Simulating Internet Worms *

George F. Riley[1]
Monirul I. Sharif[2]
Wenke Lee[2]

[1]Department of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250
riley@ece.gatech.edu

[2]College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
msharif@cc.gatech.edu

Abstract

The accurate and efficient modeling of Internet worms is a particularly challenging task for network simulation tools. The atypical and aggressive behavior of these worms can easily consume excessive resources, both processing time and storage, within a typical simulator. In particular, the selection of random IP addresses, and the sending of packets to the selected hosts, even if they are non–existent or not modeled in the simulation scenario, is challenging for existing network simulation tools. Further, the computation of routing information for these randomly chosen target addresses defeats most caching or on–demand routing methods, resulting in substantial overhead in the simulator. We discuss the design of our Internet worm models in the Georgia Tech Network Simulator, *and show how we addressed these issues. We present some results from our Internet worm simulations that show the rate of infection spread for a typical worm under a variety of conditions.*

1. Introduction

The Internet has recently been the target of widespread denial–of–service attacks in the form of Internet worms. A typical worm will exploit some vulnerability in a host operating system, usually an implementation bug or design flaw, to cause the execution of a set of unauthorized software codes on the victim systems. This malicious software, or *MalWare*, can then inflict any manner of disruptive actions on the victim, ranging from nothing at all, to complete destruction of all programs and data on the victim.

Further, the worm will attempt to replicate itself on other end–hosts with similar vulnerabilities. It does this by sending so–called *Infection Packets* to randomly chosen victims. When an infection packet reaches a target system with the same vulnerability, that system also becomes infected and in turn attempts to infect even more victims. It is easy to see that the total number of infected hosts over time (called the *Infection Rate*) can exhibit exponential growth in certain conditions. Indeed, it is estimated that the recent *Slammer* worm reached 90% of susceptible hosts in just a few minutes[10]. It is important to note that, for the class of Internet worms of interest here, no human intervention at all is needed for the worm to spread. In contrast, methods that use e–mail attachments or other file sharing methods, and require a user actions to execute the *MalWare* can potentially be thwarted simply by increased user awareness.

Given the potential for widespread disruption of the Internet, the need for detailed modeling and analysis of the behavior of these worms is obvious. Understanding how to detect when a worm attack is underway would lead to the development of filters or screening methods that could lessen the effect of future worms. Detecting and reacting to attempted infections, perhaps with active countermeasures, could slow down the infection rate sufficiently that human intervention could effectively limit the damage. Detailed studies of the so–called *White Worms* could lead to the development and deployment of active countermeasures that would eliminate or block the vulnerabilities before widespread infections occur.

We have developed worm models for the *Georgia Tech Network Simulator* (*GTNetS*) [12] that can be used to study the behavior of these Internet worms under a variety of conditions. We have included models for both the *UDP* and *TCP* style infection methods, since there have been instances of both methods in recent worms [16]. Our simulation models include complete packet–level detail, both for the worms and competing traffic on the modeled subnet-

* This work is supported in part by NSF under contract number ANI-0240477, and in part by DARPA under contract number N66002-00-1-8934.

works. By using this level of detail, we capture behavior such as queue buildup and subsequent increase in round–trip time due to the worm traffic itself, as well as the affect of competing traffic on the infection packets. This level of detail is necessary to capture a complete picture of the worm's behavior in the presence of other data flows and defensive mechanisms.

The remainder of this paper is organized as follows. Section 2 provides some information on worm modeling efforts by others, both in packet–level simulations and with analytical models. Section 3 describes in detail our worm models and section 4 discuses the modifications needed in the *GTNetS* simulator to efficiently model the worms on a large scale. Section 5 presents some preliminary results from our simulation–based experiments of worm spreading. Section 6 gives some concluding remarks and future directions for this research.

2. Related Work

In this section, we discuss some existing simulation tools that have been used to model the behavior of Internet worms. These existing tools can be loosely separated into two categories, the analytical models and the packet–level models.

