A Web-Based Network Worm Simulator

Nasir Jamil and Thomas M. Chen
Department of Electrical Engineering
Southern Methodist University
Dallas, Texas 75275
Email: nasir@mail.smu.edu, tchen@engr.smu.edu

***Abstract*— We present a worm simulator which can be run remotely through the web, based on the parameters supplied by the client. The core simulator program executes on the server and simulates the flow of the worm through a user-specified topology. The results of the simulation are then graphically displayed to the client. A variety of worm vectors can be simulated and various countermeasures such as rate throttling and quarantining can also be employed. The simulator uses a design that is efficient in terms of speed and memory requirements, while providing a lot of features for realistic simulations.**

I. Introduction

It is common to study networking problems by simulation due to the complexity and large scale of communications networks. Worms are a primary example of a networking problem that is well suited to simulation. Worms replicate themselves automatically from host to host, taking advantage of network connectivity. It is difficult to treat the dynamics of worm propagation by means of analysis because of the complex interactions between worm traffic, infection level, network congestion, and network topology. It is possible to treat worm epidemics at a simplified level of abstraction but simulations are necessary to verify analytic results and investigate dynamics at more realistic levels.

Worm simulators are typically designed and developed like other applications. A simulation program is written, compiled, and then downloaded by users to execute on their machines. Examples of worm simulators are [1]–[3].

While this approach obviously works, it has a number of well known drawbacks [4]. The simulator will be platform dependent, and each copy of the simulator and simulation results will be tied to a physical machine. User interfaces for different simulators may be inconsistent and perhaps difficult to understand. Each user will be responsible for maintenance and downloading of the latest version.

The World Wide Web offers the same appeal for simulations as for any application [5]. The simulator itself is hosted on a Web server but accessible from any Web browser. The main advantages are platform and location independence. Users can access the simulator and results from any computer through the Internet. The Web browser interface is familiar, consistent, and user friendly. In addition, users are relieved of the responsibilities for downloading and maintaining their own copies of a simulator.

Web pages are commonly viewed as static and non-interactive, but this notion is outdated. The Web browser allows flexible client-side interaction through technologies such as Java or Javascript. PHP and Ajax (asynchronous Javascript and XML) are also technologies enabling dynamic interfaces to server-side applications. One of the advantages to moving simulation to a Web-based architecture is the potential to take advantage of emerging Internet standards and technologies.

This paper describes and demonstrates a novel Web-based architecture for a worm simulator. NaSim is the first Web-based worm simulator to the authors' knowledge [6]. The simulator runs on the Web server, and receives input and outputs results through the Web browser. While it is possible to use the Web to run client-side simulation, that is the not the approach followed here because it would be limited to Java applets and not much conceptually different from downloading a simulator executable. One of the innovations is the separation of simulator and interface (which is the standard browser). In section II we present the high-level design of the simulator. Section III provides the implementation details of the server-side simulator core. Section IV presents some of simulation results generated with the simulator.

II. High-Level Design

The simulator follows a client-server design for enabling the user to order a simulation and then observe the results on his web page. Fig. 1 illustrates the simulator's high-level design. The front-end implements the client interface. After entry and validation of the simulator's parameters, the simulation is run by passing these parameters to the back-end through an HTTP POST request. The request is processed by the web server using the simulator's CGI script, which is responsible for managing the simulation process on the server side. The incoming parameters are passed on to the core simulator. This is followed by creating a software representation of the network topology. The nodes in this topology are then "populated" with hosts. The links in the topology are assigned bandwidth based on the properties of the end-nodes. Worm countermeasures are associated with the nodes based on the respective parameters. At this point, the core simulator starts simulating the traversal of the worm through the network topology. When this process is complete, the results of the simulation are stored on the server and the CGI script sends back a response to the HTTP request. The front-end then reads the data from the server and displays it in graphical format.

