Emulating sequential scanning worms on the DETER testbed [*]

L. Li, I. Hamadeh, S. Jiwasurat, G. Kesidis, P. Liu and C. Neuman
Pennsylvania State University, University Park, PA 16802
lli,pliu@ist.psu.edu, hamadeh,soranun@cse.psu.edu, kesidis@engr.psu.edu
University of Southern California, ISI
bcn@isi.edu

Abstract

Internet worm security threats have increased with their more advanced scanning strategies and malicious payloads. In this article, we extend our existing KMSim worm model to account for the self-destructive or removal/death behavior of worms. The modified model is then used to simulate the Witty and Blaster worms. Also in this paper we describe our experience of running worm emulation experiments on a clustered network testbed (DETER) and introduce the associated experiment specification and visualization tool (ESVT). The virtual node approach of network scaling-down and the design of Internet scan injection problem are presented with the example of the Blaster worm. Preliminary experimental results of Blaster enterprise network emulation are reported as well.

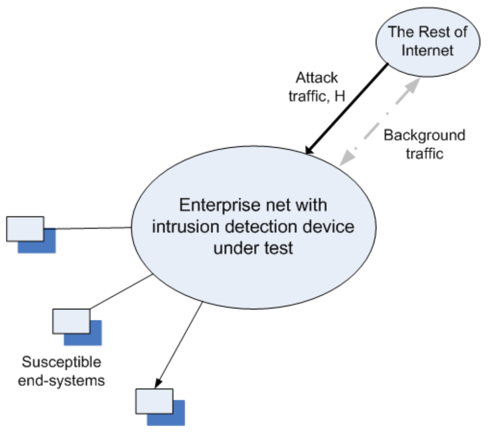


Figure 1. Simulation set-up for a worm defense DUT

1. Introduction

The objectives of DETER (The Cyber Defense Technology Experimental Research) project together with EMIST (Evaluation Methods for Internet Security Technology) project are to "build an effective experimental and testing environment and to develop a corresponding experimental methodology, for Internet security issues and defense mechanisms" [1, 2]. A testbed supplies a flexible environment with which researchers can configure arbitrary network topologies and test new protocols or security defense technologies, which also can be used for the purpose of worm simulation and emulation. The currently deployed DETER testbed has more than 200 experimental nodes, which are connected by fast switches locally and a high-speed link between ISI and Berkeley sites. Node connections are formed by a *LAN* or *link* which is actually constructed using the *VLAN* feature of the switch. Basic static routing is supported and more advanced routing can be implemented by running special routing software on the intermediate nodes. In [8], we reported our enterprise Slammer worm emulation results from experiments running on the DETER testbed.

Scanning worm defenses are expected to be deployed in peripheral enterprise networks [11, 13]. Several detection and response strategies for defense against and containment of such worms have been proposed, e.g., [20, 21, 22, 23]. Evaluation of such defenses could involve a simulation set-up such as that depicted in Figure 1 wherein: intra-network worm spread could be recreated using actual malware (on the DETER testbed [1, 2]), background traffic would need to be generated to accurately estimate false positives [24], and external attack traffic H representing scanning activity from outside directed at the network would need to be realistically modeled. Previous work on this specific problem in the context of the Slammer worm includes [12, 9, 8]. In this paper, we will extend this model in order to capture characteristics of other worms such as Witty and Blaster. The emphasis here is on the aspect of worm death or removal, the last chain of the Susceptible-Infection-Removal (SIR) model. Slammer was benign to its host but Witty, also a single-packet UDP worm, attempted to crash its host after 20,000 scans by overwriting a sector of its host's disk. The Blaster worm propagated via TCP, infected many more hosts, but scanned at a much slower rate per host. Blaster powered down its host and generally spread slowly enough to allow time for significant human counter-measures prior to its peak infection state.

In this paper, we also describe our experience emulating TCP worms using our virtual node framework (modeling packet transmission timing of TCP) together with UDP packet/message exchange on the testbed. This extends our

[*]This work is supported by both the NSF and DHS of the United States under NSF grant number 0335241.

work reported in [8], which we will briefly review here. We will go on to argue the advantage of using a UDP platform to emulate a TCP worm and report the results of our approach for the case of Blaster [7].

This paper is organized as follows. Section 2 is the related work. In section 3 the existing KMSim worm model is extended to incorporate the worm removal/death characteristics. The extended model then is used to simulate the Witty worm and Blaster worm. In section 4, we will explain how to run a network simulation or emulation experiment on the DETER testbed and the tools (ESVT-Experiment Specification and Visualization Tools) developed to accommodate such experimental activities. Section 5 details our design rationale for using UDP protocol to emulate a TCP based worm and the design of Internet scan injection program. The Blaster worm emulation results and the visualization tools are presented in section 6. In the final section we conclude the paper and list some future research topics.

2. Related Work

The success of the simple Kermack-McKendrick epidemic model for certain Internet worms, e.g., Code Red, was demonstrated in [17, 6, 4, 5, 18]. The limitation of Kermack-McKendrick model when dealing with Internet worms having special characteristics is further made evident in [6]. Its authors then propose a two-factor worm model with two modifications. The first is the decreased infection rate caused by network congestion as the worm propagates, and the other is the removal of susceptible together with infected hosts. The similar emphasis on the removal of infected hosts, together with infection delay and the re-infection of patched or inoculated infected hosts, are the topics discussed in [16]. In [5], a new detection idea, which monitors the trend of scan traffic change instead of simple scan rate change to detect worms, is proposed and argued to be able to detect a worm attack earlier. The algorithm proposed in that paper can also be used to predict the overall vulnerable population size with observation data on a small fraction of the IP space. In [17] the true possibility of the emergence of an ideal hypothetical worm that propagates most efficiently and rapidly and defies most advanced defense scheme is demonstrated and its implication is discussed. In [9], a variation of the Kermack-McKendrick mathematical model that could account for the access-link saturation caused by Slammer's scanning traffic is reported.