A good analytical model of the infection rate of Internet worms can be obtained by basing the analysis on epidemiological models of disease spread[5]. Several of the characteristics of the Internet worms have direct correspondence to disease models, such as the infection probability, infection population, hosts interactions, and others. Indeed, using such an approach was discussed as early as 1991 [7]. Zou, Gong, and Towsley [17, 18] used this approach to obtain predict the spread of worms under a variety of conditions, and to measure the effectiveness of a proposed detection and defense mechanism.

Gao et. al [1] discuss another analytical approach called the *Analytical Active Worm Propagation* (*AAWP*) model. This model improves upon the epidemiological models in a number of ways, including accounting for systems crashing, systems being repaired and patched, and delay times for infected hosts to actually begin spreading to others. Further, Gao's approach uses a discrete time model as opposed to the continuous time models in the earlier approach.

The primary benefit of the analytical models is computational efficiency, in that the numerical solutions to these models are largely independent of population size (topology size), and can easily predict worm behavior on networks of millions of elements. However, such models typically fail to take into account the affect of queuing delay, packet loss, and round–trip time delays in their predictions, nor can they measure the affect of the worm traffic on other legitimate traffic.

Packet–level detail models on the other hand, can be constructed to model in detail the effect of network characteristics, such as queue length, queuing discipline, link bandwidth, routing protocols, and the like. A significant work in this area is by Liljenstam et. al [8]. In this work, the *SSFNet* [4, 3] simulator was extended to include models of the behavior of Internet worms, and used to measure the worm's affect on the network dynamics. Their approach was a hybrid method, using complete packet level details for part of the network, and a less accurate but more computationally efficient model for other parts. They report good success with this approach in predicting the overall spread of a typical worm, and the affect of this spread on the network as a whole.

In [15], Wagner discusses a detailed simulator for worm propagation implemented in the *Perl* scripting language. In this work, the affect of link bandwidth and propagation delays are taken into account, but queuing, loss, and competing traffic are ignored. Further, the *TCP*–based worm models use a simplified behavior model for *TCP*, lacking slow start and congestion window management features.

The venerable *ns2* [9] simulator has recently been extended to include some models for worm behavior. The approached used in *ns2* is similar to that use by *SSFNet*, namely the hybrid approach with detailed packet–level simulations for only a small part of the network. Further, *ns2* has no built–in mechanisms for assigning *IP* addresses to nodes, and thus the address scanning models for worms become problematic.

To our knowledge, our approach is the only one to date that allows full packet–level detail simulations of worm infections for moderately large networks, including details of *TCP* connection establishment and slow start, correct forwarding of packets addressed to non–existent end–hosts, and the effect of competing legitimate traffic on the spread of the worm.

3. The *GTNetS* Worm Models

In this section, we discuss in detail the design of our worm models for *GTNetS*. To provide background for understanding our model design, give a brief overview about a worm's life cycle from its activation to its spreading to other hosts. Even though attempts of classifying worms have been made [16], it is still hard to find a way to make a completely generalized model that fits all of the classes. In this work, we are only concerned with the *Traditional Worms*, that do not require human intervention to spread. They infect other systems by exploiting vulnerabilities in the software. From a network point of view, these vulnerabilities are mostly due to flaws in *Application-Level* functionality that is typically built on either *TCP* or *UDP* transport level protocols. The most common flaw in the application level

software is the *Buffer Overflow* bug [2], which can erroneously allow the system to execute code contained in specially built network packets. This code either contains the complete program of the worm, or a portion of it that opens a side channel to download in the rest of the code from the infecting host. This phase can be called the *Propagation Phase* of the worm. Subsequently, the worm code executes instructions on the host in order to gain privileged or root–level access to the compromised system. In this *Activation Phase*, the worm code can deploy any kind of payload or proceed with the task of infecting other hosts. In this *Infection Phase* the worm instance selects target hosts by generating *IP* addresses of possible victims. The use of the 32–bit *IPv4* address space lets worm writers use several target selection strategies (also called the worm's *Target Vector*), such as hit–list scanning, uniform random scanning, local preference scanning etc. These scanning techniques are also an important factor affecting a worm's infection rate.

