Lightweight testbed for evaluating worm containment systems

Lucas John Vespa

Department of Computer Science,
University of Illinois at Springfield,
Springfield, IL 62703, USA
Fax: 01-217-206-6217
E-mail: lvesp2@uis.edu

Ritam Chakrovorty and Ning Weng*

Department of Electrical and Computer Engineering,
Southern Illinois University,
Carbondale, IL 62901, USA
Fax: 01-618-453-7972
E-mail: ritam2109@gmail.com
E-mail: nweng@siu.edu
*Corresponding author

Abstract: Hazardous worms can compromise hundreds of thousands of hosts in just hours. Mitigating these worm threats requires fast and effective strategies for containment and is a difficult task. Many containment systems have been proposed including anomaly detection, address blacklisting and signature-based content filtering. Meanwhile recently developed worm models enable us to develop a testbed to quickly evaluate the efficiency of defense mechanisms. Existing testbeds either require a great deal of hardware resources, or do not account for network performance impact due to containment methods. In this paper, we present a testbed which utilizes software agents to allow large scale simulation while maintaining individual host functionality. Varying containment schemes and strategies have been evaluated using this testbed in terms of number of infected hosts and performance impacts. Our results indicate that a dynamic containment system achieves better performance and security. We believe our testbed is an effective tool to explore and evaluate varying worm containment systems.

Keywords: worm containment; network testbed; network performance.

Reference to this paper should be made as follow: Vespa, L.J., Chakrovorty, R. and Weng, N. (2012) 'Lightweight testbed for evaluating worm containment systems', *Int. J. Security and Networks*, Vol. 7, No. 1, pp.6–16.

Biographical notes: Dr. Lucas Vespa is currently an Assistant Professor at The University of Illinois Springfield in the Department of Computer Science. He has a PhD in Electrical and Computer Engineering and a MS in Electrical and Computer Engineering from Southern Illinois University Carbondale. Dr. Vespa has published articles in ACM Transactions on Architecture and Code Optimisation, The Computer Journal and the like. His current research interests include network security analysis, sensor networks and applications for SIMD processing.

After completing Bachelor of Engineering from University of Pune, India, Mr. Chakrovorty came to Southern Illinois University, Carbondale (SIUC) on January 2008 to pursue Master of Science in Electrical and Computer Engineering. He started working with Dr. Weng on a Worm Containment project from August 2008 and commenced his thesis January 2009. Mr. Chakrovorty's graduation from SIUC was May 2010 and since then he have been working as a network engineer consultant.

Ning Weng received a PhD degree in Electrical and Computer Engineering from the University of Massachusetts (Amherst) in 2005. Currently he is an Associate Professor in the Department of Electrical and Computer Engineering at Southern Illinois University-Carbondale. His research interests are in the areas of securities of computer networks and embedded systems.

1 Introduction

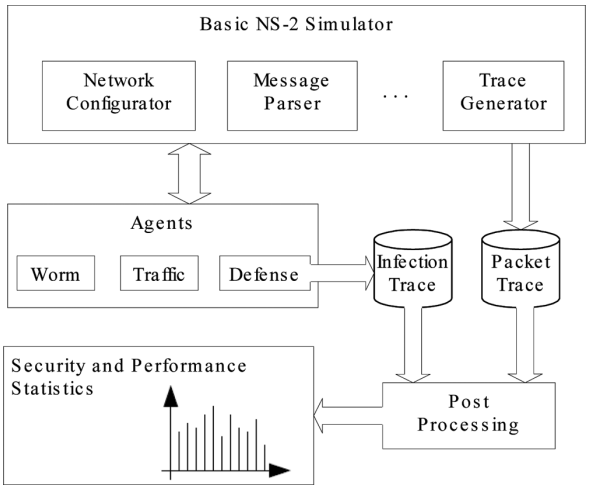
Dangerous worms like CodeRed or Slammer can spread millions of probe packets in just seconds (Cai et al., 2005) which can result in thousands of infected hosts and large losses. Therefore, fast and effective containment strategies are crucially important to protect the Internet Infrastructure. Toward this goal of fast and effective worm containment, different techniques have been presented such as address blacklisting and content filtering (Moore et al., 2003), anomaly detection (Zou et al., 2006; Weaver et al., 2004) and signature-based detection (Hyang-ah Kim, 2004; Tang and Chen, 2005; Mohammed et al., 2010; Tang et al., 2011). With the myriad of work which exists to mitigate worm threats, it is difficult to compare containment strategies. It is apparent that a tool is needed to evaluate worm containment systems in terms of security and performance.

Hardware testbeds such as DETER (DETER, 2011) have been used for evaluating containment strategies (Li et al., 2007). Even though they allow for more accurate and realistic evaluation, unfortunately hardware testbeds require a large amount of hardware resources. For example, (Li et al., 2007) requires ninety-three physical machines to emulate ten-thousand hosts. Furthermore, in order to quickly derive an effective defense system for extremely fast spreading worms, a worm testbed must be easily reconfigured with propagation models, defense strategies and network topologies. Therefore a software modular-based testbed serves as a complementary approach, which is able to simulate arbitrarily large networks without using large amounts of physical hardware.

In this paper, we present a testbed for evaluating worm containment systems as shown in Figure 1. The testbed allows for defining large networks while the functionality of individual hosts is managed by network agents. The software agents allow creation of worms and background traffic as well as worm defense mechanisms. These agents can be run on individual network hosts in the testbed. The behavior of each individual hosts is preserved by our simulation output. Our testbed can evaluate both security in terms of the number of infected hosts and performance impact in terms of delay and packet loss, which is different with previous work (Moore et al., 2003; Li et al., 2007).

Worm containment systems employ varying worm defense mechanisms. For example, content filtering mechanism performs deep packet inspection by comparing packet content with worm signatures. Blacklist mechanism compares source IP addresses to a blacklist of known infected hosts. The mechanism of worm containment systems can be statically or dynamically configured. If statically configured, all network hosts are hard coded to either perform blacklisting or content filtering. Static configurations can be used to demonstrate simple trade-offs between worm containment and network performance. If dynamically configured, network hosts can switch from one defense mechanism to another based on current network conditions and security threats. Dynamic configurations can help avoid some of the negative performance trade-offs associated with high security and aggressive worm containment. Our testbed allows users to explore varying defense mechanisms and operation strategies.