As for the simulation of worms, in [15], proprietary worm simulations are done but only coarse-grained global worm propagation activities are simulated with very high level abstraction of the worm propagation environment. In [18], SSFNet is used to simulate realistic network worm traffic for worm warning system design and testing, but only at an abstract network level.

Several worm simulation/emulation experiments have already been conducted on DETER and the need for advanced worm experiment utilities is identified. In [12], the idea of using scale-down to explore worm dynamics is investigated via DETER simulations. In [14], a hybrid quarantine defense is proposed and validated by both NS2 simulations and DETER emulations. The preliminary experiments show that reproducing enterprise worm propagation in a testbed environment using emulation and simulation is feasible and can yield detailed results that simulation alone cannot accomplish.

3. Worm Modeling

As in [8, 9], let:

- J be the number of different groups of peripheral enterprise networks in the Internet,
- σ_{ji} represent the total scanning rate *out* of the enterprise to the rest of the Internet of a group-j enterprise network with i infectives (i.e., infection-level i)
- $y_{j,i}(t)$ be the number of group-j enterprises with infection-level i,
- $C(j)$ be the maximum infection level of group-j enterprises, and
- the total scan rate *to the Internet* of the worm be

$$S(t) \equiv \sum_{j=1}^{J} \sum_{i=1}^{C(j)} \sigma_{j,i} y_{j,i}(t).$$

The likelihood that a particular susceptible is infected by a scan is $\eta = 2^{-32}$ (purely random scanning in the 32-bit IPv4 address space). The likelihood, therefore, that a scan causes an enterprise in state i at time t to transition to state $i+1$ is $(C(j)-i)\eta$ because there are $C(j)-i$ susceptible but not infected nodes in the enterprise at time t. Thus, define the infection "rate" of an enterprise in state i by

$$\beta_{j,i} \quad \equiv \quad S(t)\eta(C(j)-i).$$

The time-evolutions of the states $y_{j,i}$ are governed by the following coupled Kermack-McKendrick equations: For times $t \geq 0$,

$$\begin{aligned} \dot{y}_{j,C(j)}(t) &= \beta_{j,C(j)-1} y_{j,C(j)-1}(t), & (1)\\ \dot{y}_{j,i}(t) &= (\beta_{j,i-1} y_{j,i-1}(t) - \beta_{j,i} y_{j,i}(t)) \;\; i \in [1, C_j) & (2)\\ \dot{y}_(j,0)(t) &= -\beta_{j,0} y_{j,0}(t). & (3) \end{aligned}$$

The total number of worms (infectives) at time t is clearly

$$\sum_{j=1}^{J} \sum_{i=1}^{C(j)} i y_{j,i}(t). \qquad (4)$$

Thus, the scan-rate per worm (per infective) is

$$\frac{\sum_{j=1}^{J} \sum_{i=1}^{C(j)} \sigma_{j,i} y_{j,i}(t)}{\sum_{j=1}^{J} \sum_{i=1}^{C(j)} i y_{j,i}(t)}. \qquad (5)$$

The dynamics given at the end of Section 3 of [9] can be generalized to account for "removals/deaths" by modifying:

$$\begin{aligned} dy_{j,i}/dt &\;-=\; \delta_i y_{j,i} \\ dy_{j,i-1}/dt &\;+=\; \delta_i y_{j,i} \end{aligned}$$

where the removal/death rate is $\delta > 0$ and $\delta_i \equiv i\delta$.

In the next step, we will use the modified model to simulate Witty and Blaster worms.

3.1. Witty Computational Simulation

The Witty worm, which exploits a buffer overflow vulnerability in several security products, broke out on March 19, 2004 and caused considerable damage to the Internet and infected computers. It stands out from other recent wide-spread worms by its distinctive features such as a malicious payload and the shortest interval between the publishing of security vulnerability and worm release [3]. We study the Witty worm here for its marked self-destructive behavior and relatively well-documented propagation trace (manifested by a number of trace logs by a number of network telescopes and blackhole monitors) so we can compare our simulation results with them.

We added a removal module into the KMSim simulation program according to the mathematical model in the last section. The code can be expressed in the following pseudo-code. [1]

Algorithm 1 Removal/Death of the Infected and Susceptible

```
1:  for $j$ do
2:      $newdeath \Leftarrow \delta * ce_j * y_{j,ce_j} * dt$
3:      $death_j += newdeath$
4:      $y_{j,ce_j} -= newdeath$
5:      for $i = ce_j - 1$ downto 1 do
6:          $y_{j,i} += newdeath$
7:          $newdeath \Leftarrow \delta * i * (y_{j,i} - newdeath) * dt$
8:          $y_{j,i} -= newdeath$
9:      end for
10: end for
```

Before running the KMSim simulation program to simulate a specific worm, we need to determine the number of different groups of peripheral enterprise networks in the Internet j, the maximum infection level in each enterprise group $C(j)$, the number of enterprise networks in each group $ne(j)$, and the scanning rate of enterprise network in each infection level $\sigma_{j,i}$. The signature of bandwidth-limited worm can be simply simulated by setting the $\sigma_{j,i}$ to be a constant or with an upper limit, e.g., $bandwith/packetsize$ in the case of Slammer. Finally we need to set the removal/death rate of the worm to make the modified worm model to be effective.

One method to get such parameters is to use collected worm scanning logs by those network telescopes and blackhole monitors. But two reasons make these scanning/trace logs not directly usable. First, there are considerable discrepancies among these logs, which were reported and analyzed in [7]. Second, the numbers of infectives in those categories of enterprise networks in the logs are cumulative numbers, instead of point values which can be directly plugged into the model. For these reasons, we employed a simple method to estimate the number of susceptible hosts and scanning rates, based on those scanning logs.