While designing our worm model and worm simulation environment, we aimed at fulfilling several objectives. The first objective is that the worm model should have minimal overhead in the network simulation environment. Simple and efficient worm models will enable large–scale simulation of worms by keeping memory and CPU overhead to a minimum. Our target is to achieve million node simulations of worm propagation. While the *GTNetS* simulator has demonstrated the capability of multi–million node simulations [6], these demonstrations used idealized scenarios, and thus are not directly applicable to worm simulations. However, we have achieved more than 50,000 nodes in a worm simulation, and have not yet exploited the distributed simulation features of *GTNetS*, and thus are confident of meeting our longer–term goals.

The second objective is that the model should capture the important parameters that affect propagation and infection rate with packet level details. A third objective is that it should be flexible enough for supporting a wide range of worm activities and classes.

The worm models in *GTNetS* are designed around the behavior of the real worm code and underlying network protocols. Therefore the model allows parameters to be set that represent the worm action in its different phases. For the complete worm simulation, several other parameters including the network topology, host vulnerability probability, and the *IP Address* distribution must be specified using inherent *GTNetS* features. The parameters describing the worms are:

- Transport Protocol - The underlying transport protocol creates a distinct difference in propagation from a network's point of view. *UDP* based worms such as the *Slammer* [10] have small infection packets that are generally transmitted as fast as possible, limited only by the bandwidth of the outgoing link. These packets might get dropped, but the worm itself does not wait for any acknowledgments from the target. The propagation speed and therefore the overall scan rate of such a worm depends on the available bandwidth of the network. On the other hand, worms exploiting *TCP* require a connection to be established before the payload packets are sent, and therefore the propagation speed depends on the average round trip time (*RTT*) between hosts. These classes of worms are called *RTT-limited*. Such worms improve propagation speed by using several simultaneous *TCP* connections, typically by writing the worm as a multi-threaded application. Our model supports both of these types of worm classes. We should also add that the model is flexible enough to be extended to incorporate propagation methods that use both methods.
- Infection Length - This parameter specifies the size of exploitation data that is sent to a victim host to infect it. For example, smaller worms might need just a single *UDP* datagram or *TCP* segment, and presumably little or no *IP* fragmentation. For *TCP* based worms more *TCP* segments might mean longer delay due to several round trip times being required to transfer the worm payload. Large infection payloads can also cause more aggregate data being transferred through the network, leading to higher probability of saturating the links quickly or dropped packets due to queue overflows.
- Infection Port - The transport layer port that is used to send infection packets to can be specified. When a worm instance starts sending infection packets to other hosts, it is addressed to this specified port on the target host. This feature allows simulations with competing background traffic to hosts on other ports, or the same port.
- Target Vector and Scanning Pattern - The worm model in *GTNetS* allows flexible selection of scanning patterns. This is implemented as an extensible class that generates *IPv4* addresses for a particular worm instance on a host. This behavior of this class has several variations, as follows:
  - Uniformly Random Scanning - Any *IP* address is generated in a specified address range with the same probability.
  - Local Preference Scanning - A victim *IP* on the same subnetwork as the presently infected host is chosen with higher probability than other *IP* addresses.
  - Sequential Scanning - *IP* addresses are chosen sequentially within a specified range.

  Since our implementation is an object–oriented approach, the target vector generation class can easily be

overridden by the simulation user to create any scanning method desired.

- Scanrate - This is for *UDP* based worms only, since the rate at which infection packets are generated by *TCP* worms is a function of the number of simultaneous *TCP* connections and other external factors. For *UDP* based worms, the worm transmits infection packets at the rate specified by this parameter.
- Connections - This parameter is used to model the number of simultaneous connections that are used by *TCP* based worms. The *Code Red II* worm [11] used 300 threads to connect to different target hosts at the same time, allowing a maximum of 300 sustainable *TCP* connections. Our model allows worms like this to be represented in the simulation.

Another important factor in studying worm propagation is the network topology used to carry worm and background traffic. *GTNetS* offers the flexibility of setting up any arbitrary topology, by connecting together any of the many pre–defined topology objects. In the next section, we describe our additions to *GTNetS* that allow creation of reasonable topology models, in a scalable and efficient manner. Further, we describe methods to alleviate potentially excessive overhead in the simulator due to the random target selection methods used by worms.

