

# Behavior-based Worm Detectors Compared<sup>\*</sup>

Shad Stafford and Jun Li  
{staffors, lijun}@cs.uoregon.edu

University of Oregon

**Abstract.** Many worm detectors have been proposed and are being deployed, but the literature does not clearly indicate which one is the best. New worms such as IKEE.B (also known as the iPhone worm) continue to present new challenges to worm detection, further raising the question of how effective our worm defenses are. In this paper, we identify six behavior-based worm detection algorithms as being potentially capable of detecting worms such as IKEE.B, and then measure their performance across a variety of environments and worm scanning behaviors, using common parameters and metrics. We show that the underlying network trace used to evaluate worm detectors significantly impacts their measured performance. An environment containing substantial gaming and file sharing traffic can cause the detectors to perform poorly. No single detector stands out as suitable for all situations. For instance, connection failure monitoring is the most effective algorithm in many environments, but it fails badly at detecting topologically aware worms.

**Key words:** Internet worm, worm detector, behavior-based detection

## 1 Introduction

Network worms have long posed a threat to the functioning of the Internet. As early as the outbreak of the Morris worm in 1988 [1], they have been capable of disrupting traffic over large swathes of the Internet. Significant outbreaks such as the CodeRed [2] and Slammer [3] worms in 2001 and 2003 brought the threat to national prominence and spurred the development of a wide range of mechanisms to detect the presence of worms and to harden operating systems against common attacks. The emergence of the Conficker worm [4] in late 2008 showed that those efforts had not eradicated worms completely. As the Internet continues to play a more important role in everyday life for hundreds of millions of people and as the very nature of the devices on the Internet is changing (e.g., consumer-level mobile devices begin to make up a substantial portion of connected devices), the Internet requires more protection than ever. The question remains—can we protect our networks from worms?

---

<sup>\*</sup> This material is based upon work supported by the United States National Science Foundation under Grant No. CNS-0644434. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The relevance of this question was further highlighted in late 2009 when a network worm was found propagating exclusively on iPhones. The IKEE.B worm [5] takes advantage of a default root password set in some *jail-broken* iPhones to propagate. It brings to light some shortcomings in current worm detection and prevention work. Specifically, because it propagates via an encrypted channel, it bypasses worm detectors that rely on examining the content of network traffic; and because it exploits a configuration error rather than a buffer overflow to gain control of the target machine, it is undeterred by defensive techniques such as address space randomization.

In this paper, we examine our ability to detect the presence of a worm in a protected network. Existing detection schemes can be broadly classified into host-based systems that monitor system call information or other host-level behavior for illegal operations, and network-based systems that monitor network traffic. Network-based systems can be further broadly divided into content-based systems that monitor the bytes transmitted across the network and behavior-based systems that monitor the patterns of network traffic.

Unfortunately, it is unclear how these detection systems perform relative to each other as there is very little work that directly compares them. Algorithms are typically published with evaluations against a single network trace, which is different for different algorithms and generally not available publicly. For example, the MRW detector was evaluated against an unidentified week-long trace of a university department with 1,133 identified hosts [6], whereas the TRW algorithm was evaluated against two traces collected at the peering link of an ISP containing 404 and 451 identified hosts [7]. Furthermore, worm detectors are evaluated with different performance metrics, and tested worms do not always follow the same set of parameters (such as scanning strategy and speed). For example, the detection latency of the DSC detector is measured in the percent of the network infected at detection time [8] while the MRW detector does not provide detection latency results at all.

We seek to remedy this situation by performing a comprehensive analysis of several worm detectors that are easily deployable and in principle capable of detecting IKEE.B. We select six of the most prominent behavior-based worm detection techniques and measure their detection performance against a variety of worm propagation strategies over a common set of network traces. We evaluate each detector using key performance metrics related to accuracy and latency. The questions we seek to answer include: Is any one detection algorithm clearly superior to the others, including cases when fast worms are the only concern or a special network environment is protected (e.g., residential networks that see game or peer-to-peer network usage)? If a worm adopts smart scanning strategies such as slowing down or intelligently choosing victims, can it evade these detectors? And, does the network trace selected for evaluation significantly impact the detection performance?

Highlights of our findings include: (1) We find that the network trace impacts the sensitivity of the detectors. They are less sensitive in environments with more Internet gaming and file sharing activity, which appears more similar

to worm activity than other benign activities such as web browsing. (2) Our results show that there is no clear winner and every detector has its limitations. For example, connection-failure monitoring is the most consistently sensitive detection technique for random scanning and local-preference worms, but it fails drastically in the case of a topologically aware worm. (3) In all environments, a stealthy worm scanning at one scan per minute and employing some form of topologically aware scanning that avoids connection failures could evade all the detectors evaluated in all environments.

The rest of this paper is organized as follows: We first discuss how we selected detectors in Section 2, and then examine the selected detectors in some detail in Section 3. We discuss our selected metrics in Section 4, followed by the methodology by which we evaluate detectors in Section 5. We present the results of our evaluations in Section 6. Section 7 reviews related work, with our conclusions in Section 8.

## 2 Detector Selection

In this section we describe published worm detection algorithms and justify our choice of six specific detectors for this comparison study. We performed an extensive evaluation of proposed worm detectors, considering 36 different published works. We grouped them into the following categories based on their detection algorithm: host-based detectors, content-based detectors, and behavior-based detectors. Each category has its own strengths and weaknesses.

Detectors that we classified as *host-based* included, among others: COVERS [9], DACODA [10], TaintCheck [11], and Sweeper [12]. Several factors, however, lead us to exclude host-based detectors from this study. Host-based detectors require end-host deployment but a network operator may have no control over what software is installed on the end-hosts running in their network. Furthermore, users may circumvent host-based software installs as illustrated by IKEE.B, which targeted only those users who intentionally installed an unsupported operating system. Finally, it is unclear whether host-based systems are capable of detecting an attack like that used by IKEE.B. The systems listed above all rely on observing malicious memory manipulations such as buffer overflows, but IKEE.B did not perform any illegal memory operations; it merely exploited a configuration vulnerability.