- The total number of susceptible hosts. It is impossible to know exactly how many hosts were vulnerable to the Witty worm. We used the number 12,000 from CAIDA [3] as the maximum number of the susceptible hosts.
- j, $C(j)$, and $ne(j)$. To make the simulation simple, we set the number of enterprise groups j to be 1, i.e., there is only one group of peripheral enterprise networks. Further, the maximum number of infection level $C(1)$ is set to be 4, which is based on the belief that there is limited installation of Witty target application–firewall products. So the number of enterprises $ne(1)$ can be derived to be $12000/4 = 3000$.
- Scan rate $\sigma_{j,i}$. The Witty worm is a typical bandwidth-limited worm, which means every infective sends out scanning packets as fast as its network connection possibly permits. According to [3], we decided to set the enterprise network scan speed to be linearly distributed between 1800pps and 2400pps, which correspond respectively to the case of one single infective and that of full infection.
- Removal/death rate δ. The removal/death rate of the Witty worm can be estimated by using the decreased number of infectives in the trace logs divided by the time interval. But again, this method generates inconclusive results. In the actual simulation, we tried a number of different rate values and chose one whose simulation result fit best with the trace data.

Using this modified KMSim simulation program we run a simulation of the Witty worm. The simulation result was drawn together with the actual infection data reported by CAIDA in Figure 2. We can see that the modified KMSim worm model can replay the Witty worm propagation in high fidelity and the peak value of infective number by KMSim simulation is only slightly higher than the real number.

Figure 3 is the zoomed view of the overall worm propagation curve. Since the Witty worm reached its peak infection level 4000 seconds after its inception, it is worth perusing its behavior in the beginning stage of its overall life cycle. We can see from the figure that the actual propagation curve is at the left of the simulated curve, which means the real worm spreads faster than that simulated by the KMSim model. We believe that the reason for this difference is that in the KMSim model, for the purpose of generality, we set the initial number of infected to be one, while the real Witty worm "used

[1] Note that the code is only for the illustration of the calculation of removal/death and corresponding parameter adjustment, not the actual simulation code itself.

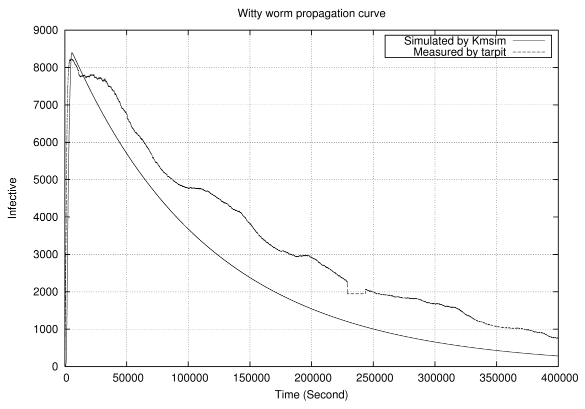


Figure 2. The Witty Worm Simulation Result

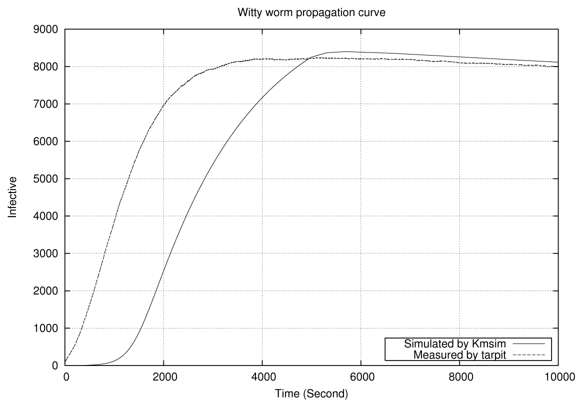


Figure 3. Witty Simulation Result: Zoomed view

either a hitlist or previously compromised vulnerable hosts to start the worm." [3] When we change the number of initial "seed" worms in the KMSim model to a larger one, the simulated curve will move left and match the actual Witty curve.

3.2. Blaster Characteristics and Modeling

Next we use this modified model to simulate the Blaster worm. Blaster infected more hosts than Slammer and Witty did and caused more severe damage to the affected enterprise networks. It exploits the DCOM RPC vulnerability of Windows and targets only machines installed with Windows 2000 and Windows XP. The Blaster worm has a more "advanced" propagation strategy than those of Slammer and Witty worms. It chooses its initial target address that is within the same local /16 with 40% probability or a random address within the whole IPV4 space with the other 60% probability. After the initial address is decided, the worm begins scanning sequentially from that address in batches of twenty IPs a time. Though its propagation algorithm is not completely random, researchers have been able to model its propagation using simple epidemic models. [5]

Since the Blaster worm sends probing packet to a well-known target of vulnerabilities–TCP port 135, which was blocked by many network telescope operators during the time of its spread, there is little detailed report or no consensus on those parameters we are interested such as the susceptible population size, scan rate, etc. We made efforts to model Blaster using the modified KMSim model as best as possible.

- The total number of susceptible hosts. The number of infected hosts was estimated from 30,000 to several hundred thousands. Based on Symantec estimate [19], we set the total susceptible host number to 200,000.

- j, $C(j)$, and $ne(j)$. The number of classes of enterprise networks was again set to 1. Because Blaster targets Windows XP and Windows 2000 machines, the number of susceptible hosts in each enterprise network should be larger than that of Witty. So the maximum number of infection level $C(1)$ was set to be 20. The number of enterprise networks $ne(1)$ was 10,000.

- Scan rate $\sigma_{j,i}$. Blaster may be bandwidth-limited if there was a pretty large percentage of hosts vulnerable to the Blaster worm in the enterprise network. Repeated rebooting of infected hosts and OS "patching" by humans may have alleviated such effect, however. We know from the worm code that it sends out 20 scan attempts a time and sleeps 1.8 seconds between two batches. And the rate of scan per single infective should be roughly equal to 10pps. Since an unsuccessful attack of the Blaster worm causes the host machine to crash and/or reboot, the actual scan speed may be slower than 10 in average. So we set the scan rate of enterprise network to be between 5 and 110, corresponding to the various infection levels. [2]

- Removal/death rate δ. The removal/death (or patching, for this case) rate of Blaster is difficult to estimate. We used a small number 1.034e-08.