4. Enhancements to *GTNetS*

The worm behavior itself leads to several potential inefficiencies in a simulation environment, which must be addressed to achieve scalable and efficient simulations. We discuss these issues and our solutions in the *GTNetS* simulation environment.

From an efficiency standpoint, a primary concern in network simulation environments is the computation and storage of packet routing information. A common approach is the a priori computation of *routing tables* for every node in this simulated topology. This method is easy to understand, easy to implement, and is a reasonable approximation of correct packet routing in the network. However, as the size of the simulated topology increases, the CPU and memory demand on the simulator become unmanageable. *GTNetS* solves this problem by using the well–known *NIx–Vector* [13, 14] routing method. With this method, routes are computed only as needed, and are stored in the *packets*, rather than in routing tables. The route computation uses a breadth first search (*BFS*) algorithm on the topology graph to find the shortest path from a source to a destination. The *NIx–Vector* is an efficient method for storing routing information in packets, and can represent most routes in 32 bits or less. To avoid excessive CPU time for computing the routes, the *NIx–Vectors* are cached at the source node and re–used as needed. The *NIx–Vector* routing method has been shown to be a reasonably efficient method for packet routing in simulated networks [13].

However, the generation of infection packets to randomly generated targets circumvents many of the benefits of *NIx–Vector* routing. The *BFS* algorithm is executed repeatedly, with little likelihood of a cache hit on a previously calculated route. Further, finding routes to non–existent hosts causes the *BFS* algorithm to exhaustively search all paths before determining that no route exists. Finally, worm packets *must* be routed in a reasonable way, even if the target does not exist (implying that no *NIx–Vector* can be found). We address these issues in our *GTNetS* worm models by several enhancements in the route computations, described below.

4.1. Routing Proxies

The notion of *Routing Proxy* was introduced in *GTNetS* to reduce the cost of searching for routes to nodes (either existing or non-existent), and to enable realistic forwarding of packets that are destined to non-existant hosts. A *Routing Proxy* can be expressed by a pair (A, S) where A is the base address of the range and S is the network size. For example $(102.10.0.0, /16)$ represents a $/16$ network starting at the address $102.10.0.0$. Any output link interface I at a node N in the network topology may be assigned one or more *Routing Proxies* denoted as P_i having the following properties:

- The *IP* addresses of any node M that is reachable via N through interface I *must* be contained in any of the address ranges P_i.
- The *IP* addresses of any node M that cannot be reached via N through interface I *must not* be contained in any of the address ranges P_i.

The above insures that when defined, the *Routing Proxies* should reveal all possible addresses reachable by that node through that interface. It should be noted though, that for nodes that are normally in the backbone or have a large list of reachable address ranges, the *Routing Proxy* would likely be omitted, since in all likelihood there would be many sets of *IP* address ranges reachable through that node. In our experiments, we defined the proxies only at the gateway nodes to sub–networks.

When the route computation algorithm encounters a node with a *Routing Proxy* defined that encompasses the target *IP* address, the algorithm halts and returns a *NIx–Vector* to the node containing the routing proxy. When the corresponding packet later arrives at the node with the routing proxy, the appropriate output interface can easily be determined by searching the proxy information. Thus, even

packets addressed to non–existent nodes will be properly routed to the ingress gateway on the proper subnetwork, just as similar packets would be in the Internet.

4.2. BFS Pruning

As mentioned earlier, the *BFS* route computation algorithm can be a source of significant CPU overhead in the simulator. Our *Routing Proxies* feature described above allows for an improvement in the standard *BFS* algorithm that substantially reduces the CPU overhead for route computations. Our modified *BFS* algorithm incorporates the information found in the *Routing Proxies* to reduce the searching cost.

We start by further restricting the definition and meaning of a *Routing Proxy* as follows:

- With *Routing Proxy* entries for node N and interface I defined, *only* those nodes with *IP* addresses encompassed by any one of those *Routing Proxy* entries is reachable via that interface.