Detectors that monitor the network instead of end-hosts seem much more promising because they do not require deployment on each host to be protected. We first look at detectors that examine the contents of network traffic, including AutoGraph [13], EarlyBird [14], PAYL [15], Anagram [16], and LESG [17]. Each of these detection mechanisms share a similar limitation that leads us to exclude them from our comparison: *they are unable to monitor encrypted traffic*. Encrypted traffic is a special case of making a worm polymorphic. Content-based systems designed to catch polymorphic worms (such as Polygraph [18]) depend on attack-specific, invariant sections of content which may not be present for an encrypted worm. Even when worms are transmitted using unencrypted con-

nections, advances in polymorphism research such as [19] have threatened the promise of these detectors. Also, it is prohibitively difficult to acquire a variety of network traces which contain full network content, making it infeasible to evaluate these detectors.

The remaining and largest class of detectors is behavior-based (or payload oblivious) detectors. These include TRW [7], RBS [20], PGD [21], and many others. These systems also monitor network traffic, but they examine the behavior of traffic from end hosts rather than the contents of their packets. This type of system is easily deployed, requiring as little as a single monitor at the network gateway. They are capable of detecting worms regardless of the scanning mechanism or propagation type (including propagation via encrypted channels), and many of them are capable of identifying the worm-infected hosts. However, we do exclude some behavior-based systems that a network operator could not easily deploy. For example, detectors using network telescopes (such as those by Wu et al. [22] and Zou et al. [23]) require a large dark address space and cannot be deployed by a network operator unless they control a large address space.

After our exhaustive evaluation of worm detectors, we are left with the following selections: TRW [7], RBS [20], TRWRBS [20], PGD [21], DSC [8], and MRW [6]. We discuss these detectors in greater detail in the next section.

### 3 The Selected Worm Detectors

Having selected detectors for our comparison work, we now describe them each in more detail in roughly chronological order of their publication. We present only a brief summary of each work, please refer to the original publications for more detail. Note we used existing acronyms for each work where available.

The TRW detector was published by Schechter et al. in 2004 [7]. TRW identifies a host as worm infected if connection attempts to new destinations result in many connection failures. TRW is based on the idea that a worm-infected host that is scanning the network randomly will have a higher connection failure rate than a host engaged in legitimate operations. Even with the IPv4 address space getting closer to complete allocation, the majority of addresses will not respond to a connection attempt on any given port. Randomly targeted connections (as in worm scanning) will likely fail.

The destination-source correlation detector (DSC) was published in 2004 by Gu et al. [8]. It detects a worm infection by correlating an incoming connection on a given port with subsequent outgoing infections on that port. If the outgoing connection rate exceeds a threshold established during training, the alarm is raised. A different threshold is maintained for each destination port.

The MRW detector was first published in 2006 [6]. It is based on the observation that whereas worm scanning results in connections to many destinations, during legitimate operations the growth curve of the number of distinct destinations over time is concave. And as the time window increases, destination growth slows. This can be leveraged by monitoring over multiple time windows

with different thresholds for each window. If the number of new destinations for a host within a given window exceeds the threshold, the alarm is raised.

The RBS detector was first published in 2007 [20] by Jung et al. . Similar to the MRW detector, RBS measures the rate of connections to new destinations. The work is based on the hypothesis that a worm-infected host contacts new destinations at a higher rate than a legitimate host does. RBS measures this rate by fitting the inter-arrival time of new destinations to an exponential distribution.

The TRWRBS detector was published alongside the RBS detector [20]. It combines the TRW and RBS detectors into a unified scheme, and observes both the connection failure rate and the first contact rate. It performs a sequential hypothesis testing on the combined likelihood ratio to detect worms.

The Protocol Graph detector (PGD) was introduced by Collins and Reiter in 2007 [21]. It is targeted at detecting slowly propagating hit-list or topologically aware worms. PGD works by building protocol-specific graphs where each node in the graph is a host, and each edge represents a connection between two hosts over a specific protocol. Collins and Reiter made the observation that during legitimate operations over short time periods, the number of hosts in the graphs is normally distributed and the number of nodes in the largest connected component of each graph is also normally distributed. During a worm infection, however, both numbers will go beyond their normal range, indicating the presence of the worm.

## 4 Performance Metrics

The goal of this study is to evaluate the selected detectors over a comprehensive parameter space to identify their strengths and weaknesses. We must first, however, determine which performance attributes we are most interested in capturing, and what metrics would be suitable for assessing them.

The focus of this study is on the ability of the detectors to discover the presence of a worm in the network. We thus want to measure their accuracy: does a detector alert us when a worm is present—but not do so when there is no worm? Furthermore, we want to measure its ability to detect a broad range of worm scanning algorithms. Moreover, accurate detection is not helpful if it happens too far after the fact. We must obtain some notion of the speed of the detectors—does it find a worm quickly or does it allow the worm free action for a long time before raising the alarm.

There are some attributes that we are not as interested in. At this time we are ignoring runtime costs such as processing or memory requirements. These are dependent on implementation and optimization details, and can vary widely for a given detection algorithm (for example, see the hardware implementation of TRW by Weaver et al. [24]). It is beyond the scope of this work to attempt to determine how efficiently each of these algorithms could be implemented. Similarly, we do not consider the complexity of installing or running the detector. This is not because installation complexity does not impact the potential adoption rate of a detector, but because it is somewhat orthogonal to the accu-

racy of the detector itself and could be addressed separately from the detection algorithm itself.

As shown in Table 1, we have identified four metrics as the most useful measures of the performance of a worm detector. We explain them below:

<b>F-</b>	Percentage of experiments where worm traffic is present but not detected in time period $\tau$
<b>F+ by host</b>	The number of false alarms raised during a time period $\tau$ , limited to at most one false alarm per host
<b>F+ by time</b>	Percentage of minutes during a time period $\tau$ where a false alarm is triggered for any host
<b>Detection Latency</b>	The number of outbound worm connections from an infected network prior to detecting the worm

**Table 1.** Metrics

Our false negative metric works as follows. For each experiment we introduce a worm to the background legitimate traffic. The detector is limited to a time period  $\tau$  (typically an hour) to detect the worm after it becomes active. If in that time span an alarm is not raised, the experiment is scored as a false negative for the detector. The *false negative rate* (F-) is the percentage of experiments scored as false negatives. (We report F- for each different scanning rate of the worm.)