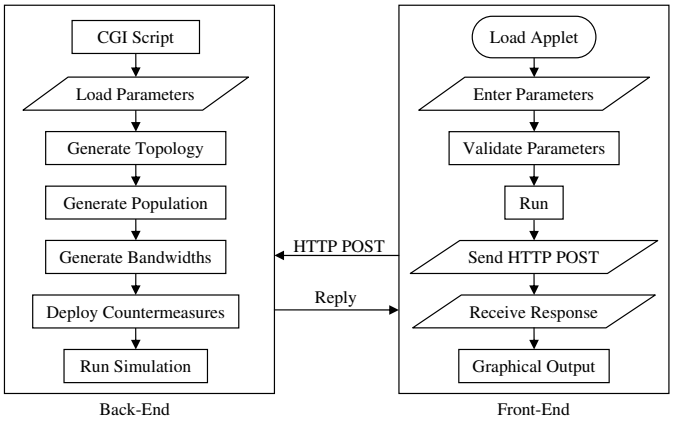

Fig. 1. High-Level Design

A. Web-Based Front-End

The front-end is an interactive GUI application provided by a Java applet and is shown in Fig. 2. It has provisions for data entry, data validation, context-based help, execution, and graphical display of the simulator's output.

The user-provided parameters have been divided into four categories; basic parameters, outbound rate control, inbound rate control, and quarantining. The basic parameters help specify the framework through which the worm would be simulated; the number of Autonomous Systems (AS) in the topology, algorithm for populating these nodes, link-capacities, and queue length. They also specify the worm vector, whether it is a uniformly spreading worm or a local-preferential one, and if the later then the percentage of probes targeting hosts of the same node.

The parameters relating to worm countermeasures can be activated on a per-group basis. The parameters for outbound rate control specify the percentage of households (AS) that would have this capability, the selection algorithm for such households, and the severity of throttling the worm traffic. The parameters for inbound rate throttling specify similar deployment criteria and throttle factor. In addition, they also specify the trigger-type and threshold for activating this countermeasure. Quarantining is the most severe countermeasure where all incoming and outgoing worm traffic is blocked. This measure will have its own deployment and triggering criteria.

Every parameter value is validated upon entry. If the validation fails, the field is highlighted in red color and positioning the mouse over the field would display a pop-up message explaining why the validation failed. When the *Run* button is clicked, a quick check is made to ensure that all fields have been successfully validated. If there are no validation errors, then the parameters are encoded into an HTTP POST message and sent to the applet's web-server.

B. CGI Back-End

The CGI script at the back-end is a python program which receives the simulation request containing the user-specified parameters. It *imports* the python core simulator and passes

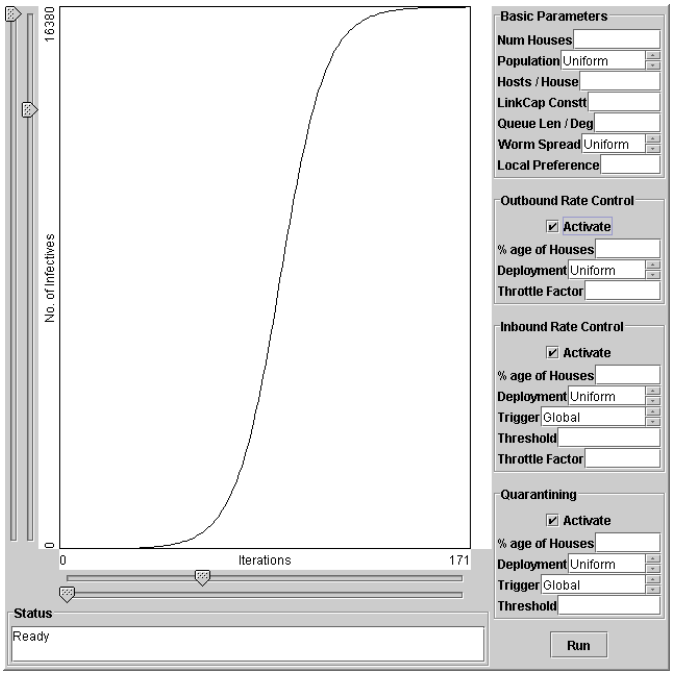

Fig. 2. Front-end of the NaSim simulator

these parameters to it along with a unique request-ID. It then invokes the core simulation function. When the simulation is complete the data is stored on the server using a file-name based on the unique request-ID and a response is sent back to the front-end. Upon receipt, the front-end reads the data file on the server and displays it in graphical form.

C. Topology Generation

By design, the simulator does not have its own topology generator. The aim is to take advantage of existing work in this area and use data from one of the available topology generators [7]. The simulator would then use this data to construct the topology in its own object space and simulate the flow or worms through it. Alternatively, one of the available data sets for Internet topology may also be used.

D. Core Simulator

After the topology is determined, the CGI script invokes the core simulation program which is a topological worm simulator. It simulates the behavior of the specified worm as it traverses the specified topology, subject to a combination of countermeasures deployed and invoked per user specification.