Figure 1 Testbed for simulating and evaluating worm containment



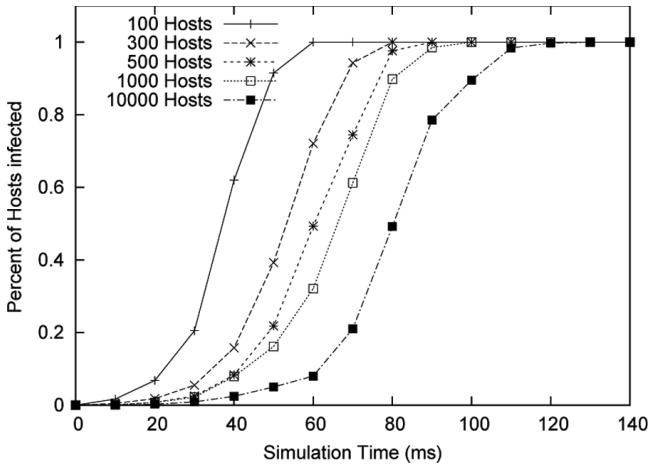
The design of our testbed is modular in that, new functionality can be easily implemented in our software agents. Different worm models (Zou et al., 2002; Chen et al., 2003; Chen and Ji, 2007) can be inserted into our testbed to propagate worms in various ways. Different containment strategies (Cai et al., 2005; Costa et al., 2005) can also be applied in our containment agent. We can also simulate differences in processing for various hardware by configuring our processing cost parameters.

We evaluate diferent containment system configurations and present results related to the performance of the containment systems in terms of mitigating worm threats as well as network performance. These observations lead us to further develop the testbed to include dynamic system response in order to improve security and performance. The main contributions of this work are:

- an efficient testbed for evaluating worm containment system security and performance in large networks with little physical hardware (i.e., 10,000 hosts per physical machine as in Figure 2)
- a modular design that allows insertion of different worm propagation models and containment techniques
- thorough evaluation of the security and performance of containment systems
- utilisation of the testbed to improve containment system design to include dynamic containment capabilities.

The remainder of this paper is organized as follows. Section 2 reviews the background behind worm modeling and containment as well as work that is most closely related to ours. Section 3 gives an overview of our testbed and Section 4 provides details about the containment agent. Section 5 presents various experiments that help to demonstrate the capability of our testbed. The paper is concluded in Section 6.

Figure 2 Infection for networks with different numbers of hosts



2 Background and related work

Modeling worm behavior (Riley et al., 2004; Zou et al., 2002; Zou et al., 2003; Chen et al., 2003) has been an important area of research for some time. Mathematical models and simulation give the ability to reproduce and predict worm behavior. For example, (Zou et al., 2007) shows that some worm models may overestimate the spreading speed of worms. More advanced worm propagation techniques (Chen and Ji, 2007) have also arisen out of worm modeling. Newer worm propagation such as permutation-scanning worms has been modeled in (Manna et al., 2008). Recently (Chen et al., 2009) has represented worm scanning using a closed-form expression. This understanding allows for better worm simulation. In our testbed we can insert any worm propagation model into the worm agent, making it possible to evaluate many types of worms and also making our testbed scalable for future worm propagation advancements.

With many worm models available, many researchers have made contributions to the study of worm defense in attempts to mitigate worm threats. Worm defense and containment strategies (Cai et al., 2005; Costa et al., 2005) include signature-based and anomaly detection.

Signature-based worm detection provides an effective way to mitigate worm attacks in that it has a low false positive rate. However, signatures must be generated in order to check for worm threats. Various solutions (Hyang-ah Kim, 2004; Singh et al., 2004; Tang and Chen, 2005). An incremental trie structure based signature generation (Tang et al., 2011) has been used to help identify familial resemblance between worms and can help to map how a worm's signature evolves. A double-honeynet system can also be used to detect zero day worms (Mohammed et al., 2010).

Anomaly detection (Kruegel and Vigna, 2003; Wang and Stolfo, 2004) such as rate-based detection, checks for abnormal network behavior in order to detect worms. This defense strategy does not need signatures or a known list of infected hosts, however, it often has a high false positive rate.

It is important to understand the benefits of these worm containment mechanisms in terms of security but also network performance. Some work has been presented which evaluates containment strategies. An analysis of address blacklisting and content filtering containment strategies has been presented in (Moore et al., 2003). They compare address blacklisting and content filtering based only on reaction time of the system. In (Li et al., 2007) they evaluate containment strategies using the DETER testbed (DETER, 2011). This requires about one hundred physical nodes to emulate a network size of ten thousand. Our testbed requires one physical machine per about 10,000 hosts. Wei et al. (2009) have developed tools to simulate worm spreads in the DETER testbed.

3 Testbed

The testbed is a customization of Network Simulator-2 (version 2.34) (2011). The simulator allows the definition of a network with many hosts. Each host in the testbed runs one of several software agents available in NS2. These agents simulate the packet level functionality of the hosts. Available agents are the traffic, worm and defense agents as shown in Figure 1. The defense or containment agent is a custom agent added by us to allow each host to run a defense mechanism.

NS2 generates trace iles that record each packet sent and received in the simulation. We write proprietary trace evaluation tools in the Microsoft Visual Studio 2008 (Microsoft Visual Basic, 2008). Simulations were developed and run on an Intel Pentium 4 CPU 3.00 GHz. with 1.5 GB RAM. The operating system used is Fedora Core 11 with OS Kernel Linux 2.6.29.5-191.

3.1 NS-2 Simulator

The NS-2 simulator is responsible for basic network configurations, message handling and generating traces. Our networks can consist of hundreds to thousands of hosts.

The hosts communicate using one or more of several available software agents. The message parser is responsible for formatting and sending messages. For example, the worm agent utilizes the message handler to send probe packets and the containment agent utilizes it to send updates. The trace generator generates a packet level trace of all network activity.

3.2 Agents

The individual hosts in the network simulation can run any or all of the available agents in the testbed. The agents include a traffic agent for generating regular network traffic, a worm agent, as well as our newly added containment agent which implements one of several defense options on a host. The following is an overview of the agents available in the testbed.