The flip side of false negatives is false positives: reporting legitimate traffic as a worm infection. This is a critical metric for worm detectors, because a detector that repeatedly raises a false alarm (“cries wolf”) will quickly be ignored by network administrators. We measure false positives by running the detector against benign traffic with no injected worm activity. (Because we have inspected the traces for known worm activity, we consider every alarm raised by a worm detector a false alarm.) However, because worm detectors often repeat their worm infection tests—on every connection in some cases, the same set suspicious behavior may cause the alarm to be raised repeatedly, and these repetitive alarms should be coalesced into a single notification to the network administrators. But the exact mechanism and scope of alarm coalescing will be specific to the needs and resources of the network administrators at each site. As a result, we present two forms of false positive rate. We present the number of hosts identified as infected (coalescing alarms by network address) as the *false positive rate by host* (F+ by host). We also define *false positive rate by time* (F+ by time), which is the fraction of minutes of the trace where an alarm is raised on at least one host; note the alarm duration is only until the end of the current minute as we coalesce alarms into 1 minute bins. The combination of these two metrics give a better view of the overall false positive performance of the detector than either does individually.

The next major performance attribute to consider is the speed with which a worm is detected. The faster detection occurs, the less damage the worm can do. We measure *detection latency* as the number of outbound worm connections initiated by all infected hosts in the protected network prior to detection of any internal infection. (Scans that do not leave the network do not inflict damage on the Internet as a whole and are not included in this count.) Alternative approaches such as using clock time or infected host count are less accurate and less descriptive than our metric.

## 5 Experiment Design

We run the detectors against legitimate traffic to measure false positives, then against legitimate traffic plus known worm traffic to measure false negatives and detection latency. We developed a custom testing framework and implemented each detector in our framework based on the detector’s published specifications. Our framework can run against online, real-time traffic on the DETER testbed [25], as well as run in an offline (not real-time) mode. We use legitimate traffic from a variety of sources and generate known worm traffic by simulating a worm with our GLOWS [26] simulator. We vary the following parameters as we evaluate each worm detector: the environment it is run in (meaning the network configuration and legitimate traffic), the worm scanning method, and the worm scanning rate. We have also studied the effects of two additional parameters: the target port attacked by the worm and the activity profile of the first host infected by the worm, but omit those results from this paper due to space constraints.

### 5.1 Evaluation Environment and Background Traffic

Worm detectors must be evaluated in the context of a subnet to be protected and against the legitimate background traffic that occurs in that subnet. For our experiments, we define an *environment* as the network address space to be monitored, the IP addresses of the active hosts inside that address space, and the IP network traffic into and out of that address space during two time periods. We use the first time period for training and the second to run experiments against. To make the environments comparable to each other and to enable us to ensure that they do not contain worm traffic, we select a /22 subnet from the original recorded traces to use as the protected address space in our environment. Every environment is thus a /22 network with between 100 and 200 active hosts. We use four distinct environments in our evaluation.

The *enterprise* environment is built from a trace collected at LBNL [27] in January of 2005. Heavy scanners were removed from the trace before it was released. It has 139 active hosts and the training and experiment segments each contain roughly 25,000 connections.

The *campus* environment is built from a trace that was collected in 2001 at the border of Auckland University [28]. The trace was anonymized using a non-prefix preserving anonymization scheme, so we cannot entirely accurately

reconstruct the internal structure of their network. Instead, we randomly select 200 hosts and construct an environment using traffic to and from those hosts. Each segment of the trace in our campus environment contains approximately 25,000 connections.

The *wireless* and *department* environments are built from traces collected at the University of Massachusetts in 2006 [29]. The department environment is built from a trace capturing all traffic to and from the wired computers in the CS department. It has 92 active hosts and approximately 30,000 connections in each segment. The wireless environment comes from a trace capturing all wireless network traffic from the university. It has 313 active hosts and approximately 120,000 connections in each segment.

## 5.2 Worm Parameters

Several key parameters of a worm may impact the effectiveness of worm detectors. We look at three scanning strategies worms can employ: random scan, local-preference scan, and topologically aware (topo) scan, and evaluate them at a variety of scanning rates. Our GLOWS simulator takes an environment as input and simulates a worm as if it were attacking the network defined by that environment. The simulation starts with a single inbound worm connection that infects one host in the protected network. We run the simulator once for each permutation of worm parameters. The scanning mechanisms are defined as follows.

A random scanning worm simply chooses target addresses at random from the entire IPv4 address space. This typically results in many connection attempts to addresses with no host present or with a host that is not running the requested service, resulting in many connection failures. Permutation and sequential scanning worms should show very similar characteristics and are not evaluated separately here.

A local-preference worm scans local addresses (in the same prefix) more frequently than addresses in the full address space. This results in more scans that do not cross the network border (and are therefore not visible to a border-located detection mechanism). Existing local-preference scanning worms, such as Code-Red II [2], target the local /16 prefix approximately 50% of the time, the local /8 25% of the time, and the entire network the remaining time. As all our traces are about a /22 network, such a worm would largely resemble a random scanning worm. Instead, our local-preference worm scans the local /22 50% of the time, the local /8 25% of the time, and the entire network the remaining time.

The topologically aware (topo) worm finds target information on the host that it infects. This target information allows it to scan effectively because it already knows about other hosts that are running the service it targets. The number of new hosts (referred to as “neighbors”) the worm discovers is dependent on its neighbor detection algorithm. We use three implementations of the topo worm with differing neighbor counts. The topo100 worm starts with 100 neighbors, the topo1000 worm starts with 1000 neighbors, and the topoall worm

starts with an unlimited supply of neighbors. After scanning its known neighbors, the topo worm must either stop scanning or switch algorithms. In our implementation it reverts to random scanning after exhausting its neighbor list. Note that the neighbors discovered by the topo worm are randomly located, so could appear both inside and outside the protected network. Also, they will be running the target service but are not guaranteed to be vulnerable.

In addition to scanning mechanism the worm uses, the rate at which it initiates connections is important. The faster a worm scans, the more visible it is to worm detectors. We run experiments for a variety of worm scanning rates ranging from 10 connections per second down to one connection every 200 seconds.