Plugging these parameters into the KMSim model, we got the simulation result showed in Figure 4. This simulation shows that the worm reached its peak at about 10 hours, and infected about 190,000 hosts at that time. After that, the number of infected hosts decreased slowly.

4. Testbed Emulation Set-up And Virtual Node Design

In [8], we reported our virtual node approach to leverage limited testbed resource to emulate worm propagation in a large enterprise network. Here we briefly review this approach and other necessary steps to run an emulation/simulation experiment on the DETER testbed. At the same time and in the following sections, we will introduce how to use our ESVT

[2]A more careful simulation will consider the effect of local scan rate.

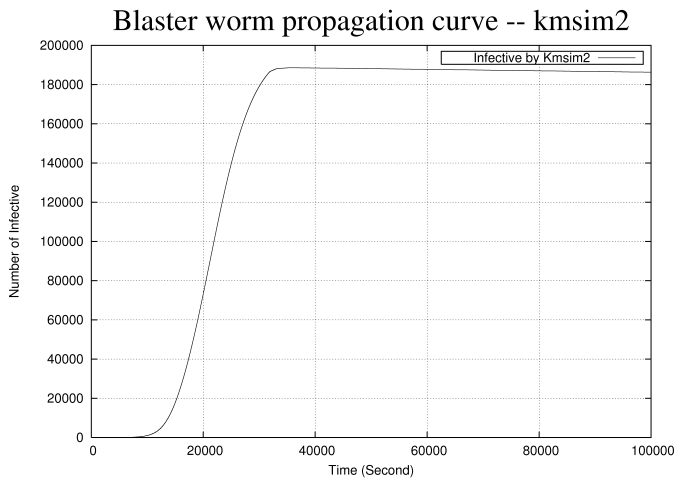


Figure 4. The Blaster Worm Simulation Result

graphical interface tool to help finish these jobs and analyze the experimental results.

4.1. Experiment Topology

We want to conduct a detailed worm propagation experiment using emulation & simulation method to gain first-hand knowledge of such propagation on enterprise networks. The results and experience of this experiment will be the solid foundation for the following worm detection and defense experiments. We decided to choose a 1000-node network (Oregon State university network as the prototype) as our experiment topology to conduct this enterprise network worm experiment on DETER. There are six internal routers, one central switch, and one border switch in this topology.

4.1.1. Setup the Experiment Using ESVT Tools. ESVT GUI supplies an integrated environment to conduct this interactive worm experiment. It is a component based topology editor, script generator, worm experiment designer, and a visualization tool of experiment results. At the first step it can be used to draw the topology based on the prototype network. The toolbox of program includes network components such as computer/host node, switch, router node, network/Internet interface, and link. Components have configurable properties such as bandwidth and link latency, which can be modified by simple mouse clicks and field editing. Computer node can be defined as susceptible or non-susceptible. A number of convenient features like "copy & paste" and "component finder" are developed to help design this rather large network topology.

Figure 5 is a screen-shot of this topology overview. In the topology, the node indexed 983 is the Internet interface node, and the two nodes indexed 942 and 955 respectively are dark address scan detectors or honeypots, which monitor worm's scan attempts to those "unused" IP addresses. To observe the

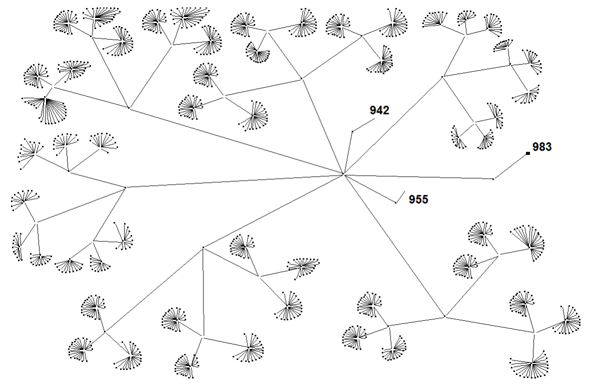


Figure 5. Finished network topology by ESVT

worm's propagation in a network with high susceptible density, we set the population size of susceptible nodes to be 50% of the total hosts.

To request network resource from testbed and apply network topology on the testbed, a NS (network simulator) style TCL script file needs to be submitted to the testbed control plane. For people unfamiliar with TCL language or specific testbed requirements, writing such a script is not an easy job. ESVT GUI has TCL script generation function that can output network topology into NS or DETER TCL script to simplify this task. The basic procedure of topology conversion is that every node (except virtualized LAN segment) in GUI topology is mapped to one node in the testbed. Links between non-switch nodes including computer node, router node, and Internet node in GUI topology are still links in the TCL file, while links connecting switch nodes and other nodes in GUI will be replaced by TCL LAN-making script *make-lan* which takes all LAN nodes as parameters. Script generator also inserts additional experiment configuration script specifically for worm experiments, which are translated from user definitions on the GUI. Some specialities of this script generator are:

- Virtual sub-network nodes. LAN segment (level 1 or level 2 switch node and LAN connected with it) that is marked to be virtualized during topology drawing will become one node in the generated final script file. On this node a specific virtual LAN program designed by us will run and the virtualized sub-network topology information and host susceptibility information can be read from the GUI generated map file (One such example in Figure 6) during the program start-up time.

- Internet interface. A special program will be run on this node to simulate the rest of the Internet. The simplest implementation of such program will be a traffic sink that only receives packets. A specific Internet interface design utilizing our worm simulation result is described in Section 5.