3.2.1 Traffic agent

The speed of infection and performance of worm containment depend on traffic intensity and network congestion. The traffic agent allows hosts to generate traffic of multiple types

in order to maintain fidelity of normal network operations, as well as simulate worm propagation more realistically. The traffic agent can generate UDP traffic, TCP traffic and custom message traffic. Hosts running the traffic agent can send UDP and TCP traffic. Other parameters for traffic include traffic rate, packet size and random traffic.

NS2 uses simple message passing for packet transmission, wherein the sending and receiving of packets is simulated by a time line using parameterized values for network delays. The background traffic is therefore devoid of a payload and is generated at random intervals to satisfy a given rate and packet size.

3.2.2 Worm agent

The worm agent enables a host to receive a worm infection. It also gives the host worm probing functionality in the case that it does become infected. The worm agent uses a packet probing method for spreading the worm. Various parameters control how the worm can propagate:

- (r_p) Probing rate (i.e., the rate at which attacker sends probe packet (probes/s))
- (s_p) Probing packet size (bytes)
- (a_r) Address range for attack (i.e., he scanning space, can be obtained from 2^{32} IP addresses)
- (n_s) Network Size (i.e., the number of total host within the targeted network)
- (p_a) Probability for successful attack (i.e., the probability of generating a probing address within a certain address range), which depends on (n_s) and If a host becomes infected the rate of valid traffic generated by that host (r_v) in bytes/s can be estimated by equation (1). This is essentially the probes that are generated within the valid address range of the network. Other worm propagation models can be used in our testbed, however for the sake of simple demonstration of the testbed features, here we use a random model.

If a host becomes infected the rate of valid trafficc generated by that host (r_v) in bytes/s can be estimated by equation (1). This is essentially the probes that are generated within the valid address range of the network. Other worm propagation models can be used in our testbed, however for the sake of simple demonstration of the testbed features, here we use a random model.

$$r_v = r_p \cdot s_p \cdot \frac{n_s}{a_r} \tag{1}$$

3.2.3 Containment agent

The containment agent is the largest contribution of this work in terms of testbed functionality. This agent allows hosts to utilize defense mechanisms with multiple modes of operation. Hosts can also switch between modes given certain network events. The network can be configured such that it runs in any operating configuration, or, combination of hosts running in different modes. The detailed information of the containment agent is discussed in Section 4.

3.3 Security and performance statistics

Output metrics from the testbed simulation can broadly be categorized into categories: network performance and containment performance.

Delay. This metric is good for analyzing the network performance of worm containment systems. End-to-end delay has many contributions which may include the processing delay overhead from the content filters and blacklist nodes. Another contributor to delay is queuing delay caused by worm traffic.

Packet Loss. Packet loss can be due to many factors such as increased traic or excessive delays. Tracking packet loss can help evaluate network performance.

Host Infection Percentage. The host infection percentage is the percentage of hosts infected at the point when a worm is no longer propagating to new hosts. Minimizing this percentage should be an important goal of any worm containment system.

Detection Time. The time it takes to detect the worm is detection time (t_d). This is calculated by Equation 2. It is the time span from when the first host is infected by the worm (t_i) to the time an infected host is blacklisted (t_b).

$$t_d = t_b - t_i \tag{2}$$

4 Containment agent

This section describes the containment agent, its operating modes and configurations, performance considerations and details of its implementations.

4.1 Modes of operation

Hosts can run the containment agent in one of two main modes: content filter mode and blacklist mode. If a host utilizes content filter mode it performs deep packet inspection on all traffic. This means that the host scans every byte of a packet to see if its payload matches a signature for a worm. If a worm signature is identified, the agent adds the source address of the identified packet to a 'blacklist'. The 'blacklist' holds the addresses of hosts that have been identified as being infected by a worm. In blacklist mode, a host simply checks the blacklist for the source address of each packet it receives.

Content filter mode can be run in one of two sub-modes: active and inactive. If active, a host performs content filtering (deep packet inspection). If inactive, a host performs as if it is in blacklist mode. To identify the presence of any hosts which are infected but currently not blacklisted, the inactive nodes should switch to content filtering (active). This switching from inactive to active should be as soon as possible. The blacklist broadcast coming from active hosts serves as a signal for inactive hosts to become active. Because this broadcast may be unreliable due to network congestion, we assume a control plane notification between routers which receives preferential treatment, similar to that used to exchange routing table information. Here we

also assume the worm signature is available in the signature database ready to be used by the active filters.

4.1.1 Content filter mode

The *new_packet* function in Algorithm 1 breaks down the behavior of content filter mode into active and inactive modes.

Active. A host running in active mode utilizes the *cf_function* to scan the packet for worm signatures. Packets containing known worm signatures are considered malicious. If a malicious worm packet is found the *new_infection_found* function is used to add the source of said packet to the blacklist and broadcast the updated list.

Inactive. A host running in inactive mode awaits the first blacklist broadcast or blacklist update packet *(bl_update)*. If such a packet is received then the inactive node activates itself. After this point, the newly activated node is running in active mode unless deactivated.

4.1.2 Blacklist mode

A host running in blacklist mode utilizes the *bl_function* to check the blacklist to see if the source of a packet is a known infected host. This same function is also used to update a host's locally stored copy of the current blacklist. This is done in the case that a *bl_update* packet is received.

Algorithm 1 Algorithm for containment agent

```
   Input: Packet: p
   Result: Action: a, Packet_Flag: f
 1 new_packet( p)
 2 begin
 3 |  if blacklist_mode then
 4 |  |  bl_function(p.src_adrs);
 5 |  end
 6 |  if content_filter_mode then
 7 |  |  if me.active then
 8 |  |  |  cf_function(p);
 9 |  |  end
10 |  |  if me.inactive then
11 |  |  |  if p.bl_update then
12 |  |  |  |  me.activate();
13 |  |  |  end
14 |  |  end
15 |  end
16 end
17 bl_function( src_adrs)
18 begin
19 |  if blacklist.contains(src_adrs) then
20 |  |  f = malicious;
21 |  end
22 |  if !(blacklist.contains(src_adrs)) then
23 |  |  f = benign;
24 |  |  if p.bl_update then
25 |  |  |  a = me.update_local_bl(adrs_list);
26 |  |  end
27 |  end
28 end
29 cf_function( p)
30 begin
31 |  if p.contains(worm_signatures) then
32 |  |  f = malicious;
33 |  |  a = new_infection_found(p.src_adrs);
34 |  end
35 end
36 new_infection_found( src_adrs)
37 begin
38 |  blacklist.add(src_adrs);
39 |  blacklist.broadcast();
40 end
```