### 5.3 Experiment Procedure

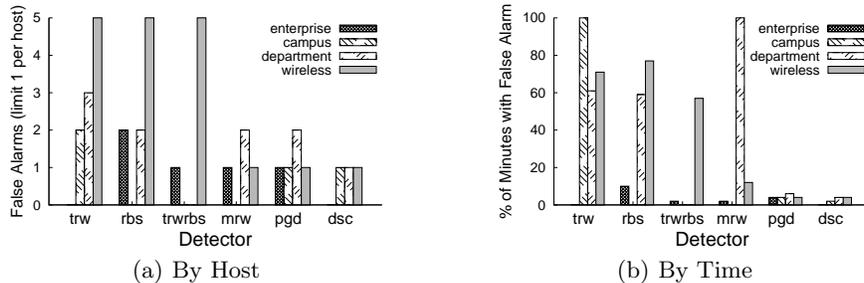
Measuring detector performance is a multi-step procedure. For each environment, every detector must (1) establish thresholds via training, (2) be evaluated against the legitimate traffic in the environment to measure false positives, (3) adjust their parameters to fix false positives at a specific level, and (4) be evaluated against legitimate traffic combined with worm traffic to measure false negatives and detection latency. Let us now discuss each of these steps in more detail.

These detectors are anomaly detectors, and they look for traffic that diverges from normal. To do this, they must first measure what normal is. The TRW, MRW, DSC, and PGD detectors are run against the training segment of the trace using the training method outlined in their publication to perform this operation. The RBS and TRWRBS detectors perform on-the-fly training as they are run against the experiment segment of the trace.

After the thresholds are established from the training segment of the trace, each detector is run against the experiment portion of the trace to measure false positives. We measure F+ using the thresholds obtained from training and the default detector parameters outlined in the original publication of each work, presenting those results in Section 6.1.

Note that each detector can be tuned to favor producing either more F+ or more F-. After reporting F+ using the default detector parameters as published, in order to provide a fair comparison of the false negative rate of the detectors, we modify each detector's parameters such that they all produce the same number of false positives in each environment. We chose to peg each detector at a rate of two false positive alarms during the experiment period. Two false positives is a high rate for the one-hour time period evaluated, but was chosen as an achievable value for all detectors requiring the minimum amount of parameter modifications.

After measuring F+ and adjusting the detectors to match their F+ levels, we then measure the performance of the detectors against worm traffic. For each detector in each environment, we run 16 experiments for every permutation of the worm parameters. A single experiment consists of running the detector for 10 minutes of the experiment trace to warm up the connection histories, then injecting the simulated worm traffic into the trace, and running until either an



**Fig. 1. False positives against legitimate traffic:** when running with default parameters against the experiment segment of the traces with no worm traffic injected

hour has elapsed or the worm is detected. Each of the 16 experiments that we run for a given set of worm parameters has a different host in the protected network being infected first and uses a different random seed. The percentage of experiments where the worm is not detected is the false negative rate, and the mean number of worm connections that have left the network at detection time is the detection latency.

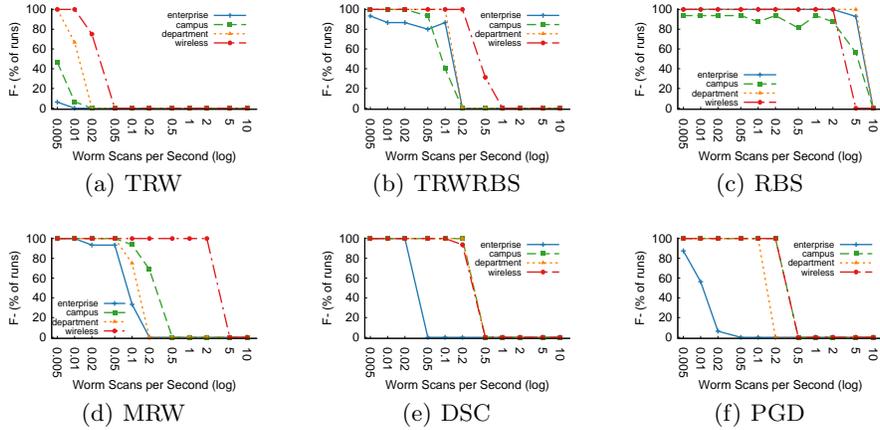
## 6 Results

We now measure the performance of the worm detectors in a variety of worm scenarios. We first look at the false positives, then introduce worm traffic to measure false negative rates and detection latency. We start with the simplest worm strategy of randomly scanning addresses, then increase the worm sophistication to local-preference and then topologically aware scanning strategies.

### 6.1 False Positives Against Legitimate Traffic

Figure 1(a) shows the results for each detector using default parameters from its original publication. Raising an alarm for a host could either (a) indicate that the host is considered permanently infected, or (b) indicate that the host is behaving anomalously *now* (for some definition of now). Figure 1(a) shows F+ results using strategy (a) (with PGD limited to one alarm per 1-minute window because it does not identify the infected host). Figure 1(b) shows F+ results using strategy (b) and with an alarm duration of one minute. Strategy (a) is probably more representative of how detectors would be deployed in practice, but it is illustrative to show that without such a limitation, in some environments RBS and TRWRBS would be in an alarm state more than 50% of the time and TRW and MRW would be in an alarm state 100% of the time.

These results also demonstrate the impact that environment has on the detector performance. TRWRBS has five F+ in the wireless environment but none in the campus or department environments. MRW is in an alarm state 100%



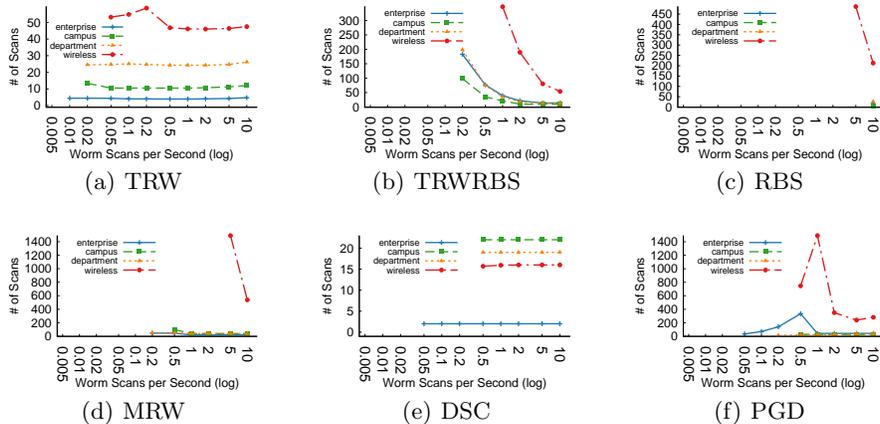
**Fig. 2. F- against random worm:** percent of experiments where the worm was **not** detected (lower is better performance) with a *random* scanning worm infecting randomly selected hosts. For each environment and scanning rate we conducted 16 individual experiments using different first infected hosts and different random seeds. In each case the experiment was run until the worm was detected or one hour elapsed without detection.

of the time in the department environment but not at all in the campus environment. An evaluation using only a single environment could produce grossly inaccurate results.