- Normal and vulnerable nodes. Background traffic generator will be run on normal, non-susceptible nodes. For

```
#network address/prefix
10.1.1.1/16
#node & virtual node map file
#n-## TYPE(B/I/V/R) S/N ##(GUI node index) #(Last segment of IP)
n-902  V N 29   254
n-902  V N 27   253
n-902  V N 32   252
n-902  V N 36   251
n-902  V N 38   250
n-902  V N 40   249
n-902  V N 43   248
```

Figure 6. Virtual LAN topology data file: Each line represents one virtualized node (with exception of bandwidth and extra lines), five columns are switch node index, line type, computer node susceptibility, computer node index, and last byte of node IP address.

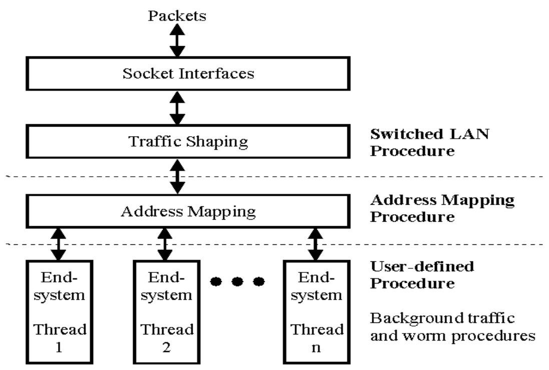


Figure 7. Programming layout of virtual node model

vulnerable nodes specially designed program will simulate vulnerable service and wait to be “attacked”.

- Bandwidth, latency, addresses, OS. Those host and link properties specified by the experimenter will be translated to the TCL file as best as possible. Currently there are FREEBSD, RedHat Linux, etc., that can be chosen in the DETER testbed. For our experiment, we used the default FREEBSD for the experimental nodes.

When composing this script we consider some technical challenges as well. For example, to alleviate TCPDUMP problems (To collect the network traffic data we need to run traffic collecting program such as TCPDUMP on experimental nodes. But TCPDUMP may not be able to log every packet that actually flows through if the traffic volume is high or the CPU is busy during the experiment.), faster computers are allocated to packet forwarding nodes that are expected to carry heavy traffic load while other nodes that need less computing power are given slower machines. The priority of TCPDUMP process may be adjusted to run at a higher rank.

4.2. Virtual Node Design

Employing a one-to-one emulation approach entails substantial resources that a normal testbed cannot support. Both the Emulab and DETER testbeds have about 200 nodes. To emulate our 1000-node enterprise network using those testbeds, we need some kind of scale-down or virtualization while keeping the fidelity level high. In [8], we compared our virtual node design with other kinds of virtualization methods such as VMWare and Emulab VM and concluded that the performance of the virtual node design in realistic LAN simulation is comparable with the all-real-node scenario, while consuming much less resource than other virtualization approaches. In the following, we will briefly introduce our virtual node design.

4.2.1. A Virtual Node for a Peripheral LAN. Figure 7 shows the programming layout of virtual node model. The virtual node, a multi-threading application on a single CPU, consists of a virtual switch, a number of virtual end-systems, and a background traffic generator. The virtual switch is simply a token bucket traffic regulator. Virtual end-systems are implemented by threads, and each is attached with a background traffic generator. In addition, user can attach a user-defined worm code to a susceptible virtual end-system in order to generate scanning traffic.

4.2.2. End-system Threads. Both a background traffic generator and user-defined worm procedures are functions that are assigned to and constantly executed by end-system threads to generate normal and worm scanning traffic. The background traffic sending rate is limited to 10% of the upstream link bandwidth of each end-system. The design of Blaster worm thread is given in the next section.

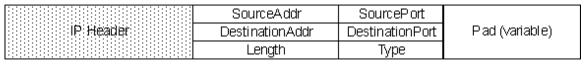
4.2.3. Address Mapping. There are two types of addresses used in the virtual node program, namely a *virtual* IP address of an end-system and an *actual* IP address of a DETER experimental node. The virtual IP address is an identifier of a virtual end-system on a virtual LAN while the actual IP address is a physical IP address of a experimental node in the DETER network. A unique IPv4 address is assigned to each virtual end-system, and the network addresses of virtual end-systems are mapped with the DETER experimental node IP addresses and kept in an intra-enterprise address mapping table. Figure 8(a) shows an example of an intra-enterprise address mapping table.

Traffic among DETER experimental nodes is communicated by the actual IP addresses. But virtual IP addresses are only known to the virtual node program, not to the other DETER emulated-network devices. Therefore, a virtual IP address has to be translated into a testbed IP address according to the intra-enterprise address mapping table so that the scanning traffic can be sent across the DETER network. In Figure 8(b), we show where both virtual and actual IP addresses are located in an IP packet.

The *length* field contains the size of packet, not including the actual IP header. The *Type* field indicates whether the packet is a worm packet or a normal packet. Finally, the

Network Address	DETER Node Address
10.1.3.0/24	10.1.3.2
10.1.4.0/24	10.1.4.2
10.1.5.0/24	10.1.5.2
10.1.6.0/24	10.1.6.2
default	10.1.8.2

(a) An intra-enterprise address mapping table



(b) A virtual node IP packet structure

Figure 8. Address mapping and IP packet structure

variable-length *Pad* field is used to fill up the payload to create a packet according to the virtual header length field.

4.2.4. Traffic Shaping for a switched LAN. We used a simple mathematical model for shaping traffic of a switched LAN in our virtual node model. The throughput of a switched LAN is bounded by only the upstream link bandwidth capacity. If we assume that n end-systems are transmitting at full rate R, the throughput is bounded by a nondecreasing $R \cdot n$. Therefore, the traffic volume on the upstream link of a switched LAN is determined by taking the minimum of its upstream link bandwidth C and the aggregate throughput of the end-systems it connects.

