SWORD: Self-propagating Worm Observation and Rapid Detection*

Jun Li, Shad Stafford, and Toby Ehrenkranz
Department of Computer and Information Science
University of Oregon
{lijun, staffors, tehrenkr}@cs.uoregon.edu

Abstract

As the launching of a worm can have disastrous effects on the Internet in just minutes, it is essential to automatically and reliably detect worms in their early stages. In contrast to content-based approaches, in this paper we study the feasibility of a behavior-based solution through our SWORD framework. As SWORD does not inspect the payload of traffic, it is resilient against polymorphic worms and avoids the expense of examining traffic payload.

We focus on three algorithms embraced in the SWORD framework: the causal similarity identification algorithm, destination address distribution analysis algorithm, and continuity analysis algorithm. We investigate how they may identify worm-like connections and raise an alarm by identifying *essential* behaviors that a worm must display. Our evaluation shows that SWORD exhibits promise in quickly, accurately, and efficiently detecting self-propagating worms of different speeds and scanning methods. We also point out extensions to SWORD that can detect infected hosts and classify a worm based on its behavior. Although some limitations and open issues remain, SWORD is an important step toward detecting zero-day self-propagating worms via a behavior-based approach.

1 Introduction

The ability to detect self-propagating Internet worms is key to the security of the Internet. Worms can infect millions of hosts in just minutes [1], making manual inspection of traffic essentially useless for worm detection. One could automatically scan traffic payloads for known worm signatures, but as Internet hosts continue to have new vulnerabilities, worms exploiting those vulnerabilities will continue to appear in unknown forms. One could check traffic in real time for suspicious byte patterns (*e.g.*, [2, 3, 4]), but worms can be polymorphic with an almost arbitrary payload [5], limiting the effectiveness of such schemes.

These difficulties have inspired a different paradigm for worm detection: monitoring and analysis of worm *behavior*. Existing solutions include monitoring unexpected traffic to unused IP addresses (*e.g.*, [6, 7]) or honeypots (*e.g.*, [8, 9]), watching for error messages (*e.g.*, ICMP Unreachable [10] or TCP Reset [11]) or the lack of DNS queries ([12]), or observing similar outgoing and incoming connections at end hosts (*e.g.*, [13, 14]). Unfortunately, while these solutions are effective against certain worms, the behaviors they look for are either avoidable by worms (worms can choose to only contact hosts in actual use, try to avoid error messages, as well as force DNS queries), or often seen in legitimate applications (*e.g.*, web proxies or SMTP servers can also have identical incoming and outgoing connections).

We tackle this difficult problem via a unique approach. We design, develop, and evaluate an **essential-behavior-based** worm detection framework called SELF-PROPAGATING WORM OBSERVATION AND RAPID DETECTION, or **SWORD**. It runs at the gateway point of an administrative domain to monitor the domain's inbound and outbound traffic. It is not only *content-independent*—thus resilient to worms with polymorphic

*This research is being funded by NSF CAREER award CNS-0644434 and a research grant from Intel Corporation.

payloads, but also departs from existing behavior-based approaches: SWORD focuses on understanding the essential or nearly essential worm behavior—*without which self-propagating worms cannot propagate or propagate too slowly to put the Internet infrastructure at risk*—and the detection of worms based on them. Toward this end, SWORD identifies and studies the following four distinct but complementary behaviors of the *connections* from worms:

- **Causal relationship.** A worm *must* propagate from host to host. While it propagates, a transitive causal ordering will arise: an outbound worm connection from an infected host must be preceded by an earlier infection connection toward, or from, the host.
- **Self-similarity.** Any worm *must* exploit a certain vulnerability to gain control of the target host. With a finite number of extant vulnerabilities, a worm will quickly repeat its attacks using the same vulnerability and these attacks will be similar. Note, the payload itself is not necessarily similar since a worm can be polymorphic, but some aspects of the connection must be similar.
- **Greedy destination visiting pattern.** A worm *will* attempt to infect a large number of victims. A normal host typically visits a small number of destinations most of the time, but an infected host will connect to a larger number of destinations as it spreads.
- **Continuity.** A worm *will* continually initiate new worm connections. The rate at which these connections are initiated may be very slow, but as more and more hosts of a domain become infected a growing number of worm connections will cross the gateway of an infected domain.

A behavior is **essential** if a worm cannot propagate without it, or **nearly essential** if a worm must significantly slow down in order to avoid demonstrating the behavior. For the above four behaviors, the first two are essential, and the last two are nearly so. As we will often study the first two in unison, we refer to them together as **causal similarity** in this paper.