After topology generation, the core simulator populates the AS nodes with hosts and specifies the bandwidths for all the links. It then deploys the countermeasures among a percentage of the total nodes, selecting the nodes based on a user-specified criterion (random, ascending, or descending in terms of the node population). The deployed counter measures would then be triggered at a certain point in the simulation process when the triggering threshold is crossed. Each measure has its own parameters for deployment and triggering. After this step the core simulator starts the simulation.

E. Output Of Results

The core simulator stores the statistics of the simulation on a per-iteration basis and upon completion this data is stored on the web-server using the unique request-ID as a file-name. The front-end will then read this file and display the data in graphical form.

F. Job Scheduling

A feature of Web-based simulation is the possibility that multiple users may request simulations at the same time. Each user would be tagged differently based on the CGI process ID, date-stamp, and the client's IP address, and the simulation results would be stored in a file named by that tag. The simulator can also keep track of the number of requests in process and put a cap on this number, so as not to over-burden the server.

Since many users would be running simulations on the same server, a library of previously run simulations can be maintained. Thus, the client can browse through the simulations already run for different sets of parameters, and have the simulation results displayed based on the stored data file. The storage requirements for such files would not be much, and it would save time for the client as well as save the server from undue processing burden.

III. Implementation Details Of The Core Simulator

A. Representation of Network Topology

The Internet is a collection of many locally-administered domains called Autonomous Systems (AS). An autonomous system is connected to the global Internet through one or more access routers. Each AS can be treated as a node in a graph whose properties have been studied in the recent years [8]. The simulator starts with the logical representation of the Internet topology in terms of nodes and links:

1) Nodes: The simulator models the Autonomous system as a class object, containing the following details:

- Degree of the Node
- Links to other nodes
- Description of the hosts contained in the node
- Queue for infectious packets
- Deployed countermeasures
- Activated countermeasures

There are several topology generators for the Internet. The simulator is designed to accept the topological data generated by these programs, and represent it in the form of the AS objects called *households*.

2) Links: Each link between nodes is a link object whose reference is stored by the houses on both ends. A link object also defines its usage, capacity and cost. The link capacities may be imported from a data file or automatically assigned based on certain pre-defined criteria such as degree and population of the nodes at either end of the link. The cost of the link is derived by comparing its link capacity with the high-water mark of all link capacities and factoring in the usage of that link. This cost is then used by the routing algorithm for calculating the optimal path for the worm flow.

B. Representation of Hosts Within the Topology

Each node within the topology represents a *household* where the hosts live. The populations of the respective households may be imported through a data file or determined through a pre-defined algorithm, correlating the population of the node with its degree and link capacities.

The proper representation of the host data structures was a challenge. Host objects are the key structures for the entire simulation process and are going to occur in very large numbers. It is, therefore, imperative that they are stored in a way that leads to efficient processing and lowest possible memory requirements. There were two basic approaches for representing the hosts:

1) Per Household Storage: This approach would require each household to store all its host objects. Although appealing, it would make it difficult to choose a random host from the global population. This would make host targeting a complex and therefore less efficient operation.

2) Global Storage: According to this approach, all the hosts are stored in one linear array. That way, choosing a random array index would effectively choose a random host. Of course, identification of the target host is followed by determining the household it lives in, so that a realistic network traversal can happen from the source to the target household. This can easily be achieved by storing the household ID in each host object.

One more issue needed resolution before the global approach could be adopted. While each host knew its own household, how does a household keep track of all its hosts? This would be a requirement for local-preferential worms which target hosts within the household with higher probability. Of course, each household could store the IDs of all its hosts, but in view of the enormous number of hosts, that would be a real waste of memory. The solution was to have sequential IDs for all the hosts in a household. That way, storing the starting ID and the number of hosts is sufficient information for determining the hosts in a household, and if needed, for choosing a random target from among them.

C. Simulator Core

The core of the simulator is responsible for an iteration cycle of the simulator. Its goal is to cycle through every infected host calculating how to spread the infection. Infectious attempts that cannot make it all the way to the target are queued at the node beyond which it could not travel. Therefore, the core simulator also enables queued infectious vectors from previous iterations to reach their respective destinations. Here are the highlights of the core:

1) Infectious hosts: The simulator cycles through the hosts and when it comes across one that has been marked as infective, it gives it a chance to choose its target host. For a uniformly spreading worm, this would mean generating a random index for the host array. For a local preferential worm, if would use a weighted probability to decide whether to

target a host inside its household or globally. In either case, if the target is within the same household as the source, then there are no transmission issues to be considered and the infectious attempt is considered successful, provided that the target has not already been marked as infected or removed. If the target is outside the household then an optimal path would be calculated and a transmission attempt would be made.