4.2 Processing delay

The average containment agent processing delay at a host can be calculated as follows:

$$d_{proc} = d_{cf} \cdot p_{cf} + d_{bl} \times (1 - p_{cf}) \tag{3}$$

where d_{bl} is the worst case delay for a host to compare a packet address to the blacklist addresses, p_{cf} is the percentage of hosts in content filter mode and d_{cf} can be calculated as follows:

$$d_{cf} = r \cdot d_{bl} \tag{4}$$

where r is the ratio of processing delay for hosts in content filter mode to hosts in blacklist mode. r and d_{bi} are controllable input parameters for our testbed and they depend on both the packet size and also the technology used for content filtering. Assuming a deterministic multi pattern search based on Aho-Corasick (Aho and Corasick, 1975) or DFA (Vespa et al., 2009; Brodie et al., 2006), the processing overhead due to deep packet inspection is a constant delay per packet byte.

4.3 Operating configurations

The two main categories of operating conigurations are: static and dynamic.

4.3.1 Static configuration

In static configurations, all content filters are active before a worm is detected. When a worm is detected, the number of active filters remains static.

4.3.2 Dynamic coniguration

When dynamically allocating nodes as content filters, the configuration starts with a small percentage of content filters used to initially detect a worm. When a worm is detected, many other nodes are immediately activated as content filters allowing for a large percentage of filters to quickly blacklist all infected hosts. If an active content filter receives a worm probe, it broadcasts an updated blacklist with the source address of this probe. Accordingly, inactive content filters become active. When selecting a starting percentage of content filter hosts, it is important to be sure that the filters will be able to detect the worm fast enough to dynamically enable filter nodes in time to contain the worm. The detection time metric can be used to intelligently select a starting percentage of active filters for a network.

5 Results

The scope of our results starts with evaluating worm behavior without the presence of a containment mechanism and then compares static and dynamic containment mechanisms. The specifications for the range of parameters used are shown in Table 1. The default value is used unless otherwise specified. The value range is the total range that a value may take in all experiments.

Table 1 Experiment parameters

Parameter	*Default*	*Range*
Blacklist delay (ms)	5	5
Content filter delay (ms)	100	25–100
Active/inactive filter (%)	0	2–80
Scan rate (probes/sec)	7000	7000–13000
Worm probability (%)	20	20
Packet size (bytes)	1000	1000
Network size	500	100–10000
Link bandwidth (Mbps)	10	10
Queue length	100	100
Traffic rate (Mbps)	7	7–13
Traffic type	UDP	UDP
Traffic source	Random	Random

5.1 Network size

Figure 2 shows the infection of a worm for networks of sizes 100 to 10000 hosts. Because no containment mechanism is in place here, eventually all network hosts become infected. Due to the nature of worm probing behavior, the trend of percentage of hosts infected v.s. time is similar for different size of networks. At the beginning, the infected rate is relatively small because of fewer infected hosts to spread worms. With the number of infected hosts increasing, the infected rate is larger and larger. However, the infected rate will drop eventually to zero due to the smaller number of uninfected hosts. The peak value of infected rate is achieved when the number of infected hosts is the half of network size. This claim is justified as following.

With a_r as scanning space, n_s as network size, n_i as number of infected hosts and homogenous scanning rate s, the probability that an uninfected vulnerable host is hit by next time step is:

$$p = \frac{n_s - n_i}{a_r} \cdot n_i \cdot s \tag{5}$$

Then the time to recruit a new infected host, T, follows the geometric distribution, which leads to

$$E[T] = \frac{1}{p} \tag{6}$$

Therefore, the average infected rate is p.

To obtain the optimal infected rate, need finnd n_i that satisfies:

$$\frac{\partial_p}{\partial_{ni}} = \frac{n_s - 2 \cdot n_i}{a_r} \cdot s = 0 \tag{7}$$

It is clear that the optimal average infected rate is achieved when the number of infected hosts (n_i) equals to half of the network size (n_s).

5.2 Worm probing rate

Figure 3(a) shows the infection of the worm for 7k–13k probes/s. The overall infection takes slightly longer for a slower probing rate. Because no containment is used here, all hosts are eventually infected. Figure 3(b) shows the average network delay for 7k – 13k probes/s. When the worm reaches the network at about 500 ms, network delay immediately begins to increase. It increases at a faster rate for the faster propagating worms. Eventually, the delay settles because no new infections occur and the network traffic generated by infected hosts is constant. When network delay reaches a threshold, packets begin to drop as observed in Figure 3(c). The faster the worm propagates, the sooner the network sees packet loss.

5.3 Static containment

The first type of containment strategy we evaluate is static allocation of content filters and blacklisting nodes.

Figure 4(a) shows the percentage of hosts that were infected by the worm over time using anywhere from 2 percent to 40 percent of network hosts as active content filters. The more content filters, the slower the worm propagation and, more importantly, the less hosts infected before the worm is contained. For example, using 2 percent of hosts in content filter mode allowed over 97 percent of hosts to become infected with the worm. Using 40 percent of hosts as content filters allowed only about 11 percent of hosts to become infected.

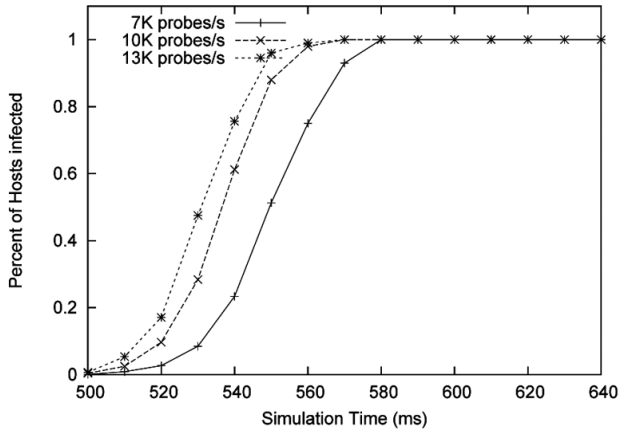
This section evaluates the performance impact of the static containment configuration. In Figure 4(b), before the worm hits the network (500 ms), the delay is fairly constant. This Figure shows the average delay for the network before during and after the worm. Although we see in Figure 4(a) that utilizing a larger percentage of content filters provides better defense against a worm, it can cause a negative trade-of to performance. The average delay on the network before the worm, is greater as the number of content filters increases. Figure 4(c) shows that as the percentage of content filters increases, the amount of packets dropped decreases. This is due to a smaller amount of worm traffic being introduced in the network.