Note that SWORD is intended to discover only self-propagating worms that initiate their connections by themselves; worms that piggy-back on existing connections or rely on user interaction will not necessarily exhibit the behavior SWORD looks for.

The primary challenge of this research is that we must accurately and quickly extract worm behaviors from an infected domain's inbound and outbound traffic while not falsely identifying normal traffic as worm-like, yet we must first formalize or quantify these essential worm behaviors. In this paper, we focus on the following *core* components of the SWORD framework: **(i) Causal similarity identification:** How can a worm monitor determine whether a connection is likely caused by one or more previous connections *and* that the connection is similar to them? **(ii) Destination address distribution analysis:** How can a worm monitor determine whether the current destination visiting pattern deviates from the normal pattern such that it signifies suspicious worm behavior? Underneath this question, what is a *normal* pattern? and **(iii) Continuity analysis:** What would be an effective mechanism to analyze recent patterns of worm-like connections and distinguish between the pattern of worm-like but legitimate connections and the pattern of the true worm connections? We also discuss two *extended* components of the SWORD framework: **(iv) Infected host detection:** Can a worm monitor determine which hosts are infected by an ongoing worm? **(v) Behavior-based worm classification:** Once SWORD detects a worm is occurring, can it further tell what type of worm it is based on its behavior?

Our SWORD prototype is not only effective at detecting worms based on their behaviors, but also computationally light-weight and easy to deploy and administer. SWORD requires deployment only at the gateway point of an administrative domain and it limits its traffic analysis to the flow level, avoiding the expense of examining payload information. While it maintains a non-trivial amount of internal state, it requires quite manageable amounts of computation on a per-connection basis, making it scalable to large

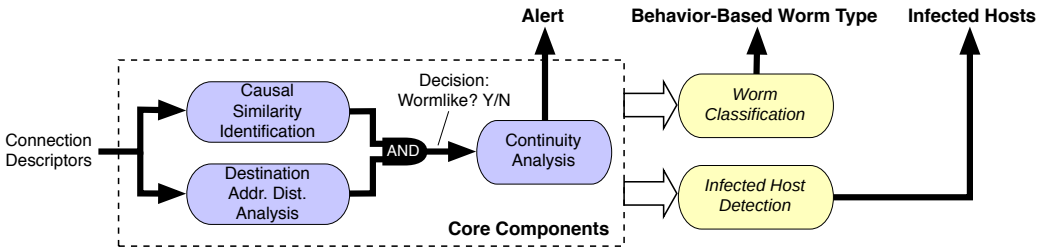


Figure 1: **The architecture of SWORD.**

domains. Our evaluation through trace-based simulation has shown that SWORD is not only light-weight and fast, but also accurate in detecting both high-speed and low-speed simulated worms, where the worms utilize a variety of scanning mechanisms via either TCP or UDP in populations of high or low vulnerability.

We elaborate, evaluate, and discuss our approach and SWORD in the following sections. We first describe our algorithms that detect the manifestations of these essential worm behaviors in Sec. 2, then discuss detecting infected hosts and classifying worms based on their behavior in Sec. 6. We then introduce our methodology for evaluating SWORD in Sec. 4, with the results and analysis regarding SWORD's speed and accuracy as well as its overhead in Sec. 5. We compare SWORD with related research in Sec. 7 and conclude our paper in Sec. 8.

2 Algorithms of SWORD

The objective of the SWORD framework is to observe essential behaviors of self-propagating worms and detect them based on these behaviors. Its observation and detection process must be *efficient*, *fast*, and *accurate*. Moreover, it should be *resilient* against the desperate efforts of worm authors in thwarting the efficacy of the framework.

We assume SWORD monitors connections at a domain's gateway, minimally intrusive for deployment yet giving it a suitable vantage point to observe all connections into and out of the network. This scheme requires far less administration than solutions that require installation on end hosts, and does not need to monitor traffic that stays within the internal network.

Connections that SWORD operates on are any kind of flows between two hosts. It can be either a TCP connection, or defined by some number of UDP packets with some temporal proximity and sharing common source and destination addresses and ports. SWORD coalesces the header information from individual IP packets of the flow into what we call a **connection descriptor**, which contains the basic information about the connection such as the source and destination address and ports and any TCP flags seen. This connection descriptor is fed into SWORD for classification.