The importance of the third restriction is such that the *BFS* algorithm can prune large portions of the topology graph when computing a shortest path route. Since the proxy specifies that only the specified range is reachable via a specific interface, all neighbors of that interface can be pruned if the destination address is not in the specified range.

A portion of the modified *BFS* algorithm is given below.

```
u = Q.GetFront()
if (u.HasRoutingProxyInfo() &&
      !u.CanRoute(sourceIP) &&
      !u.CanRoute(targetIP))
      Prune(u)
else {
      Continue processing...
}
```

4.3. NIx–Vector Aggregation

The use of the *Routing Proxies* leads to another improvement in the computation and storage of routes, namely the use of *NIx–Vector* aggregation. Recall that the *NIx–Vector* calculation will stop when a node is found that has a *Routing Proxy* defined encompassing the target *IP* address. The calculated *NIx–Vector* is used to route the packet to the proxy node, which in turn routes the packet to the final destination. However, in this case the calculated *NIx–Vector* is valid not only for the original target *IP* address, but for all *IP* addresses *in the proxy range*. In other words, if a *NIx–Vector* to a /16 proxy address is calculated, that same *NIx–Vector* is correct for 2^{16} targets encompassed by that proxy. Thus, when future packets are addressed to any target in the same /16 range, the same *NIx–Vector* should be used.

The *NIx–Vector* aggregation feature stores the cached *NIx–Vectors* at each source node along with a specification of the *IP* address range for which this *NIx–Vector* is valid. Subsequent cache lookups for a target *IP* address in that range return the same *NIx–Vector*. This adds a small extra cost in *NIx–Vector* cache lookups, but reduces the overall number of route computations needed in the worm simulation.[1]

4.4. Random Tree

Another important issue when studying the infection rate of Internet worms is a reasonable topology graph, particularly one that has some mix of *IP* addresses that are valid and modeled, as well as addresses that represent non–existent hosts. In other words, for a given subnetwork with a /16 address range assigned (for example), it is rarely the case that all 2^{16} *IP* addresses in that range actually are assigned to a working host. It is likely the case however, that all packets addressed to any address in that range from elsewhere in the network will be delivered to the subnetwork (at a minimum to the ingress gateway), and then later dropped at some point within the subnetwork. The correct and efficient modeling of these *holes* in the *IP* address space is an important criterion for simulation–based worm studies.

In *GTNetS*, we introduced a topology model object for a *Random Tree*. A random tree is defined by it's depth and fanout values, as are normal tree objects, but extended to include a *child probability factor*. As the tree topology is being constructed, child nodes are randomly pruned based on the child probability factor. The result is a topology object with a *sparse IP* address space, with some fraction of the possible *IP* addresses unassigned. To illustrate this point, we give an example.

Let f be the fanout, d be the depth, q be the address utilization ratio and p be the probability of generating a child. In the random tree, any non–leaf node will have an expected number of fp children. So, the expected number of leaf nodes of the entire tree is $(fp)^{d-1}$.

Therefore, for a given tree with f and d, we can calculate the necessary p to get a ratio q in the following way.

$$\begin{aligned} \frac{(fp)^{d-1}}{f^{d-1}} &= q \\ (d-1)\ln p &= \ln q \\ p &= e^{\frac{\ln q}{d-1}} \end{aligned} \tag{1}$$