In order to send a packet, a virtual end-system has to acquire a token (mutex). Let P represents a sending packet size, and C is the outbound sending speed of the upstream link bandwidth. When the token is not available, the packet from the virtual end-system will be dropped and the virtual end-system thread will be stalled for P/R time-unit before it sends a new packet. However, if the sending virtual end-system successfully acquires the token, the packet will be sent out, the token will be hold by the sending end-system for P/C time-unit, and the virtual end-system thread will be stalled for P/R time before sending a new packet.

5. Design for Blaster Emulation

In this section, we present our design rationale for using UDP to emulate the TCP based Blaster worm and the design of Internet scan injection program.

5.1. Using UDP to Emulate the Blaster Worm

Different from the Slammer worm which accomplishes both the target probing and infection by one UDP packet, Blaster communicates with potential targets by TCP protocol. A successful infection has to go through the procedure of connection set-up by three-way hand shake, malicious code transfer, and connection tearing-down. (The actual procedure is more complicated which involves more than one TCP conversations.) But the virtual node program was written to support UDP communication only, so we faced a choice between TCP and UDP when we planed to utilize the existing virtual node program to emulate the Blaster worm. The following considerations were weighted in our decision process.

⊙ The objective of emulation. Our objective is not to study the exploitation itself, i.e., how the worm probes the vulnerability of a real host application and compromises it. We are interested in how the worm spreads itself in a typical enterprise network by means of target selection, stealth or rapid scanning, or in other words, emphasis on the propagation. For such purpose, the emulation of Blaster using UDP packet exchange is appropriate and sufficient to get the data we want.

⊙ Extent of Fidelity. With the worm propagation as the principal emulation objective, the extent of matching between emulation and real world situation in such aspects as traffic volume, round trip time, etc., is not of most dominant concern any more, as long as the emulation does not distort the actual traffic dynamics in the process of worm propagation so much that the worm's spreading behavior begins to deviate from the real case. We believe in the case of Blaster where the aggregated traffic is not high enough to congest the enterprise network, the use of UDP platform will not violate this principle.

⊙ The limitation of testbed resource. As stated in the previous section, our virtualization approach uses one single testbed machine to simulate a LAN with up to 255 virtual hosts. At the same time, each TCP connection, including its initialization and maintenance, consumes lots of CPU and memory resource: large sending and receiving buffers, numerous status variables, several transmission timers, etc. If we used TCP protocol to emulate Blaster in our virtual node program, the program would have had to open thousands of TCP connections at any time (worm scan&infection and background traffic), which will probably overwhelm the slower testbed machines with CPU speed as low as 733MHZ. On the other hand, UDP protocol is stateless and consumes much less resource, which will exert little stress on the testbed machines and guarantee a smooth emulation.

⊙ Design of virtual node program. Using UDP also makes the design of the virtual node program easier because we can inherit the existing design and make fewer changes. Otherwise we have to implement a NAT-like mechanism to support TCP conversation.

After these considerations, we decided to use UDP platform to emulate the Blaster worm, or a hypothetical sequential scanning UDP worm. Algorithm 2 of active worm thread in the virtual node program is an illustration of such design.

5.2. Design of the Internet Scan Injection

The problem of Internet interface design is to recreate the scanning traffic of the Blaster worm directed to the enterprise network under test by the rest of the Internet. Since there is no easily available total scan-rate data of Blaster from network telescopes, we decided to use the simulation results from our modified KMSim model.

Suppose that the total scan rate, $S(t)$, of a worm is obtained, under the assumption of a uniform distribution, the scan-rate from the Internet directed at the enterprise under simulation

Algorithm 2 Sequential Scanning Worm Body

```
    repeat
2:    for $j = 1$ to 20 do
        Calculate Next IP $tip_j$
4:      $pkt.destinationIP = tip_j$
        Send UDP packet
6:    end for
      Sleep 1.8 seconds
8:    for $j = 1$ to 20 do
        if $tip_j$ is infected then
10:       pkt.payloadlenth = BLASTERSIZE
          $pkt.destinationIP = tip_j$
12:       for $l = 0$ to 8 do
            Send UDP packet {Simulate the actual worm infection}
14:         $SendingDelay = (wormpacketsize + 28) * 80;$
            Delay $SendingDelay$
16:       end for
          Sleep 1 second
18:     end if
      end for
20: until Program Is Active
```

could be approximated as $(A/2^{32})S(t)$, where A is the size of the address space of the enterprise network. For a random scan worm, the target of individual scans directed to the enterprise network could be easily chosen at random. But to maintain or replay the salient feature of the Blaster worm, in which every infected victim scans sequentially from the initial IP forever, we have to craft both the source (scanning) IPs and the destination IPs carefully.

First, we divide the total infective count by 2^{16} (the total number of hosts in the /16 network) to get the number of active scanners that send scan traffic into the enterprise in the current emulation time. Then, each scanner will send scan traffic based on the scan_per_second input from KMSim model. If the scan_per_second of a scanner is less than 20, i.e., the original scanning rate of Blaster victims, we use a random function to select target addresses so that the number of scans actually sent out is scan_per_second, while the target IP addresses still increase by the rate of 20 per cycle. Note that this mechanism is just for sending probing or SYN packets. For the worm data packets, they will be sent if only the host at the target address is deemed infected (by UDP message feedback or simple global status table look-up). The number of scanners and the scan rate of each scanner per time cycle are adjusted based on the input of KMSim simulation results. If the target IP address of any active scanner goes outside the address space of the enterprise network, that scanner will be removed from the scanner table and a new one will be generated with the source IP chosen randomly from the IPV4 space and an initial target IP chosen randomly from the /16 address space.