The detection procedure of SWORD consists of three core components (Figure 1). It first classifies connections as worm-like or legitimate by applying the causal similarity identification algorithm (Sec. 2.1) *and* the destination address distribution analysis algorithm (Sec. 2.2): if both regard a connection as conforming to the behavior of a worm, the connection is worm-like. However, a legitimate connection may be mislabeled as worm-like. SWORD further applies the continuity analysis algorithm (Sec. 2.3) to determine if the pattern of recent worm-like connections matches that of a worm: if so, it then raises an alert of worm occurrence. Here, all these algorithms are completely agnostic to the payload of the connections, so are entirely robust against polymorphic worms.

As shown in Figure 1, there are also two extended components of SWORD: *Detecting infected hosts* (Sec. 3.1) that is built on top of the three core components, and *Classifying worms based on their behavior* (Sec. 3.2) that is a significant add-on. These two components are not the focus of this paper; we will describe them at a high level in Sec. 3.

2.1 Causal Similarity Identification Algorithm

The Causal Similarity Identification Algorithm identifies a connection as worm-like when it observes blocks of similar connections along a potential causality tree. The motivation behind this is as follows: every time a worm attacks a remote host, it must connect in such a way so as to compromise the target host. Multiple attacks on a specific vulnerability, either from a single host or from multiple hosts, all compromise the target host in the same way and therefore all must use connections that are *similar* to each other. Of course non-worm connections may also exhibit some level of similarity; in fact if you compare a connection against enough other connections, you are essentially guaranteed to find another similar connection. The key distinction, however, is that non-worm connections are not *guaranteed* to be similar like worm connections are. Therefore, if we can compare a given connection against some set of connections that would be relevant if a worm were present, and not find similarity, we know that the connection in question is not a worm connection. In this way, this algorithm works somewhat like a filter, separating out traffic that is distinctly non-worm-like. Understanding that, the issues that remain are to identify a meaningful and useful measure of connection similarity and a method of limiting the connections we compare against to a relevant subset of those we have available. We address both of these issues below.

The first issue to resolve is "What makes a connection *similar* to another connection?" in some meaningful way. A very fine-grained similarity comparison would look at every attribute of the connection right down to the payload it carried. This in fact is conceptually similar to many existing worm-detection systems, which look for similar payloads which can be used to build byte-stream signatures. This approach has already been shown to have some limitations though, because encrypted or polymorphic payloads will not exhibit any similarity to each other. We therefore need to include only those attributes that cannot easily be modified while still attacking a given vulnerability. The connection attributes that we believe will be constant over any type of worm attack are listed in Table 1. Our algorithm considers two connections to be similar if they have the same value for all the attributes listed. Simple inspection of this list will lead the reader to the conclusion that if one were to simply compare random connections, one would still find a great many that show similarity by this measure. This is in fact, the case, and this algorithm—if used in isolation—would show many false positives for legitimate connections. Similarity is necessary but not sufficient for connections to be worm connections. We can cut down the number of connections identified by limiting the scope of connections that we compare against to those that would be relevant to worm propagation.

Table 1: Connection attributes considered for similarity comparison

Attribute	*Value*
Protocol	TCP/UDP
Dest Port	Port #
TCP SYN	Yes/No
TCP SYN/ACK	Yes/No
TCP Connection	Yes/No
TCP Src FIN	Yes/No
TCP Src FIN ACK	Yes/No
TCP Dest FIN	Yes/No
TCP Dest FIN ACK	Yes/No
TCP RST	Yes/No

("Yes/No": is the TCP flag seen?)

How then, do we determine which connections are relevant to worm propagation? The first step is to imagine the scenario that takes place during a worm infection. Let a connection from host H_1 towards host H_2 be represented as $H_1 \rightarrow H_2$. Suppose host A becomes infected, it will then begin scanning for additional remote hosts to infect. When it infects host B through connection $A \rightarrow B$, B will also begin scanning and will eventually infect host C though $B \rightarrow C$. If we then examine some connection $C \rightarrow X$, the connections we would most like to compare it to for similarity are the infecting connections: $B \rightarrow C$ and $A \rightarrow B$, and any other worm scans made by host B or host A. In practice of course, it is impossible for our monitor to know the complete infection path of the worm and compare connections against only worm connections. To do so would require complete knowledge of network traffic and an ability to identify which connections are worm connections with perfect accuracy. Instead, SWORD actually has a somewhat limited view of the network, seeing only those connections that cross the gateway of its protected domain, and not knowing with any certainty which of those connections might represent a zero-day worm attack. SWORD leverages its limited knowledge to the fullest extent by maintaining a graph of connections—from those it sees—that represents

potential causation. This graph allows us to compare a connection with those connections that may have been the infecting connection or a worm scan from the infecting host. This *causal connection graph* is a directed graph where each node represents a connection and each edge represents a potential causation. To limit the in-degree of each node, only the most recent potentially causal connections are connected as direct parents. We can then use this graph to find all the potentially causal connections for a given connection by examining all of its ancestors.