The extra queuing delay created by new worm traffic has varying impact on overall delay. The impact depends on not only the percentage of content filters but also the relative processing cost of content filters (r). For example, in Figure 5(a), the increase in delay due to queuing delay is not enough to cause the overall delay to be greater when utilizing a smaller percentage of content filters. The impact is substantially increased when the ratio (r) is dropped from 20:1 to 10:1 as shown in Figure 5(b). Eventually, as seen in Figure 5(c) at a ratio of 5:1, the impact of queuing delay with less content filters, is greater than those of using more content filters.

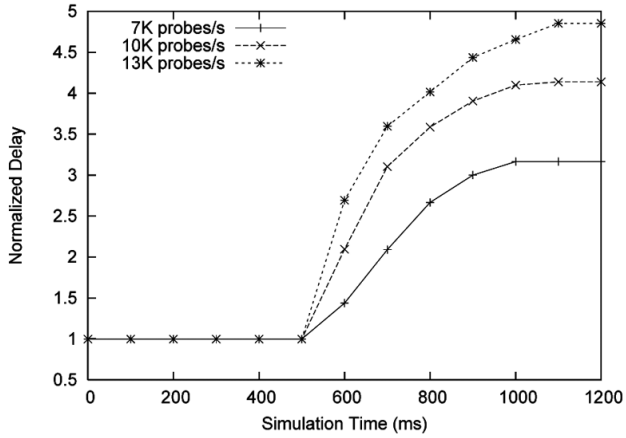
5.4 Dynamic containment

The second type of containment strategy we evaluate is dynamic allocation of content filters. During normal network

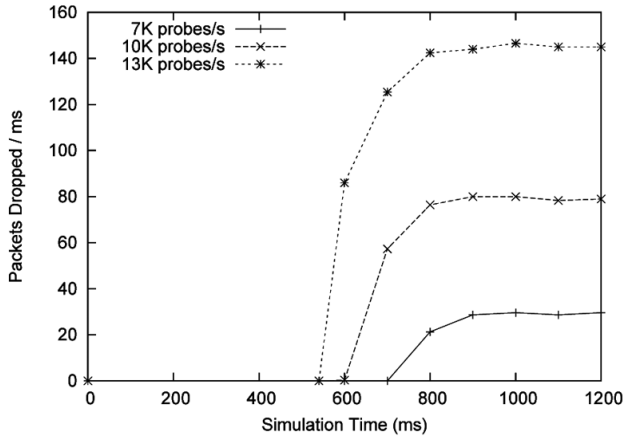
Figure 3 Infected hosts and network delay for different worm probing rates (a) Infected hosts, (b) Delay (Normalized to (processing + transmission) delay) and (c) Packet loss



(a) Infected hosts

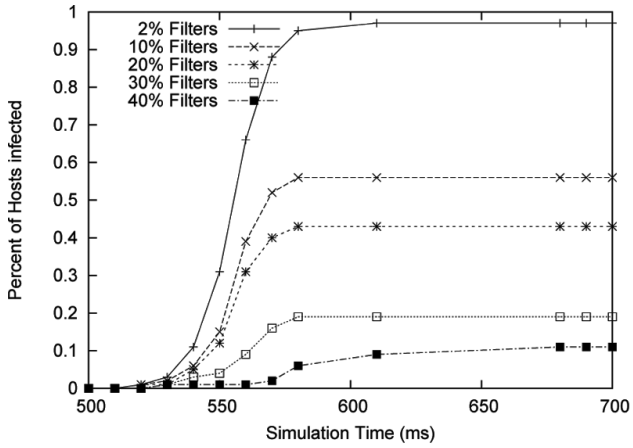


(b) Delay (Normalized to (processing + transmission) delay)

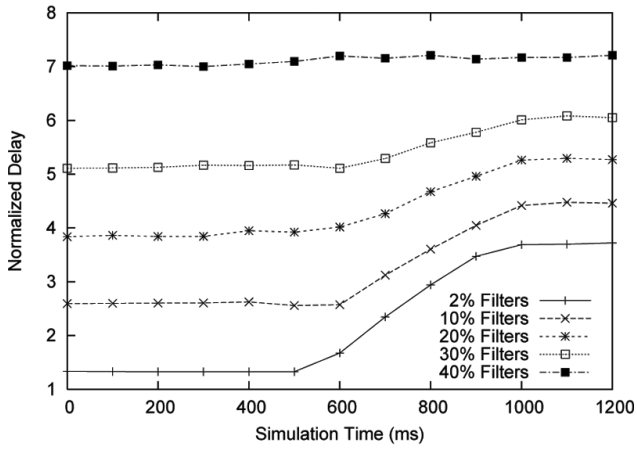


(c) Packet loss

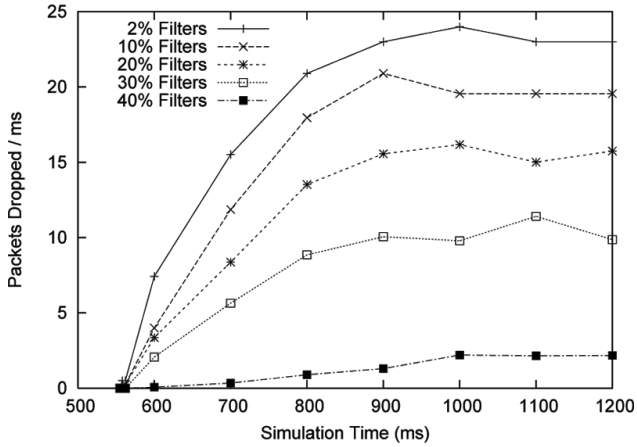
Figure 4 Infected hosts, network delay and packet loss for static containment with varying percentage of content filtering and $r = 20$ (a) Infected hosts, (b) Delay (Normalized to blacklisting delay) and (c) Packet loss



(a) Infected hosts



(b) Delay (Normalized to blacklisting delay)