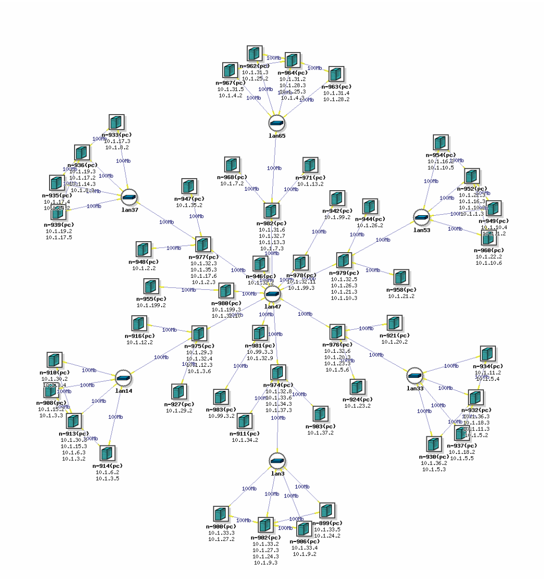


Figure 9. Experiment Topology Viewed in DETER

6. Testbed Emulation Results

With the topology and experiment specification TCL file, virtual node program, and the Internet interface program ready, we run the enterprise Blaster propagation experiment on the testbed. The actual testbed resource utilization and topology view can be seen in Figure 9.

The Blaster worm spreads much slower than the Slammer worm: it took it at least 10 hours to reach the peak point according to our simulation. We cannot run an emulation experiment to last that long on the testbed. On the other hand, we need not do so because it only takes a few minutes for the worm to infect the majority of a single enterprise network, depending on the distribution of the susceptible hosts and the network topology. We configured the emulation to run for 600 seconds, and chose the data window between the 89400th second and the 90000th second from the simulation result as the scanning data feed for the Internet scan injection program.

6.1. Visualize the results using the ESVT toolkit

The traffic log and worm infection log files after each experiment are rather formidable: more than 1,400 files (One TCPDUMP log file for each node/IP and one infection log file for each infected node.) with a total size at about 1 to 4 Gigabytes. How to present these data in a straightforward way is a big challenge. We believe that visualization gives a special and powerful perspective on worm analysis that any other methods can not equal at. There is such effort in EMIST GUI program development.

The GUI reads network traffic flow and worm infection dynamics from experiment log files and uses different animations or histogram charts to replay the worm propagation process and traffic dynamics. Step time of animation (data window) can be adjusted from 1 millisecond to 1 minute. The pro-

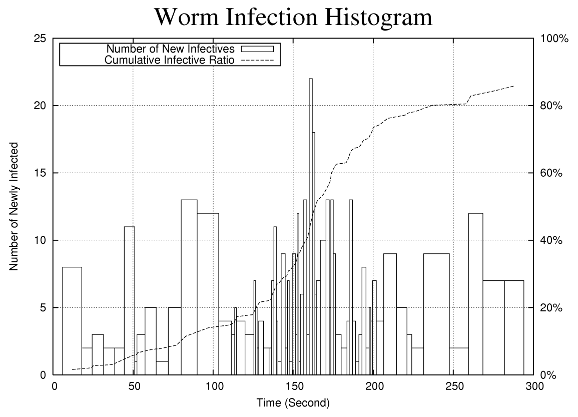


Figure 10. Blaster's propagation in the enterprise network

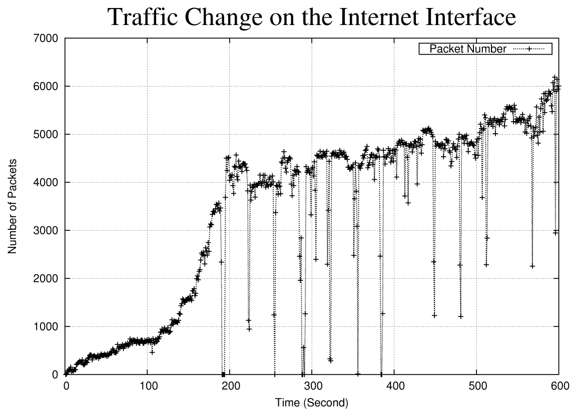


Figure 11. Traffic change observed on the Internet interface: the number of scanning packets increases

gram scans all TCPDUMP files and finds the earliest packet time stamp. It then uses this time minus two steps' time as the starting time for the following calculation and statistics. The animation refreshes every second and each snapshot is a view which shows node infection status and summarizes the average traffic rate during the past time interval.

In every time step each host node will change its display color to update its current worm infection status. Gray node is non-susceptible, green node is susceptible not yet infected, and red one is infected. So the effect of worm propagation can be seen as the number of infected red nodes increases. The color of links as the representation of traffic volume changes categorically from gray color, which means trivial traffic below one percent of link bandwidth, to red color, representing heavy traffic above thirty percent. If a user is interested in detailed traffic change on one particular link, a bar chart view can be chosen to show the histogram of traffic flow on that link. Another view – worm traffic pie chart – uses simple rules to single out worm traffic and visualizes the comparison of worm vs. non-worm traffic by a colored pie. Animation can be paused and fast forwarded or rewound to a desired time so the animation can be skipped or replayed.

6.2. Discussion on the Infection and Traffic

We had run the same 10 minute experiment for a number of times and the extent of worm propagation varied. The rate of scanning from the Internet interface node s was about 6 scans per second per source IP, and the average number of simultaneously scanning IP n is 3. The total scan attempts in the 600 second experiment from the Internet was $6*3*600 = 10800$. So it is not surprising if the experiment resulted no infection in the /16 network. But once one susceptible host was infected, its neighbors would likely be infected as well and itself would begin sending scanning traffic rapidly. That was the case for the experiment from which the data in Figure 10 and Figure 11 came.

From Figure 10, we see that the first infection happened at about the 11th second, and at the 300th second, about 83% percent of total susceptible hosts were all infected. The majority of them were infected between the 120th second and the 180th second.

The traffic pattern seen at the Internet access link in Figure 11 roughly matched the infection change in Figure 10: we see a rapid traffic increase between the 110th and the 200th second, reflecting the increasing number of infectives scanning to the Internet. The average number of scan attempts per infective after the 300th second was about $4500/420 = 11$. (The exact number should be lower because the background traffic is not excluded.) The outliers or the sudden-drop-points in the figure are probably the results of TCPDUMP anomalies we mentioned earlier.