The causal connection graph is maintained by SWORD as follows: Each new connection that is examined by SWORD is connected to the graph by a set of parent-connections that *could* have caused this connection to occur. Specifically, for a new connection from a given source-host (inside or outside the protected domain), parent-connections will include the most recent outgoing connection from the source-host and all of the inbound connections to the source-host that have occurred since the last outbound connection from it. The following example illustrates the reasoning behind constructing the graph in this way. For a given connection, say $C \rightarrow X$, if it is a worm connection, its causal connection must be one of two forms: $* \rightarrow C$ (this connection infects C, then C initiates $C \rightarrow X$ to infect X), or $C \rightarrow *$ (C is likely sending out a series of infections; or, C actually infects someone else first, and learns X is also a victim to infect). Furthermore, the causal relationship is *transitive*; if two connections, ω_1 and ω_2 cause a connection ω_{new}, and ω_1 causes ω_2, we only need to use ω_2 as the parent of ω_{new} on the graph. Finally, because the *causing* connection must happen before the *caused* connection, *SWORD ensures that every edge of the graph is a happened-before relation.* This way, when the causing connection and the caused connection are represented by two nodes in the graph, there will always be a directed path between the two nodes. These conditions allow SWORD to efficiently add new connections to the graph by maintaining short lists of parent connections from or to each protected host in the network. To keep the number of ancestors of a given node tractable, we also limit the overall size of the connection graph (currently to 20,000 nodes) by dropping the oldest connection when we add a new connection.

Using this causal connection graph allows SWORD to detect worm-like connections by comparing a given connection with its ancestors in the graph. When more than n ancestor connections are similar (we currently use 8 for the value of n), we consider the current connection to be worm-like. Despite the limited range of connections that we compare against, this algorithm identifies more than an insignificant number of legitimate connections as worm-like. On its own it would not be an effective worm detector, *however, the causal similarity algorithm works synergistically with the destination address distribution algorithm outlined below. Their combination is very effective at detecting worm-like behavior with few false positives*.

2.2 Destination Address Distribution Analysis Algorithm

The Destination Address Distribution Analysis algorithm identifies a connection as worm-like when that connection is part of a trend of connections changing the shape of a host's destination distribution pattern. It is based on another behavior of self-propagating worms: the desire to propagate to additional hosts. To propagate, a worm-infected host must make connections to a large number of remote addresses, often making only a single connection to each one. Whether these connections happen quickly or slowly, the behavior is in contrast to the behavior of a normal host, which is to contact a relatively smaller number of hosts and make multiple connections to many of the destinations.

The question then, is how to accurately and effectively distinguish the worm-like behavior from normal behavior. There are many methods to measure the destination address distribution of a host, from measuring the percentage of destinations that are visited only once, to plotting the frequency at which hosts are visited. Exploratory analysis—omitted here for brevity—has led us to use a ranking-based analysis algorithm. It works as follows: We first collect the destination addresses visited by a given host over some recent connection history, and then rank them from most-visited to least-visited. We then plot the ranks versus the number of visits to each destination on a log-log scale and analyze the resulting line. Normal traffic distribution

from an uninfected host typically shows a very strong linear correlation between the log of the rank and the log of the visit count. When the host becomes infected with a worm, however, the correlation becomes less strong as the host makes connections to more destinations. By measuring the trend of this correlation, we can observe the presence of a worm.

A more formal description of the algorithm is as follows: For the last S connections from a given host, suppose that N_d is the number of connections that the host initiates toward destination d. Sort the N_d values for all destinations in S in decreasing order. R_d is defined as the rank of N_d in the sorted list of all N_ds. We then measure the correlation coefficient of $logR_d$ vs. $logN_d$ for all d in S. S is a constant size, so that each new connection in effect slides the window of connections under examination. If a given connection is part of a *trend* of connections for which the correlation coefficient is decreasing, this algorithm considers *that connection* to be worm-like!