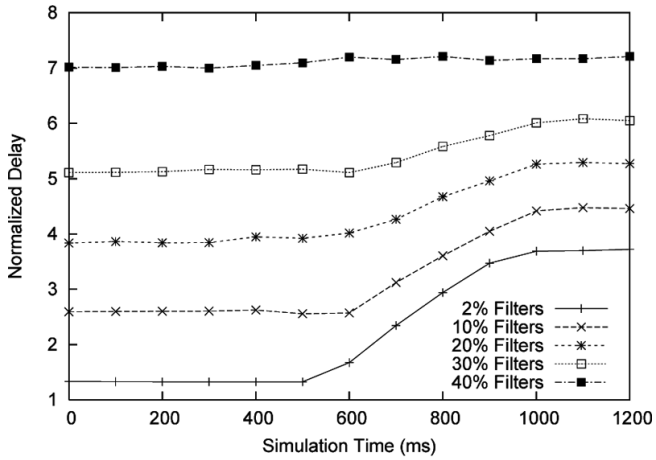


(c) Packet loss

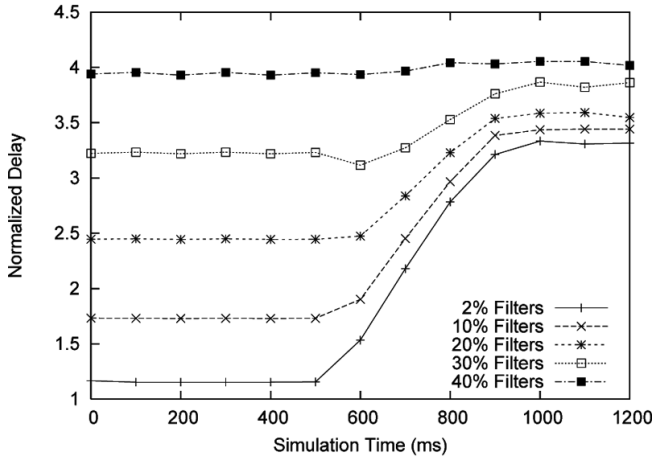
operation, it may be beneficial to network performance to use less active filters and more inactive filters that can be activated when a worm is detected. It is still important though to have enough active filters to detect a worm fast enough for the containment system to react to the threat.

Table 2 shows the detection time for 2 percent to 40 percent content filters. There is only about a 27 ms difference in detection time between the average detection time for 10 percent and 40 percent filters. Our tests will therefore use 10 percent as a starting percentage of active content filters.

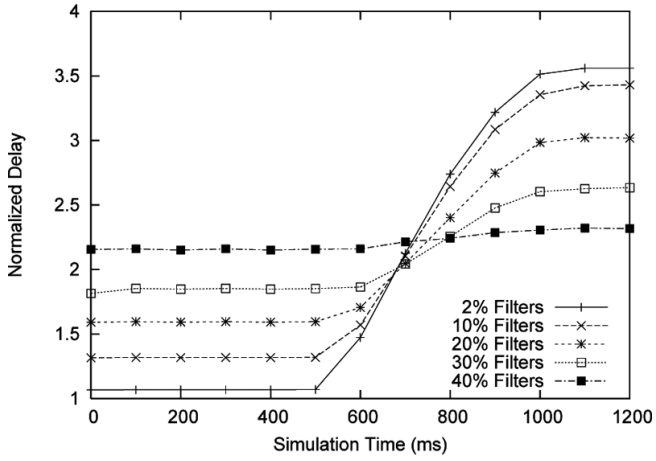
Figure 5 Network delay for static containment with varying cost(r) of content filtering (Normalized to blacklisting delay) (a) Delay ($r = 20$), (b) Delay ($r = 10$) and (c) Delay ($r = 5$)



(a) Delay (r = 20)



(b) Delay (r = 10)



(c) Delay (r = 5)

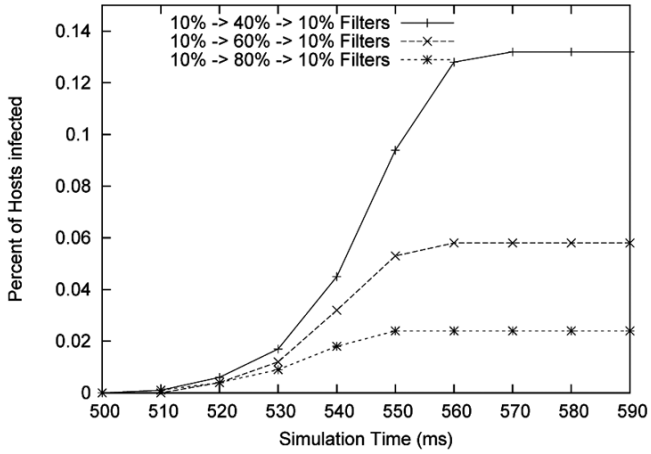
5.5 Static and dynamic comparison

Figure 6(a) shows the infected hosts percentage when using 10 percent active filters and sequentially activating an additional 30 percent, 50 percent and 70 percent inactive filters when a worm is detected. This makes a total of 40, 60 and 80 percent of hosts in content filter mode when a worm is detected. Results are promising in that, for 40 percent content filters in Figure 6(a), the percentage of infected hosts only rises to about 13 percent. This is only slightly above the 11 percent seen in Figure 4(a) or 40 percent statistically allocated content filters. This small increase in infection percentage is due to the slightly longer detection time for 10 percent versus 40 percent active hosts.