The wireless environment showed the most F+ activity with the default parameter choices. This appears to stem from several hosts playing network games such as Counter-Strike (UDP connections on ports in the 27010-27050 range) and NeverWinter Nights (TCP connections on port 5121) as well as from hosts using BitTorrent (33 hosts active on ports in the 6881-6999 range). This environment represents the most residential/recreational usage patterns and indicates that this sort of traffic is less amenable to behavior-based worm detection than the less variable traffic of the enterprise environment. This represents the first findings we are aware of that validate a common hypothesis: current behavior-based anomaly detectors are not optimized for residential style network traffic and may not show satisfactory performance in such an environment.

## 6.2 Detector Performance Against Random Worm

In this section we report false negative and latency results against random scanning worms. Figure 2 shows that TRW is the most consistently effective detector across the environments, discovering all instances of the worm down to 0.05 scans per second and catching the majority of the slower scans in the enterprise and campus environments. RBS is the least effective, only able to consistently detect the worm scan rates greater than five scans per second. TRWRBS blends the

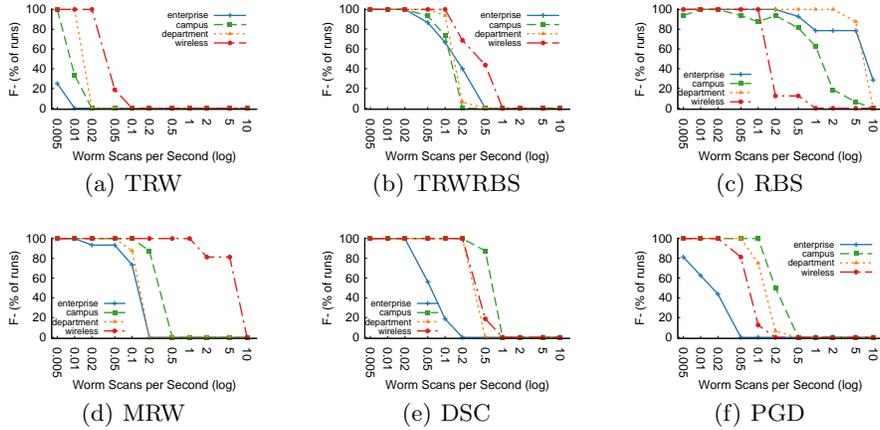


**Fig. 3. Latency against random worm:** from worm infection time to detection time for *random* scanning worm, measured as the number of worm connections leaving the protected network prior to detection. We report results only for those environments and scan rates where the worm was detected with 100% accuracy.

two detectors with results right in the middle. The DSC and PGD detectors are an order of magnitude more effective in the enterprise environment than in the other environments due to the lower activity levels (and hence lower thresholds) in the enterprise environment. The MRW detector provides middle of the road performance except against in the wireless environment where it is unable to detect the worm at speeds slower than five scans per second.

Figure 3 shows the average number of connections each infected network was able to make before detection. Note that the scale is not consistent across the graphs. We only show the value for those scenarios where  $F^-$  is zero in order to eliminate selection bias in the results. DSC is consistently the fastest detection mechanism, never allowing the worm to scan more than 23 times before detection. TRW again highlights the variation between environments, allowing roughly 50 worm scans in the wireless environment before detection, but only five scans in the enterprise environment. MRW and RBS allowed several hundred scans before detection in the wireless environment, but were much faster in the other environments. PGD showed the most variation, allowing over 1000 scans before detection in some scenarios in the wireless environment but detecting the worm in 30-40 connections in the other environments. TRWRBS showed increasing latency as the scan rate drops. This is due to the influence of the RBS algorithm that increases the destination threshold as the time window increases. The fast scanning worm is caught in a short window, but the slower scanning worms take a substantially longer time to hit the critical number of destinations.

Across the board, TRW shows the best detection performance against random scanning worms. This indicates that connection failures are a strong and highly identifiable signal. TRW also had consistent and low latencies, limiting



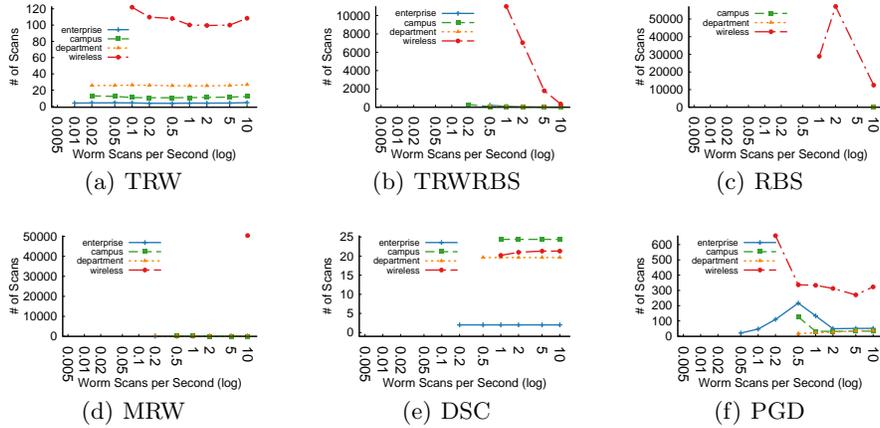
**Fig. 4. F- against local-preference worm:** percent of experiments where the worm was **not** detected (lower is better performance) with a *local-preference* scanning worm.

the damage a worm could do. Destination pattern based detection such as MRW and RBS typically requires greater numbers of connections for accurate identification. PGD performed adequately, but is designed to detect multiple infected internal hosts which did not happen with the random-scanning worm.