Implementation of this algorithm requires the resolution of further details: the size of the connection history to measure and what constitutes a trend away from a strong correlation. In determining the best history size, intuitively we can see that the impact of a single connection is smaller for a longer history and greater for a shorter history. Too small a history, however, makes the algorithm overly sensitive such that normal traffic is less distinctive. We determine that a significant trend is occurring when α out of β connections decrease the correlation. This ratio of $\frac{\alpha}{\beta}$ must be selected carefully. If the ratio is too high, worm traffic might not be caught when legitimate traffic is interspersed with worm traffic. If the ratio is too low, legitimate traffic will often be misidentified as worm-like. We want SWORD to be as sensitive as possible, so we want the ratio to be as low as possible while maintaining a low number of false positives. Similarly, the absolute number of connections required, α, should be low to avoid long detection times, but not so low as to induce false positives.

We choose the values for these parameters by evaluating a range of options during the training period and selecting those that give the minimum ratio $\frac{\alpha}{\beta}$ while maintaining a false positive rate lower than 0.5% (in case two triples have the same $\frac{\alpha}{\beta}$ value, we choose the one with a smaller α). The parameter choices evaluated are selected as follows: let history $H_i = 2^i$ for $4 \leq i \leq 10$. α and β then depend upon the history size: let $\beta_{ij} = 8 + \frac{H_i}{2^j}$ for $1 \leq j \leq 4$ (we assume any values less than 8 will be overly sensitive). The numerator $\alpha_{ijk} = \beta_{ij} - k$ for $0 \leq k < \beta_{ij}$. Using the training data, SWORD will calculate the false positive rate of $(H_i, \beta_{ij}, \alpha_{ijk}) \; \forall \; i, j, k$ as described above. After selecting the best set of values, they are further refined by linearly reducing the history size so long as doing so does not increase the false positive rate.

It might seem as though this algorithm would be effective at determining whether a given host is infected rather than just a single connection. However, this algorithm on its own is prone to false positives, and only when applied in conjunction with the causal similarity algorithm does it become effective.

2.3 Continuity Analysis Algorithm

The third worm behavior that SWORD leverages is that of **continuity**. In order for a worm to spread, it *must* make connections to remote hosts to infect them. This statement holds true both for worms that scan randomly to find addresses with hosts that are vulnerable to its attack, as well as for worms that start with a hit-list of vulnerable targets. This characteristic of initiating connections in order to spread is critical to the success of SWORD.

SWORD leverages this *continuity* behavior to differentiate between legitimate connections that may appear worm-like and actual worm connections from an infected host. In SWORD, a connection is considered worm-like if both the casual similarity identification algorithm and the destination address distribution algorithm agree that it is worm-like. However, legitimate connections will also occasionally be labeled as worm-like, causing a **connection-level false positive**, albeit at a significantly lower frequency than actual worm connections. We show through our experiments that the number of connection-level false positives is low, but nevertheless it is significant enough that we cannot report that there is a worm occurring upon

seeing a single worm-like connection.

The continuity behavior of worms tells us that an infected host will send a stream of worm connections—which will be identified as worm-like with a greater frequency than legitimate connections. This change in the number of worm-like connections detected is used to detect the presence of a worm with great accuracy. This distinction is true for both TCP and UDP worms, and is the core principle behind SWORD's detection of zero-day worms. It works for worms using virtually any scanning or infection mechanism.

The core of the continuity algorithm is a sliding window which compares the number of worm-like connections observed within the window against a threshold. When the threshold is exceeded, the network is considered to be infected. The size of the sliding window and the value of the threshold itself are clearly critical to the success of this algorithm and are obtained deterministically from a training period. During the training period, SWORD observes what is assumed to be worm-free gateway traffic for the network that is to be protected and measures the number of legitimate connections that are marked as worm-like.

The motivating force driving the threshold and window-size selection is that the threshold must be greater than the expected level of *connection-level* false positives (also called **noise**) for a given window size, while the window-size should be as short as possible to minimize detection time.

We determine a suitable threshold by observing the maximum background noise for a range of pre-selected window-sizes for each day during a training period. These maximum values should form a normal distribution from which we can extrapolate the *expected* maximum value and standard deviation (σ). Adding 3σ to the expected maximum value will yield a threshold that we can expect to be exceeded on a given day with only 0.3% probability. In other words, we can expect that legitimate traffic will exceed the threshold and cause a *detection-level* false positive about once per year.