2) Mechanism of Spread: When a worm vector needs to be transmitted across nodes (households), a least-cost optimal path is determined using Dijkstra's algorithm. The program then follows the links of this path and in each case verifies whether the usage is less than the link capacity. If so, then the usage and cost are incremented and it moves on to the next link. If it comes across a link whose entire capacity has been used up by other worm transmissions during the current iteration, then the target host ID is queued against the latest node along the path. The attempt would then be made during the next iteration cycle.

3) Queued Attempts: A node queues the ID of the target host when the link to the next node has no available capacity. Each node has a queue whose size may depend on the characteristics of that node. If the queue is full, the additional queueing attempts on that node are dropped.

At the start of each iteration cycle, the program first cycles through the queues of all the nodes and gives the queued attempts the first shot at reaching their destination. It is assumed that the link usage of an infection vector is for one iteration only, since transmission from source to destination can be completed in the same iteration if there is available bandwidth on the links. Consequently, the usage of all links is reset to zero at the beginning of each iteration cycle. This makes it possible that attempts which were queued in the previous iteration would now have available bandwidth to move forward.

D. Countermeasures

The countermeasures are deployed among a subset of the households according to the user's specification. At the end of each simulation cycle the status of countermeasures is examined for each household and they are activated if the household has been assigned that capability and the worm statistics exceed the triggering threshold. The main countermeasures of interest are router-based rate throttling (inbound and/or outbound) and router-based quarantining [9]–[14].

During the worm spread, an outgoing probe is dropped if the household has activated outgoing rate-throttling and the weighted probability function favors throttling for this one. Similarly, an incoming probe is dropped if the target household has activated inbound rate throttling and it is determined that this attempt needs to be blocked. It is worth noting that even if the attempt is blocked, the simulator still takes into account the bandwidth used by the probe to reach its target. The houses that have activated quarantining block all outgoing and incoming probes. None of these countermeasures impact the intra-household propagation of the worm.

E. Data Collection and Storage

One of the issues with large-scale simulations is that they can take a while to complete. With this possibility in mind, the core simulator is intended to work as a background process with no user interaction. As each iteration cycle is performed, data is stored on a per-iteration basis. All this data is written to a data file upon completion of the simulation. This data can then be used to resurrect the simulation process and give the user a sense that things are happening in realtime. In other words, the user interface has been disconnected from the actual simulation. This helps the simulation to run unfettered without wasting precious execution time in updating the user interface, while at the same time saving the user from having to watch the simulation at a very slow speed. This client/server approach made it possible to deploy the simulator on the web. It has also made it possible to display the simulator output from previously run simulations just by reading the data file.

IV. Simulations

The simulator was used to run 4 sets of simulations for 2200 hosts populated among 8 households, with a user-defined topology. A uniform-spreading worm was simulated in this scenario. The first simulation does not employ any countermeasures and the worm is only constrained by the available bandwidth. The second simulation deploys outbound rate-throttling in 50 percent of the households, with a throttle factor of 70 percent. The third simulation employs the same outbound throttling as before, and adds inbound throttling to 40 percent of the households, with a throttle factor of 60 percent. The fourth simulation uses the same throttling strategy as the previous two, but adds quarantining capability to 40 percent of the households. However, the quarantining capability is only activated when 60 percent of the hosts are infected globally. The graph clearly shows a significant slow-down in the rate of infection at around the 60 percent infective level.

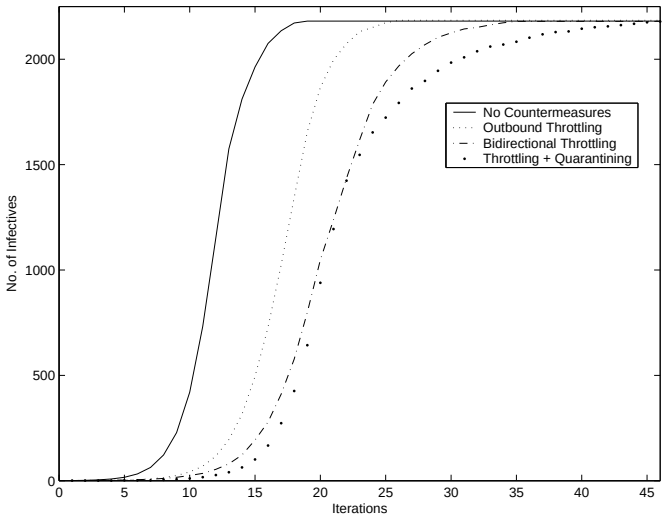

Fig. 3. Simulations showing the effects of countermeasures

The intent of these simulations was to display the capabilities of the simulator, so a small topology was used. There is further work underway, during which larger topologies would be examined and the results compared with those of known worm outbreaks.