### 6.3 Detector Performance Against Local-Preference Worms

Having examined the baseline case using the random scanning worm, we now investigate performance against a more advanced foe: the local-preference scanning worm. The local-preference worm directs half its connections at the local network, meaning both that it is more likely to infect multiple hosts inside the protected network and that fewer connections per time period are visible to a gateway-based detector. However, the scan is still random in nature, so shares the same general characteristics as the purely random scanning worm.

Figure 4 shows that for most scenarios, the detectors show a slight decrease in sensitivity. This is visible as a shift to the right in the false negative curves. The TRW detector was able to detect 100% of the random worms in the wireless environment at 0.05 scans per second, but is only able to detect 100% of the local-preference worms at 0.1 scans per second. TRWRBS, RBS, MRW, and DSC all show similar decreases in performance in some environments. The reason for this is simply the reduction of worm scans that are visible to the detector. The limit of a detector’s ability to spot the worm—meaning the slowest worm that it can detect reliably—is at the point where it can just barely observe enough evidence to infer that a host is infected. If a worm scans more slowly or not all its scans cross the gateway (as in local-preference worms), the evidence visible to the detector may not be enough to make the determination that a worm is present.



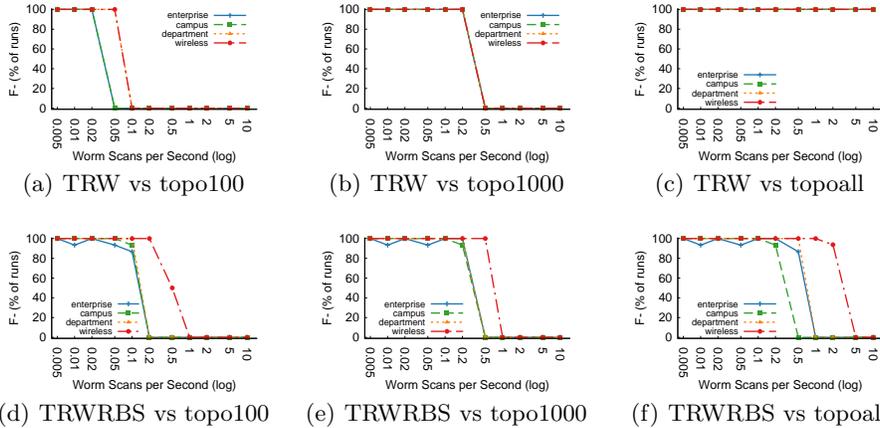
**Fig. 5. Latency against local-preference worm:** from worm infection time to detection time for *local-preference* scanning worm, measured as the number of worm connections leaving the protected network prior to detection.

The one detector that shows a significantly different response is the PGD detector, showing *better* performance against the local-preference worm than it did against the random worm. The PGD detector measures the protocol graph of all hosts in the network, and the more infected hosts there are, the more scanning there will be using the protocol the worm targets. This leads to either more total nodes in the graph or a larger connected component, allowing the PGD detector to spot the local-preference worm in situations where it would not have detected a random scanning worm.

The latency results are also impacted by the local-preference scanning strategy (Figure 5). The TRWRBS, RBS, DSC, and MRW detectors show worse detection latency in all environments for the local-preference worm as compared to the random worm. This is because the worm targets the local network so aggressively that in many scenarios it infects multiple hosts inside the network before it is detected. Recall that our latency metric measures the combined external scanning of *all* infected hosts in the network. The TRW detector, on the other hand, shows identical latency performance for all environments when comparing random and local-preference worms because it detects the worm before it infects *multiple* hosts (except in the wireless environment).

PGD behaves quite differently than the other detectors. It detects the local-preference worm more quickly than the random worm in the enterprise and campus environments, but slower in the department environment. And in the wireless environment the local-preference worm is detected more quickly at scanning rates of two scans per second or less, but the random worm is detected more quickly at rates above two scans per second.

The DSC detector is the fastest, allowing fewer than 25 outgoing worm connections in all scenarios where it was able to detect the worm 100% of the time.



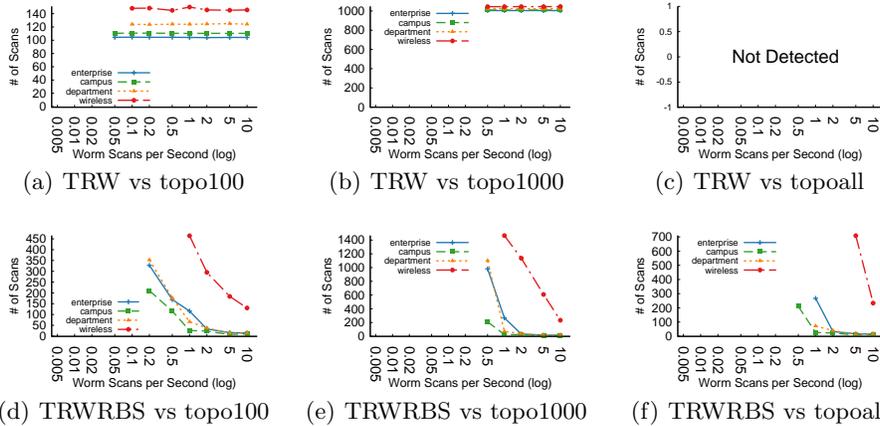
**Fig. 6. F- against topo worm:** percent of experiments where the worm was **not** detected (lower is better performance) by the TRW and TRWRBS detectors with a *topo* scanning worm. The *topo100* worm uses 100 neighbors before reverting to random scanning, the *topo1000* worm uses 1000 neighbors before reverting to random scanning, and the *topoall* worm never uses random scanning.

TRW is also quite fast, allowing fewer than 27 connections in all environment except for the wireless environment where it allows roughly 100. Note TRW also is the most sensitive detector, successfully detecting the worm at the lowest scanning rates in all environments.

#### 6.4 Detector Performance Against Topo Worms

Topo scanning changes the observed behavior of an infected host by reducing the number of connection failures that the detector can observe. The neighbors discovered by the topo worm are vulnerable at the same level as other hosts in the network but are guaranteed to be present, different from random scanning where a large number of scans go to addresses with no host present. The only detectors that are impacted by this strategy are those detectors that rely on observing connection failures: TRW and TRWRBS. The RBS, MRW, DSC, and PGD detectors show identical performance against the topo worm and the random worm. The pattern of neighbors—whether they can be connected to or not—is random in both the random and topo worms and thus triggers those algorithms in the same way.