6.3. Placement of Dark Address Scan Detector

A honeypot is an effective way to monitor and collect intrusion attack behavior and information about a network since the traffic targeting these "dark addresses", i.e., unused IP addresses, is very likely malicious or at least suspicious. In our enterprise emulation experiment, we configured two virtual nodes (node 942 and 955 in Figure 5) to simulate the honeypots (/24) that passively gather scanning traffic from both outside (scans from Internet interface node) and inside (scans from infected nodes). On these honeypot nodes, Both TCPDUMP and application level traffic collecting programs (a modified version of virtual node program to log the real IP addresses stored in the packet payload) were set up to receive any traffic to these two /24 network segments. Virtual node programs recognize the existence of "dark addresses" by reading the complete topology map file so that the background traffic will skip these addresses and only worm packets will be directed to these nodes. [3]

Though under the concept of network telescope it is feasible to estimate the worm threat to the global network by monitoring the scanning activities in a limited IP space, our experience from the experiments shows that monitoring only segments of the network is not an effective way for enterprise worm early detection. In our experiment the first host was infected at the 11th second and more than 400 hosts were infected after 400 seconds, but the "honeypot" received the first scan packet at the 360th second. Admittedly, the placement of such dark address scan monitors is important. But unless they are placed at every LAN segment or all the scanning traffic is redirected to them with the help of other devices, it is not as valuable and can not be solely relied upon for early detection.

7. Summary and Future Work

In this article, we extended our existing KMSim worm model to incorporate the removal/death behavior of worms. The modified model was then used to simulate the Witty and Blaster worms. The simulation shows much improved results in the case of the Witty worm. The modified KMSim2 program will be released soon in our EMIST project web page.

Also in this paper we described our experience of running worm emulation experiments on a clustered network testbed–DETER and introduced how to use the specifically designed experiment specification and visualization tools for such experiments. The preliminary experimental results of Blaster enterprise network emulation were reported as well.

In the next step, we will continue to extend our model to those worms that had (and will have) more complex scanning strategies than those of Slammer and Witty. We will consider the "hit-list" worm or more stealth worm behaviors in the future. Utilizing our simulation framework and the testbed to test various worm containment strategies is on the agenda as well.

We thank CAIDA for their analysis of the Witty worm and data.

References

[1] T. Benzel, B. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experience with DETER: A Testbed for Security Research. in the *2nd IEEE Conference on testbeds and Research Infrastructures for the Development of Networks and Communities*, Spain, 2006.

[2] EMIST Project. *http://emist.ist.psu.edu*

[3] C. Shannon, D. Moore. The Spread of the Witty Worm, *Available at http://www.caida.org/analysis/security/witty/*

[4] Z. Chen, L. Gao and K. Kwait. Modeling the spread of active worms", In *Proc. IEEE INFOCOM*, San Francisco, 2003.

[5] C. Zou, L. Gao, W. Gong and D. Towsley. Monitoring and Early Warning for Internet Worms. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, October 2003.

[6] C. Zou, W. Gong and D. Towsley. Code Red Propagation Modeling and Analysis. In *Proceedings of the 9th ACM Conference on Computer and Communication Security*, 2002.

[7] E. Cooke, M. Bailey, Z. Mao, D. McPherson. Toward Understanding Distributed Blackhole Placement. *Proceedings of the 2004 ACM WORM Workshop*, Washington, DC.

[8] L. Li, S. Jiwasurat, P. Liu, G. Kesidis. Emulation of Single Packet UDP Scanning Worms in Large Enterprises. In *Proc. 19 International Teletraffic Congress (ITC19)*, August, Beijing, China, 2005.

[9] G. Kesidis, I. Hamadeh, and S. Jiwasurat. Coupled kermack-mckendrick models for randomly scanning and bandwidth saturating Internet worms. In *Proc. QoS-IP*, Catania, Sicily, Feb. 2005. Springer-Verlag.

[10] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 2004.

[11] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proc. IEEE INFOCOM, San Francisco*, 2003.

[12] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson. Preliminary results using scale-down to explore worm dynamics. In *Proc. ACM WORM, Washington, DC*, Oct. 2004.

[13] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proc. 13th USENIX Security Symposium*, Aug. 2004.

[14] P. Porras, L. Biesemeister, K. Levitt, J. Rowe, K. Skinner, and A. Ting. A Hybrid Quarantine Defense. In *Proc. ACM WORM, Washington, DC*, Oct. 2004.

[15] A. Wagner, T. Dubendorfer, B .Plattner and R .Hiestand. Experiences with worm propagation simulations. *ACM CCS WORM Workshop*, 2003.

[16] Y. Wang, C. Wang. Modeling the Effects of Timing Parameters on Virus Propogation. *ACM CCS WORM Workshop*, 2003.

[17] S. Staniford, V. Paxson, and N. Weaver. How to own the Internet in your spare time. In *Proc. USENIX Security Symposium*, pages 149–167, Aug. 2002.

[18] M. Liljenstam, D.M. Nicol, V.H. Berk and R.S. Gray, Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing. In *Proc. ACM WORM*, Washington, DC, Oct. 2003.

[19] Update: Blaster worm infections spreading rapidly. *Available at http://www.networkworld.com/news/2003/0812blastinfect.html*

[20] M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Annual Computer Security Applications Conference*, 2002.

[21] J. Jung, V. Paxson, A. Berger and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. in *Proc. IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May. 2004.

[22] X. Chen and J. Heidemann. Detecting early worm propagation through packet matching. Technical report, ISI-TR-2004-585, 2004.

[23] D. Whyte, E .Kranakis, and P. Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network. in *NDSS'05*, San Diego, CA, February, 2005.

[24] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer. Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic. in *Proc. Internet Measurement Conference 2005*, Berkeley, CA, Oct, 2005.

[3]The line with a 'D' value in the second field marks the "dark address" space in the virtual LAN topology file.