Having established a suitable threshold for each of the different measured window sizes, we now must choose which window size will yield the highest accuracy and lowest latency for our worm detector. The ratio between the threshold for a given window size and the size of the window, which we will call γ, is key in choosing the best window size. γ can be thought of as the *rate* at which our worm detector must detect worm connections to exceed the threshold within a window. In other words, for us to detect that a domain is infected with a worm, that worm must scan with a rate greater than γ. In fact, in some cases such as the local-preference-scanning worm, the worm must scan at an even greater rate because not all of its scanning connections will cross the gateway and be visible to the monitor. We know that we want SWORD to be sensitive enough to detect worms scanning at a minimum rate of 1 scan per second, and we know that the expected number of those scans that will cross the gateway is only roughly 50% for the local-preference-scanning worm, so we must choose a window with a γ value lower than 1×0.5, or 0.5.

Any window size with a γ less than 0.5 will be suitable, so how do we choose which one? Observation shows that background noise is bursty (see Sec. 5.1), meaning that the longer the window size, the lower the value of γ. We also know that the shorter the window, the lower the threshold. A lower threshold means faster worm detection, because it will take less time for the worm to produce enough connections to exceed the threshold. It is thus advantageous to choose the shortest window with a γ value that meets our criteria.

Our window size and threshold selection process is therefore: measure the maximum observed background noise during a training period for a variety of window sizes, from this calculate a threshold for each window size, then choose the smallest window with a γ that satisfies the sensitivity requirements.

Having established a target window-size and threshold, we simply monitor the number of worm-like connections crossing the gateway. If it exceeds the threshold, we raise a worm infection alert.

3 Extended Components of SWORD

Building from the core components, SWORD can be extended for further functionality. We have already begun adding functionality such as the host-level detection and behavior-based classification described below,

but do not report our results on those components here.

3.1 Detecting Worm-Infected Hosts

Using essentially the same algorithms as those in detecting whether a protected domain is infected (Sec. 2), SWORD can be extended to detect which individual hosts are infected. The primary modification is to increase the amount of state that SWORD maintains, using host-level continuity analysis rather than aggregating the worm-like connection count at the host level. SWORD can still maintain its advantageous location at the gateway of a domain, the only real cost to this approach is increased memory requirements.

Host-level infection detection has the following advantages: (i) If we know a host is infected, then we know a domain is infected; (ii) knowing which hosts are infected can improve quarantine effectiveness; and (iii) it demonstrates that a host infection detection solution need not be rooted at individual hosts. For details regarding host-level detection, please refer to [15].

3.2 Behavior-Based Classification of Worms

Upon detecting a worm, this extended component of SWORD will attempt to report what type of worm is occurring based on its behavior with respect to causal similarity, destination visiting pattern, and continuity.

Previous work in classifying worms has been primarily based on full knowledge of what the worm actually does rather than on the observed behavior. Staniford *et al.* catalogued various scanning techniques a worm may use in [1], and Weaver *et al.* extended this to a full taxonomy of worm characteristics in [16]. Categorizing a worm within these taxonomies requires a detailed examination of a worm's operational code. This level of knowledge is expensive to acquire and requires human involvement. This component extends SWORD to classify worms based on SWORD's temporally and spatially limited observations of their behavior—even with noise from legitimate traffic.

Worms can exhibit different behavior with respect to causal similarity, destination distribution, and continuity. SWORD combines a worm's behavior from each aspect to classify the worm. For causal similarity, for example, causal connection subgraphs can be used as an abstract representation of different causal relationships of worm connections. For destination distribution, different worms may exhibit various degrees and patterns of deviation from normal patterns. For continuity, inter-arrival distribution of worm connections can vary for different worms.

For brevity, we omit the details of classifying a worm according to its behavior from this paper.

4 Methodology of Evaluating SWORD

We adopt a trace-based simulation approach to evaluating SWORD's performance. In the following, we first describe what metrics we use, and then describe the traffic utilized for the evaluation and how we run the experiments.

4.1 Metrics

The metrics employed must be able to evaluate the following: the *latency*, or the average speed with which a worm is detected, and the *accuracy* with which a worm is identified. The latency can be measured either as the time between the first host in our network getting infected and the time the worm activity is detected, or as the number of internal hosts infected before detection. We measure accuracy in terms of **false positives**: the number of times a worm alert is raised without the presence of a worm, and **false negatives**: the number of times the network is infected but *no* alert is raised.