1 Presently, the implementation of *NIx–Vector* aggregation in *GTNetS* is work–in–progress and is not complete.

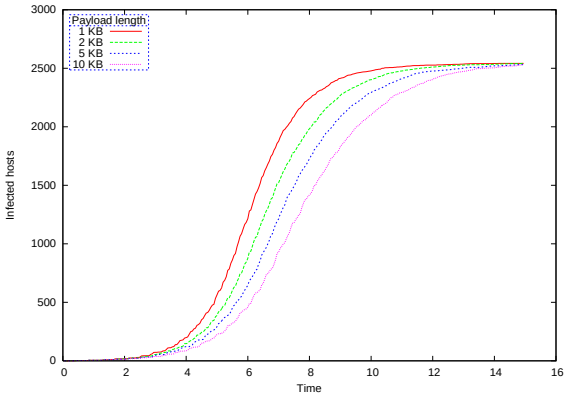

Figure 1. Effect of Payload size for TCP

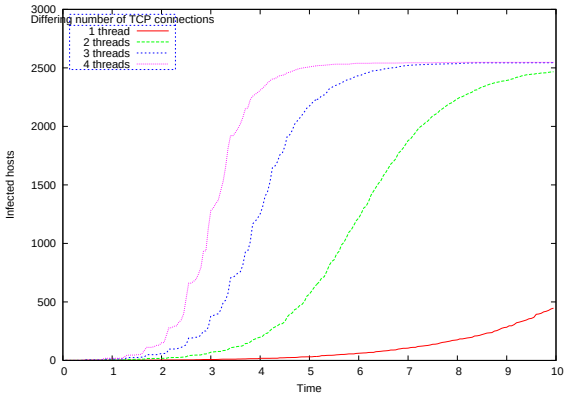

Figure 2. Effect of Parallel Connections

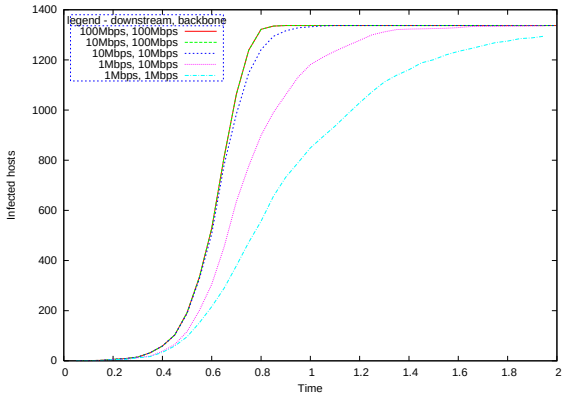

Figure 3. Effect of Bandwidth of links

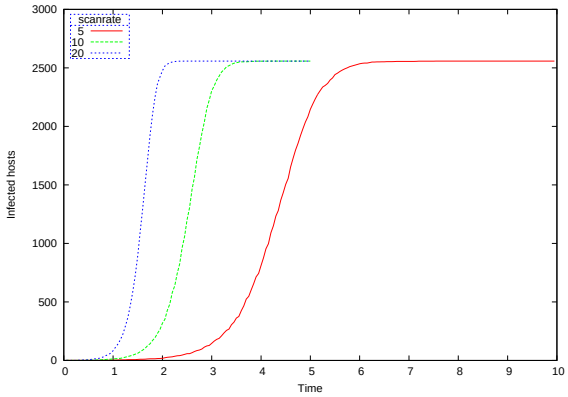

Figure 4. Effect of Scanrate for UDP worms

So equation 1 shows that the value of p does not depend on the fanout of the tree. If it is required to generate a random tree having 25% of the addresses used and a depth of 5, we set $q = 0.25$ and $d = 5$ in equation 1 and get the resultant value of $p = 0.707$.

5. Experimental Results

In this section, we give some preliminary results for worm infection rates for both *UDP* and *TCP* worms. These results are intended to illustrate the capabilities of our *GT-NetS* worm models, and not intended to show a comprehensive study of worm infection rates.

The results in figure 1 show the effect of the *TCP* payload size on on the infection rate. As expected, worms with larger payloads will spread less quickly, due to more round trip times elapsed during the payload forwarding. Results in figure 2 show the effect of the number of simultaneous *TCP* connections on the overall worm propagation. Again, as expected, the worms with fewer connections are less effective.

The results in figure 3 demonstrate the effect of link bandwidth on *UDP* worm infection rates. *UDP* worms are typically limited by the first–hop link bandwidth, and the results show this is the case. Slower links results in slower infection rates. Figure 4 shows that *UDP* worms that send infection packets at a higher rate (within the constraint of the first–hop link bandwidth) will spread faster than those with slower scan rates. Finally, figure 5 shows that *UDP* worms with larger payloads will propagate more slowly than those with small payloads, although this is not as pronounced as the *TCP* worms with larger payloads. This is due to the fact that *UDP* worms do not wait of acknowledgments and thus are largely independent of round trip times.