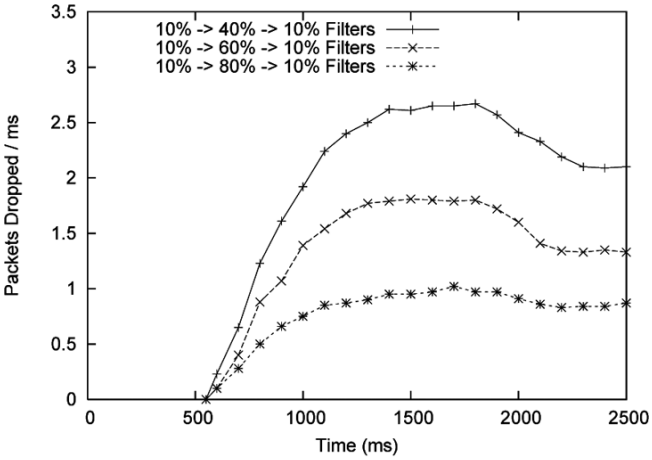
Table 2 Minimal, mean an maximal detection time for varying content filter percentage

Fraction of content filter	*Minimal detection time*	*Mean detection time*	*Maximal detection time*
2%	82.38	87.36	91.57
10%	73.82	77.77	82.85
20%	67.14	72.18	76.40
30%	56.57	61.28	65.89
40%	47.15	50.45	55.41

Figure 6 Infected hosts and packet loss for dynamic containment with varying percentage of content filtering (a) Infected hosts ($r = 20$) and (b) Packet loss ($r = 20$)



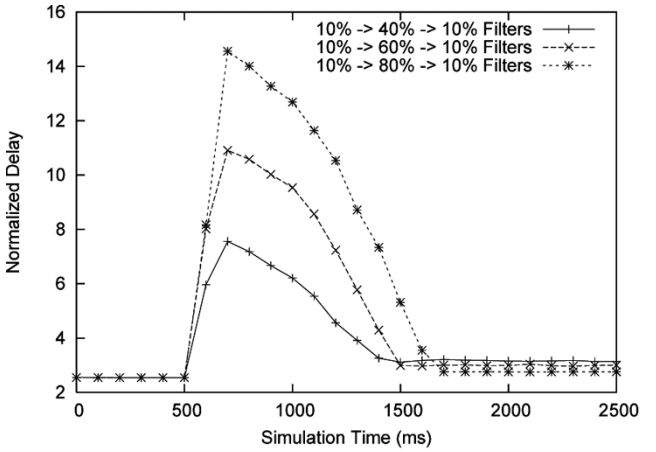
(a) Infected hosts (r = 20)



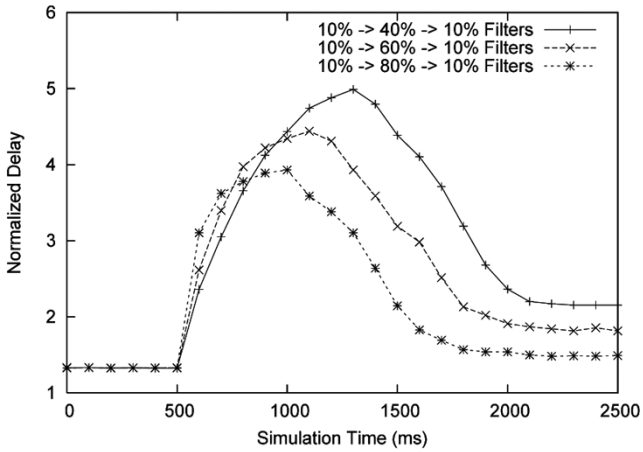
(b) Packet loss (r = 20)

The benefit of dynamic configuration is that the network delay during normal operation only suffers from the processing delay caused by the active hosts, not the inactive hosts. As shown in Figure 7(a), when 80 percent of content filters are active during a worm threat network delay jumps fairly high. However, after the worm is quickly contained, the filters are deactivated back to 10 percent active filters and the delay returns to a low level. In fact, for 80 percent filters, the delay returns to the lowest of all configurations. This is because using 80 percent filters results in the lowest worm infection percentage. This same fact makes the packet loss for 80 percent filters lower than that of 60 and 40 percent as seen in Figure 6(b). This is because the increase in traffic due to infected hosts is less for 80 percent filters and therefore there is less worm traffic and less packet loss. Also as seen with static configurations, eventually, the ratio (r) may reach a point where the queuing delay caused by new worm traffic overtakes the effect of the processing delay caused by the processing overhead of content filters. Figures 7(a) and 7(b) demonstrate this point. Figure 7(a) shows the network delay for a dynamic configuration where $r = 20$. Figure 7(b) shows the same for $r = 5$. In Figure 7(b), when the inactive filters are activated, the delay for 80 percent filters begins to rise the fastest, however, the delay for 40 percent filters overtakes it soon after. This is due to the queuing delay generated by worm traffic as the infection rate is higher for 40 percent filters than for 80 percent filters.

Figure 7 Network delay for dynamic containment with varying cost(r) of content filtering (Normalized to blacklisting delay) (a) Delay ($r = 20$) and (b) Delay ($r = 5$)



(a) Delay (r = 20)



(b) Delay (r = 5)

Table 3 shows network behavior for both static and dynamic approaches: before, during and after a worm attack. To compare the approaches before the presence of a worm, we examine network delay (normalized to d_{bl}) in columns 3 and 4. Here we see that all percentages of content filters, the delay for static is greater than the delay for dynamic. This is because dynamic does not allocate all available filters as active filters until a worm is detected. Therefore, the processing delay is much less in a dynamic configuration during normal network operation.

Examining columns 5 and 6 of Table 3 shows the delay when a worm is present. Here we see that the delay for dynamic is comparable to that at static with just a slight increase from static. At this point, the inactive filters in the dynamic configuration have been activated increasing the processing delay up to the level of static. The very small number of hosts will be infected that would not be infected in a static configuration. However this number is very small and yields a very small difference in delay. To further support this we can examine the packet loss rate during a worm presence in columns 6 and 7. Again, the packet loss is slightly higher for dynamic but only by a few packets. This is due again to the detection time and activation time for a dynamic configuration.

After a worm is contained and no new hosts are being infected by the worm, the difference in delay seen in columns 8 and 9 becomes substantially favored for dynamic. This is mostly due to the deactivation of active hosts which significantly reduces the processing delay for dynamic.

Table 3 Comparison of ST. (static) and DY. (dynamic) containment before, during and after a worm threat. (Delay is normalized to blacklisting delay)

	Before worm		*During worm*				*After worm*					
	Delay		*Delay*		*Packet* loss/ms		*Delay*		*Packet* loss/ms		*% Infected hosts*	
% *Filters*	*ST.*	*DY.*	*ST.*	*DY.*	*ST.*	*DY.*	*ST.*	*DY.*	*ST.*	*DY.*	*ST.*	*DY.*
40%	7.02	2.55	7.21	7.56	2.27	2.67	7.21	3.14	2.21	2.09	0.11	0.13
60%	10.39	2.55	10.79	10.91	1.78	1.82	10.79	3	1.79	1.36	0.05	0.06
80%	13.97	2.55	14.23	14.56	0.85	0.98	14.24	2.76	0.85	0.84	0.02	0.02

The main metric for judging the security of a containment system is the percentage of hosts that get infected by the worm. Columns 12 and 13 show this percentage. It is comparable for static and dynamic. The longer detection time and activation time for dynamic yields an only slightly higher infection percentage for dynamic. However, dynamic configurations show comparable performance during worm mitigation and much better performance both before a worm threat and after a worm is contained.