4.2 Background Traffic

The realism of the background (or *normal*) traffic used in evaluating SWORD is important because the key feature of any worm detection tool is in differentiating between normal and worm traffic. We cannot use a live traffic feed because we must be able to run repeatable experiments, and there are no simulations that can provide realistic enough traffic to test against, so instead we use a pre-recorded trace from a real network.

The real trace in our experiments is the Auckland-IV trace [17]. It is a continuous 45 day GPS-synchronized IP header trace recorded at the University of Auckland and Auckland University of Technology. Traffic was tapped from an OC3 ATM link that connects the Universities to the service provider. The inside networks contain two /16 and several /24 prefixes and all IP addresses are anonymized in the trace. The trace includes all the TCP and UDP header information necessary for our experiments, but no payload information. We redistributed the anonymized IP addresses from the trace into a fictionalized IP range that properly reflects the topology of the network inside the Auckland border router. On any given day there were approximately 2,250 hosts making roughly 2.3 million total outbound connections and another 1 million or so incoming connections. The hosts which were active varied from day to day, and roughly 10,000 total internal hosts were active at some point in the trace.

The trace we used is the most recent trace that we can find that fully meets our needs. However, it was recorded between February and April of 2001, which is almost six years ago now. The age of the trace has both pros and cons. On one side, we are more confident that there are no worms active in this trace (no report has identified worms in it), simplifying the analysis of our results. On the flip side, the trace won't contain traffic from more modern applications. To make best use of it, we assume that it has no worm traffic present in it, and focus on checking whether SWORD can accurately detect injected simulated worm traffic (which we discuss in the following section).

4.3 Worm Traffic

We have created our own worm simulator to model the spread of the different types of worms used in our experiments. It uses a network topology that matches that of the Auckland trace—two internal /16 networks separated from the Internet by a border router, and captures worm traffic that crosses the border router to a trace file. Readers can refer to [18] for the details on how the worm simulator works.

To study SWORD's performance in detecting all kinds of different worms, we chose to vary four parameters:

- *Connection type.* A connection can be either TCP-based or UDP-based.
- *Vulnerability Percentage*. The percentage of internal hosts running the service that are vulnerable can be either high at 100% or low at 10%.
- *Scan speed.* We also controlled the speed of the worm, allowing us to study relatively high speed (100 connections/second) and low speed (1 connection/second) worms.
- *Scan type.* Using previously presented worm methodologies [1], we were able to model random-scanning, permutation-scanning, partition-scanning, sequential-scanning, local-preference-scanning, and topological-scanning worms.

Each scanning technique works as follows: random-scanning worms choose each new target address completely randomly. Local-preference-scanning worms choose randomly, but with a preference towards choosing randomly from the local subnet: they choose an address from its class B address space with a 50% probability, from its class A address space with a 25% probability, and from the Internet as a whole with a 25% probability. The topological-scanning worm starts with a list of roughly 500 addresses known to be

Table 2: Number of worm-like connections counted in legitimate traffic during training period

(a) TCP Connections

Window Size	Max Observed Value	Expected Max Value	σ
60 seconds	40	31.3	4.7
120 seconds	54	38.7	9.7
240 seconds	64	53.6	11.4
480 seconds	102	81.4	16.5

(b) UDP Connections

Window Size	Max Observed Value	Expected Max Value	σ
60 seconds	61	46.7	12.3
120 seconds	84	63.6	18.0
240 seconds	119	78.6	27.5
480 seconds	129	96.9	28.2

running the target service, though not necessarily vulnerable. Once these have been contacted, it reverts to a pure random-scanning worm. Our initial analysis found that our algorithms performed nearly identically for the first four scanning types, so we omit results from the permutation, partition, and sequential-scanning worms.

To create a worm trace, we begin with 3,000 hosts infected in the Internet and a worm connection crossing the border router and infecting one of our internal hosts. This is meant to replicate a reasonable scenario for a zero-day worm in the early stage of its development, while also reducing the time required to run our simulations. We run each simulation until 100,000 worm connections have crossed the gateway, or 1 hour has passed in the simulation, whichever comes first. Totally we run ten simulations for each combination of worm parameters, using a different random seed for every simulation.