V. Conclusions

The worm simulator presented in this paper simulates the flow of a variety of worms through a user-specified topology, taking into account a combination of countermeasures such as router-based rate-throttling and quarantining. One of the unique features of this simulator is the separation of the core simulator from the interface, thus making it possible to deploy the simulator through a web-page which interacts with the back-end server. Some simulations have been presented here which demonstrate the effectiveness of the countermeasures. Future work is anticipated in which this simulator would be used for much larger topologies and the results compared to data from previous worm outbreaks. For a certain type of worm, the comparison between the empirically observed data and the simulated data would serve as a baseline thus making it possible to relate the simulated effects of countermeasures to their real-world deployment. Towards that end, it will help evaluate the effectiveness of these measures with respect to different worm vectors. The simulator has already been put on the web and is functional.

References

[1] M. Liljenstam, "SSF.App.Worm: A network worm modeling package for SSFNet." http://www.crhc.uiuc.edu/~mili/research/ssf/worm/index.html, Sept. 28, 2006.

[2] M. Liljenstam, Y. Yuan, B. J. Premore, and D. Nicol, "A mixed abstraction level simulation model of large-scale Internet worm infestations," in *10th IEEE/ACM Symp. on Modeling, Analysis, and Simulation of Comp. Telecom. Sys. (MASCOTS 2002)*, (Fort Worth, TX), pp. 109–116, Oct. 11–16, 2002.

[3] B. Ediger, "Network Worm Simulation (NWS) System." http://www.users.qwest.net/~eballen1/nws/, Sept. 26, 2003.

[4] T. K. Leong, B. Ali, V. Prakash, and N. Nordin, "A prototype of Web-based simulation environment: using CGI and Javascript," in *Proc. IEEE TENCON 2000*, (Kuala Lumpur, Malaysia), pp. 357–360, Sept. 24–27, 2000.

[5] J. Kuljis and R. J. Paul, "A review of web based simulation: whither we wander?," in *WSC '00: Proceedings of the 32nd conference on Winter simulation*, (San Diego, CA, USA), pp. 1872–1881, Society for Computer Simulation International, 2000.

[6] N. Jamil and T. M. Chen, "NaSim (Nasir's Worm Simulator)." http://engr.smu.edu/~nasir/simulator.html, Sept. 28, 2006.

[7] J. Winick and S. Jamin, "Inet-3.0: Internet Topology Generator," Tech. Rep. UM-CSE-TR-456-02, University of Michigan, 2002.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," *ACM SIGCOMM 99, Computer Communication Review*, vol. 29, pp. 251–262, Oct. 1999.

[9] N. Jamil and T. M. Chen, "Effectiveness of Rate Control in Slowing Down Worm Epidemics," in *IEEE Globecom 2006*, (San Francisco, California, USA), Nov. 27 – Dec. 1, 2006.

[10] M. Williamson, "Throttling viruses: restricting propagation to defeat malicious mobile code," in *18th Annual Comp. Sec. Appl. Conf.*, (Las Vegas, NV), Dec. 9–13, 2002.

[11] P. Porras, L. Briesemeister, K. Skinner, K. Levitt, J. Rowe, and Y.-C. A. Ting, "A hybrid quarantine defense," in *ACM Workshop on Rapid Malcode (WORM 2004)*, (Wash. DC), pp. 73–82, 2004.

[12] G. Ganger, G. Economou, and S. Bielski, "Self-securing network interfaces," Tech. Rep. CMU-CS-02-144, Carnegie Mellon U., Aug. 2002.

[13] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet quarantine: requirements for containing self-propagating code," in *IEEE Infocom 2003*, (San Francisco, CA), pp. 1901–1910, 2003.

[14] T. M. Chen and N. Jamil, "Effectiveness of Quarantine in Worm Epidemics," in *IEEE International Conference on Communications (ICC 2006)*, (Istanbul, Turkey), June 11 – 15, 2006.