6 Conclusion

This paper has introduced a testbed which utilizes software agents to allow large scale simulation with individual host functionality. Containment systems can be evaluated using this testbed in terms of security and performance tradeoffs. By presenting a variety of results, we have shown that our testbed can be used to successfully evaluate worm containment systems. This evaluation yields many insightful understanding of containment strategies currently utilized in real networks. Furthermore, this testbed can be used as an aid in the design of new containment strategies such as the dynamic containment strategies.

In future work we plan to use the modularity of our testbed to apply more advanced worm propagation models. We will also implement other containment strategies such as anomaly detection algorithms. We also plan to implement these containment strategies in the DETER testbed in order to compare our results with those in physical hardware.

References

Aho, A. and Corasick, M. (1975) 'Efficient string matching: an aid to bibliographic search,' *Communications of the ACM,* Vol. 18, pp.333–340.

Brodie, B.C., Taylor, D.E. and Cytron, R.K. (2006) 'A scalable architecture for high-throughput regular-expression pattern matching,' *SIGARCH Comput. Archit. News,* Vol. 34, No. 2, pp.191–202.

Cai, M., Hwang, K., Kwok, Y.-K., Song, S. and Chen, Y. (2005) 'Collaborative internet worm containment,' *IEEE Security and Privacy*, Vol. 3, No. 3, pp.25–33.

Chen, Z. and Ji, C. (2007) 'Optimal worm scanning method using vulnerable host distributions,' *Int. J. Secur. Netw.*, Vol. 2, Nos. 1/2, pp.71–80.

Chen, Z., Chen, C. and Li, Y. (2009) 'Deriving a closed form expression for worm scanning strategies,' *Int. J. Secur. Netw.*, Vol. 4, No. 3, pp.135–144.

Chen, Z., Gao, L. and Kwiat, K. (2003) 'Modeling the spread of active worms,' in *INFOCOM*, pp.1890–1900.

Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L. and Barham, P. (2005) 'Vigilante: end-to-end containment of internet worms,' in *In Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP)*, pp.133–147.

DETER (2011) 'Network security testbed,' http://www.isi.deterlab.net/

Hyang-ah Kim, B.K. (2004) 'Autograph: toward automated, distributed worm signature detection,' in *In Proceedings of the 13th Usenix Security Symposium*, pp.271–286.

Kruegel, C. and Vigna, G. (2003) 'Anomaly detection of web-based attacks,' in *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS '03).* Washington, DC, October, pp.251–261.

Li, L., Liu, P., Jhi, Y.C. and Kesidis, G. (2007) 'Evaluation of collaborative worm containment on the deter testbed,' in *DETER: Proceedings of the DETER Community Workshop on Cyber Security Experimentation*, Berkeley, pp.5–12.

Manna, P.K., Chen, S. and Ranka, S. (2008) 'Exact modeling of propagation for permutation-scanning worms,' in *INFOCOM*, pp.1696–1704.

Microsoft Visual Basic (2008) Express Edition, http://www.microsoft.com/express/

Mohammed, M., Chan, H., Ventura, N., Hashim, M. and Bashier, E. (2010) 'Fast and accurate detection for polymorphic worms,' in *International Conference for Internet Technology and Secured Transactions (ICITST),* Nov, pp.1–6.

Moore, D., Shannon, C., Voelker, G. and Savage, S. (2003) 'Internet quarantine: requirements for containing self-propagating code,' in *IEEE Conference on Computer Communications* (*INFOCOM*), pp.1901–1910.

Riley, G.F., Sharif, M.I. and Lee, W. (2004) 'Simulating internet worms,' in *MASCOTS '04.* Washington, DC, pp.268–274.

Singh, S., Estan, C., Varghese, G. and Savage, S. (2004) 'Automated worm fingerprinting,' in *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Berkeley, CA, pp.45–60.

Tang, Y. and Chen, S. (2005) 'Defending against internet worms: a signature-based approach,' in *In Proceedings of IEEE INFOCOM05*, pp.1384–1394.

Tang, Y., Xiao, B. and Lu, X. (2011) 'Signature tree generation for polymorphic worms,' *IEEE Transactions on Computers,* Vol. 60, No. 4, pp.565–579.

The Network Simulator ns-2 (v2.34) (2011) http://www.isi.edu/nsnam/ns/

Vespa, L., Mathew, M. and Weng, N. (2009) 'P3fsm: portable predictive pattern matching finite state machine,' in *20th IEEE International Conference on Application-Specific Systems, Architectures and Processors,* Boston, MA, USA, pp.219–222.

Wang, K. and Stolfo, S.J. (2004) 'Anomalous payload-based network intrusion detection,' pp.203–222.

Weaver, N., Staniford, S. and Paxson, V. (2004) 'Very fast containment of scanning worms,' in *SSYM'04 Proceedings of the 13th Conference on USENIX Security Symposium*, Berkeley, CA, pp.29–44.

Wei, S., Ko, C., Mirkovic, J. and Hussain, A. (2009) 'Tools for worm experimentation on the deter testbed,' in *Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops*, April, pp.1–10.

Zou, C.C., Duffield, D., Towsley, N. and Gong, W. (2006) 'Adaptive defense against various network attacks,' *IEEE Journal on Selected Areas in Communications*, Vol. 24, No. 10, pp.1877–1888.

Zou, C.C., Gao, L., Gong, W. and Towsley, D. (2003) 'Monitoring and early warning for internet worms,' in *CCS'03: Proceedings of the 10th ACM Conference on Computer and Communications security*, New York, NY, pp.190–199.

Zou, C.C., Gong, W. and Towsley, D. (2002) 'Code red worm propagation modeling and analysis,' in *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications security*, New York, NY, pp.138–147.

Zou, C.C., Towsley, D. and Gong, W. (2007) 'Modeling and simulation study of the propagation and defense of internet e-mail worms,' *IEEE Trans. Dependable Secur. Comput.,* Vol. 4, No. 2, pp.105–118.