The TRW detector is unable to detect the topo worm during its topo scanning phase because of the lack of connection failures. It *only* detects the worm after it reverts to random scanning. In the *topo100* scenario (Figure 6(a)), this occurs relatively quickly as it does not take long for the worm to exhaust its list of 100 neighbors. TRW is able to detect the worm at speeds as low as 0.01 scans per second in all environments. However, in the *topo1000* scenario, the list of



**Fig. 7. Latency against topo worm:** from worm infection time to detection time for *topo* scanning worm, measured as the number of worm connections leaving the protected network prior to detection. The *topo100* worm uses 100 neighbors before reverting to random scanning, the *topo1000* worm uses 1000 neighbors before reverting to random scanning, and the *topoall* worm never uses random scanning.

neighbors is not exhausted during the one-hour experiment for speeds below 0.5 scans per second and the TRW detector is unable to detect *topo* worms with slower scanning rates (Figure 6(a)). In the *topoall* scenario—where the *topo* worm never exhausts its list of neighbors—the TRW detector is *never* successful at detecting the worm (Figure 6(c)).

Not only is TRW’s ability to detect the worm compromised, but even in scenarios where it does detect the worm it is much slower at it. Figures 7 show the latency results for TRW against the *topo* worm. Because during the worm’s *topo* phase none of its scans were detected, the latency results against the *topo100* worm are approximately 100 scans worse than they were for TRW against the random scanning worm. Similar results can be seen for the *topo1000* scenario, where TRW’s detection latency is 1000 connections worse than it was for the random scanning worm.

This shortcoming in TRW is one of the motivations for the TRWRBS detector. It uses connection failures in the detection algorithm, but it can also detect a worm even with no connection failures by checking the rate of connections to new destinations. The TRWRBS detector is able to detect the *topo100* at rates above 1 scan per second in the wireless environment and above 0.2 scans per second in all other environments (Figure 6(d)). It does not perform quite as well as TRW in this scenario because TRW is able to leverage the connection failures so effectively. In the *topo1000* scenario the detectors are effective at approximately the same worm scanning rate (Figure 6(e)); but if one looks at the latency, the TRWRBS detector is able to detect the worm more quickly at most scanning rates (Figure 7(e)). At worm scanning rates of 2 scans per

second and higher, TRWRBS can detect the worm in under 30 connections in all the environments except for the wireless environment. This compares well against the TRW algorithm which requires over 1000 scans before detecting the topo1000 worm. The TRWRBS detector even detects the worm in the topoall scenario where the TRW detector could not.

This reliance on connection failures highlights a potential weakness of the TRW algorithm. If a worm can generate a big enough list of hosts running the target service that are likely to exist, it can make enough successful connections to completely evade the TRW algorithm. The detectors based on destination distributions do not have this weakness.

## 6.5 Summary

We now recap our findings and answer the questions posed in the introduction. We found that no detector was clearly superior to the others in the study. The TRW detector can detect slower random and local-preference scanning worms than any of the other detectors in all the environments we tested. However, it performs poorly against topo worms. In fact, a topo worm with a large supply of neighbors to scan is entirely undetectable by the TRW algorithm. The PGD detector was capable of detecting all types of worms scanning at 0.5 scans per second or faster in all environments, but was relatively slow, frequently allowing several hundred scans prior to detection. The TRWRBS detector was similar to the PGD detector, but showed decreased performance against topo worms. The RBS detector was only capable of detecting fast scanning worms. The MRW detector struggled to detect worms in the wireless environment and was incapable of detecting the local-preference worm in that environment. Finally, the DSC detector performed quite well in many respects, but is otherwise quite limited due to the requirement that an inbound infecting connection be observed in order for the detector to function. An initial infection that came via some other vector (removable media, direct download, etc.) would be undetectable by DSC.

If we narrow our criteria, however, we may be able to identify some detectors as being superior at specific tasks. For example, if we only consider fast scanning worms—those that make 10 scans per second—the TRWRBS detector suddenly stands out as being an excellent choice. It detects fast scanning worms in every environment regardless of scanning type and is the fastest in most scenarios.

The wireless environment was the most difficult for detectors to operate successfully in. In virtually all scenarios, detectors showed the worst sensitivity in the wireless environment, and detection latencies were typically an order of magnitude worse. The traffic in this environment is more focused around entertainment type activities such as network gaming and peer-to-peer file sharing. These activities are prone to resembling worm scanning activity, making it more difficult for the detectors to differentiate between legitimate hosts and worm infected ones. For example, a peer-to-peer network client may receive a list of peers who were recently active and attempt to contact every host on the list. If the peer-to-peer network has a high churn rate and hosts on the peer list have left the network, this activity will result in many connection failures, just as if a

worm were scanning for potential targets. Even in the face of this type of activity, however, the detectors were still typically able to detect true worm activity. As in the other environments, the TRW detector was able to detect slower worms than any other detectors. The PGD detector showed the next best performance and had the advantage of also detecting the topo worm in the wireless environment.

Our results indicate that worms scanning at one connection per second or better are relatively easily detected in most environments, but a worm that utilizes some sort of topo scanning with a low connection failure rate could evade worm detectors in all our tested environments—if it scanned at a rate no greater than 1 scan per 10 seconds.

## 7 Related Work

The most directly related work to ours—aside from the original publication of the detectors evaluated here—is a study by M. Patrick Collins and Michael K. Reiter that evaluates behavior-based (or payload-oblivious as they term it) detectors [30]. This work is closely tied to ours, but is complementary in nature. Their work, like ours, evaluates the effectiveness of several behavior-based detectors. The key distinction is that instead of monitoring an *internal* network for infections, they considered the performance of these systems in detecting incoming scanning from *external* networks. This is actually a substantially different problem than detecting internally infected hosts. There is a considerable volume of incoming scan traffic to most networks [31], and separating worm scanning from other scanning traffic is a different problem than detecting outgoing scans among legitimate outgoing traffic. They developed new metrics for their evaluation, measuring an attacker’s payoff over an observable attack space. This new metric does not apply well to the job of detecting internal scanners, however, as the target address space of an internal scanner is potentially the entire IPv4 address space.

A work by Li, Salour, and Su surveys behavior and content-based worm detectors [32] and covers many of the works referenced here. They do not measure the performance of detectors, however, limiting their study to describing and classifying them instead. Our work briefly addresses broad classifications of detectors, but then focuses on their relative performance in real world situations.

