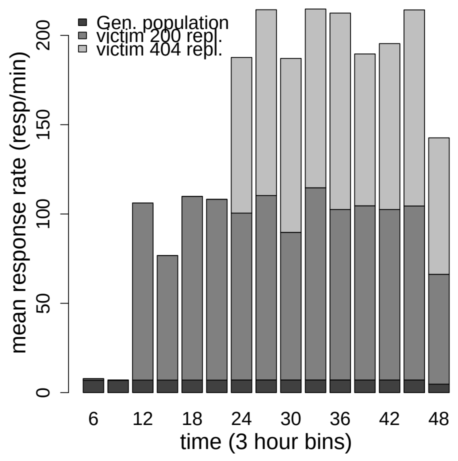

Figure 8: HTTP replies issued by www.good.com. A DDoS attack initiated by www.bad.com at T=12h and by www.ugly.com at T=24h.

and browser nodes can use the TCP/IP modeling of the INET framework for their transport, or alternatively, employ the OMNeT++ direct message passing feature, when a lighter-weight approach is required.

The HttpTools project was started to fulfill our needs for our particular research, but we hope that the components will be useful for the OMNeT++ community at large as fine-grained simulations of HTTP server and browser nodes. All code is publicly available, under GNU General Public License. The components have to date been used for initial experimentation in our work on developing methods for distributed monitoring of Web applications. Some examples of our preliminary results are reported in this paper and indicate that the components are indeed useful for a variety of situations.

HttpTools is currently work in progress and further development is expected in the coming months. This includes fuller parametrization of browser and server, using existing studies and our own measurements. Specifically, we plan to measure and model complex Web 2.0 Ajax applications, including mashups. Modeling of such sites will entail further development work on the components presented here.

6. REFERENCES

[1] T. O'Reilly, "What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software." [online] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, September 2005.

[2] J. J. Garret, "Ajax: A New Approach to Web Applications." [online] http://adaptivepath.com/ideas/essays/archives/000385.php, February 2005.

[3] A. Varga, "The OMNeT++ discrete event simulation system," in *European Simulation Multiconference (ESM'2001)*, June 2001.

[4] "INET Framework for OMNeT++/OMNEST." [online] http://www.omnetpp.org/doc/INET/neddoc/index.html, 2006.

[5] J. C. Mogul, "Network behavior of a busy web server and its clients," Tech. Rep. WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, October 1995.

[6] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of www client-based traces," Tech. Rep. BU-CS-95-010, Boston University, Boston, MA, USA, 1995.

[7] B. A. Mah, "An empirical model of http network traffic," in *INFOCOM '97: Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, (Washington, DC, USA), p. 592, IEEE Computer Society, 1997.

[8] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in web client access patterns: Characteristics and caching implications," *World Wide Web, Special Issue on Characterization and Performance Evaluation*, vol. 2, pp. 15–28, 1999.

[9] F. D. Smith, F. H. Campos, K. Jeffay, and D. Ott, "What tcp/ip protocol headers can tell us about the web," in *ACM SIGMETRICS*, (Cambridge, MA), pp. 245–256, June 2001.

[10] J. Charles Robert Simpson, D. Reddy, and G. F. Riley, "Empirical models of end-user network behavior from neti@home data analysis," *Simulation*, vol. 84, no. 10-11, pp. 557–571, 2008.

[11] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigle, "Stochastic models for generating synthetic http source traffic," *INFOCOM*, vol. 3, pp. 1546–1557, March 2004.

[12] K. V. Vishwanath and A. Vahdat, "Realistic and responsive network traffic generation," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 111–122, ACM, 2006.

[13] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 1, pp. 151–160, 1998.

[14] V. Lam, S. Antonatos, P. Akritidis, and K. Anagnostakis, "Puppetnets: Misusing web browsers as a distributed attack infrastructure," in *CCS06*, (Alexandria, Virginia, USA.), October 2006.