6. Summary

We have introduced our worm models for the *Georgia Tech Network Simulator*, and discussed a number of efficiency issues that must be addressed in any simulation environment designed to perform packet–level simulations of Internet worms. A number of novel features for our simula-

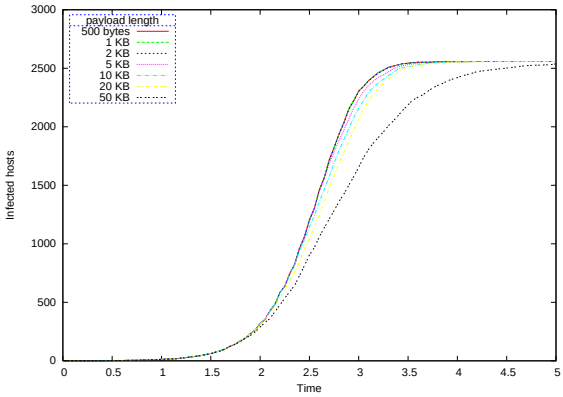

Figure 5. Effect of Payload size for UDP

tor have been designed and developed, and have lead to the capability to perform simulation studies of Internet worms in realistic environments.

We have not yet exploited the parallel and distributed simulation features of *GTNetS* for this research, but will do so in the near future. Our current success in modeling moderately large topologies with the sequential version of *GTNetS*, coupled with earlier success modeling extremely large topologies with *GTNetS*, leads us to be optimistic about future Internet scale worm models and simulation studies.

References

[1] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *Proceedings of INFOCOM '2003*. IEEE Computer Society, 2003.

[2] C. Cowan, F. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows: attacks and defenses for the vulnerability of the decade. In *DARPA Information Survivability Conference and Exposition 2000*, Jan 2000.

[3] J. Cowie, H. Liu, J. Liu, D. Nicol, and A. Ogielski. Towards realistic million-node internet simulations. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, June 1999.

[4] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global internet. *Computing in Science and Engineering*, January 1999.

[5] D. J. Daley and J. Gani. *Epidemic Modeling: An Introduction*. Cambridge University Press, 1999.

[6] R. Fujimoto, G. Riley, K. Perumalla, A. Park, H. Wu, and M. Ammar. Large-scale network simulation: How big? how fast? In *Proceedings of Eleventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'03)*, Oct 2003.

[7] J. O. Kephart and S. R. White. Directed graph epidemiological models of computer viruses. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1991.

[8] M. Liljenstam, D. Nicol, V. Berk, and R. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *Proceedings of the 2003 ACM workshop on rapid malcode*, Oct 2003.

[9] S. McCanne and S. Floyd. The LBNL network simulator. Software on-line: http://www.isi.edu/nsnam, 1997. Lawrence Berkeley Laboratory.

[10] D. Moore, V. Paxon, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy Magazine*, 1(4), July 2003.

[11] D. Moore, C. Shannon, and K. Claffy. Code–Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the second ACM workshop on Internet Measurement*, Nov 2002.

[12] G. F. Riley. The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 5–12. ACM Press, 2003.

[13] G. F. Riley, M. H. Ammar, and R. M. Fujimoto. Stateless routing in network simulations. In *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, August 2000.

[14] G. F. Riley, M. H. Ammar, and E. W. Zegura. Efficient routing using nix-vectors. In *2001 IEEE Workshop on High Performance Switching and Routing*, May 2001.

[15] A. Wagner, T. Dubendorfer, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *Proceedings of the 2003 ACM workshop on rapid malcode*, Oct 2003.

[16] N. Weaver, V. Paxon, S. Staniford, and R. Cunningham. Internet worms: past, present and future: A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on rapid malcode*, Oct 2003.

[17] C. Zou, L. Gao, W. Gong, and D. Towsley. Information warfare: Monitoring and early warning for Internet worms. In *Proceedings of the 10th ACM conference on Computer and communication security*, Oct 2003.

[18] C. Zou, W. Gong, and D. Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *Proceedings of the 2003 ACM workshop on rapid malcode*, Oct 2003.