## 8 Conclusions

The relative lack of worm attacks in recent years has caused network operators to focus their attention on other threats. However, Conficker and IKEE.B illustrate the continued threat that worms pose. Lapses in worm activity are not new—13 years separated the Morris worm from the series of large worm outbreaks in the early 2000’s—and continued vigilance is required to protect our networks.

Despite the large number of worm detectors published, it is still unclear whether state-of-the-art systems are capable of coping with modern worms successfully. It is even unclear how these systems compare to each other. We have

not seen a systematic comparison study that evaluates worm detectors against the same performance metrics across the same parameter values.

This paper addresses that issue. We focus on behavior-based worm detectors under different real-world environments, studying their false positive, false negative, and latency in detecting worms at various scanning rates using random, local-preference, or topological-aware scanning methods. We found that worms that scan at a low rate are the hardest to detect; for example, a topologically aware worm scanning one destination per minute can evade all tested detectors in all environments. Also, among all the environments we studied, the wireless environment poses the biggest challenge, where almost every detector incurs a lower—sometimes unacceptable—accuracy and higher latency than in other environments. No detector is a clear winner; TRW performs the best against the random and local-preference worms, for example, but it fails badly at detecting a topologically aware worm.

## References

1. Eisenberg, T., Gries, D., Hartmanis, J., Holcomb, D., Lynn, M.S., Santoro, T.: The Cornell commission: on Morris and the worm. *Communications of the ACM* **32**(6) (1989) 706–709
2. Moore, D., Shannon, C., Claffy, K.C.: Code-red: A case study on the spread and victims of an Internet worm. In: *Proceedings of the ACM Internet Measurement Workshop*. (2002) 273–284
3. Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., Weaver, N.: Inside the slammer worm. *IEEE Security and Privacy* **1**(4) (2003) 33–39
4. Symantec, I.: The downadup codex. Technical report, Symantec (March 2009)
5. Porras, P.A., Saidi, H., Yegneswaran, V.: An analysis of the ikee.b (duh) iPhone botnet. Technical report, SRI International (December 2009)
6. Sekar, V., Xie, Y., Reiter, M.K., Zhang, H.: A multi-resolution approach for worm detection and containment. In: *Proceedings of the International Conference on Dependable Systems and Networks*. (2006)
7. Schechter, S.E., Jung, J., Berger, A.W.: Fast detection of scanning worm infections. In: *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. (2004)
8. Gu, G., Sharif, M., Qin, X., Dagon, D., Lee, W., Riley, G.: Worm detection, early warning and response based on local victim information. In: *Proceedings of the Annual Computer Security Applications Conference*. (2004)
9. Liang, Z., Sekar, R.: Fast and automated generation of attack signatures: A basis for building self-protecting servers. In: *Proceedings of the Conference on Computer and Communications Security*. (2005)
10. Crandall, J.R., Su, Z., Wu, S.F., Chong, F.T.: On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In: *Proceedings of the Conference on Computer and Communications Security*. (2005)
11. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: *Proceedings of the Network and Distributed System Security Symposium*. (February 2005)
12. Tucek, J., Newsome, J., Lu, S., Huang, C., Xanthos, S., Brumley, D., Zhou, Y., Song, D.: Sweeper: A lightweight end-to-end system for defending against fast worms. In: *Proceedings of the EuroSys Conference*. (2007)

13. Kim, H.A., Karp, B.: Autograph: Toward automated, distributed worm signature detection. In: Proceedings of the USENIX Security Symposium. (August 2004) 271–286
14. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: Proceedings of the Symposium on Operating System Design and Implementation. (2004) 45–60
15. Wang, K., Cretu, G., Stolfo, S.J.: Anomalous payload-based worm detection and signature generation. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection. (2005)
16. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A content anomaly detector resistant to mimicry attack. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection. (2006)
17. Li, Z., Wang, L., Chen, Y., Fu, Z.: Network-based and attack-resilient length signature generation for zero-day polymorphic worms. In: Proceedings of the IEEE International Conference on Network Protocols. (October 2007) 164–173
18. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: Proceedings of the IEEE Symposium on Security and Privacy. (2005)
19. Mason, J., Small, S., Monroe, F., MacManus, G.: English shellcode. In: Proceedings of the Conference on Computer and Communications Security. (2009) 524–533
20. Jung, J., Milito, R., Paxson, V.: On the adaptive real-time detection of fast-propagating network worms. In: Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment. (July 2007) 175–192
21. Collins, M.P., Reiter, M.K.: Hit-list worm detection and bot identification in large networks using protocol graphs. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection. (September 2007) 276–295
22. Wu, J., Vangala, S., Gao, L., Kwiat, K.: An effective architecture and algorithm for detecting worms with various scan techniques. In: Proceedings of the Network and Distributed System Security Symposium. (2004)
23. Zou, C.C., Gong, W., Towsley, D., Gao, L.: The monitoring and early detection of Internet worms. *ACM Transactions on Networking* (2005)
24. Weaver, N., Staniford, S., Paxson, V.: Very fast containment of scanning worms. In: Proceedings of the USENIX Security Symposium. (2004) 29–44
25. : DETER: Cyber defense technology experiment research (DETER) network. <http://www.isi.edu/deter/>
26. Stafford, S., Li, J., Ehrenkranz, T., Knickerbocker, P.: GLOWS: A high-fidelity worm simulator. Technical Report CIS-TR-2006-11, University of Oregon (2006)
27. : LBNL/ICSI enterprise tracing project. <http://www.icir.org/enterprise-tracing/> (2005)
28. Group, W.N.R.: WAND WITS: Auckland-IV trace data. <http://wand.cs.waikato.ac.nz/wand/wits/auck/4/> (April 2001)
29. : Umass trace repository. <http://traces.cs.umass.edu/>
30. Collins, M.P., Reiter, M.K.: On the limits of payload-oblivious network attack detection. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection. (September 2008) 251–270
31. Allman, M., Paxson, V., Terrell, J.: A brief history of scanning. In: Proceedings of the ACM Internet Measurement Conference. (October 2007) 77–82
32. Li, P., Salour, M., Su, X.: A survey of internet worm detection and containment. *IEEE Communications Society Surveys and Tutorials* **10**(1) (2008) 20–35