4.4 Experiments with Merged Traffic

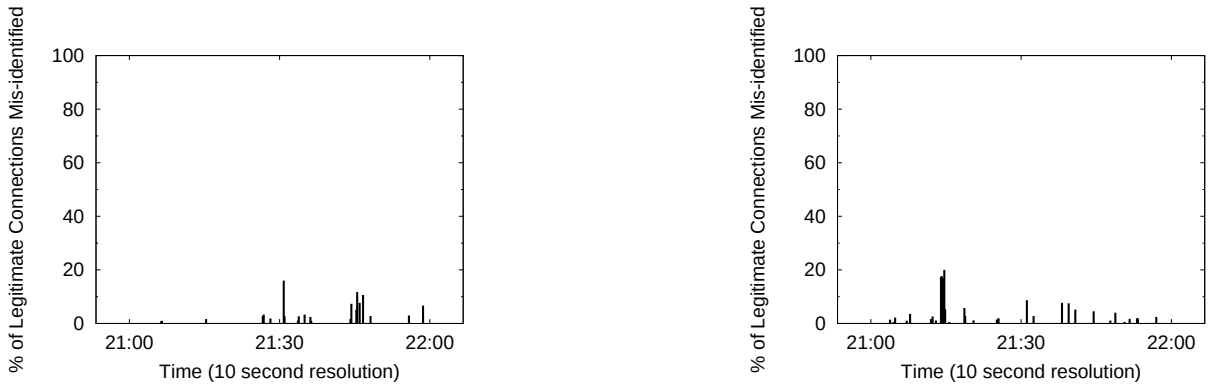
To run experiments against SWORD, we created merged traces to be the input. A merged trace consists of a single day's worth of connections just from the Auckland trace to populate the algorithms' connection graph and history, followed by connections interlaced from the desired worm trace and the next day's Auckland trace. These connections are interleaved based on the connection start-time, resulting in a single trace where we can identify which connections are worm connections and which hosts are infected at what time, allowing us to accurately measure the performance of our detection system.

We first run SWORD over a training period against one week of normal traffic to establish parameters (Sec. 2). This training period also acts as a warm-up period, populating the causal connection graph and destination address distribution histories.

We then run SWORD against merged traces for each of the 24 combinations (or scenarios) of worm parameters outlined in Sec. 4.3, We have ten simulations of each scenario with a total of 240 merged traces. We merge each simulation for a given scenario with the normal traffic from a different day of the Auckland trace with a random start time. We present our results in the next section.

5 Evaluation Results

In this section, we evaluate SWORD from several key aspects, focusing on its detection accuracy, detection latency, and computational and storage overhead.



(a) TCP (b) UDP

Figure 2: Percentage of legitimate traffic misidentified as worm-like from a one-hour period.

Table 3: Calculated 3σ-thresholds and their relation to the window size

(a) TCP Thresholds

Window Size	Threshold	γ
60 seconds	45	0.76
120 seconds	68	0.56
240 seconds	**88**	**0.37**
480 seconds	131	0.27

(b) UDP Thresholds

Window Size	Threshold	γ
60 seconds	84	1.40
120 seconds	117	0.98
240 seconds	161	0.67
480 seconds	**181**	**0.38**

5.1 Parameter Selection

As was discussed in Sec. 2, SWORD needs to observe a period of normal traffic to learn the values for several parameters. We use the first week of the trace as a training period to establish these values, and present the final values here.

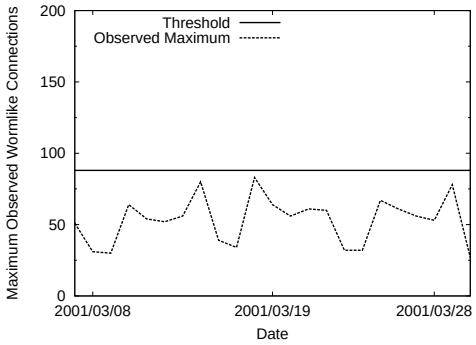
We used the method described in Sec. 2.2 to determine that for the destination address distribution analysis algorithm the most effective parameters were a history-size of 50 with 16 connections out of a span of 24 constituting a trend in the correlation.

We also obtained the optimal values for the two critical parameters in the continuity analysis algorithm: the *window-size* over which worm-like connections are tallied and the *threshold value* that the number of worm-like connections must exceed to raise the alarm. Recall that we must first observe the daily maximum observed values in the training period for a set of window sizes. For our experiments, we examined 60, 120, 240, and 480-second windows. Tables 2(a) and 2(b) show the maximum observed noise, expected maximum, and σ values derived from the training period. We then calculate the ratio between the threshold and window-size, γ, and choose the smallest window with a γ that satisfies the sensitivity requirements—in this case a γ less than 0.5. The window size selected by this process for TCP traffic is 240 seconds with a threshold of 88. The window size selected for UDP traffic is 480 seconds with a threshold of 181.

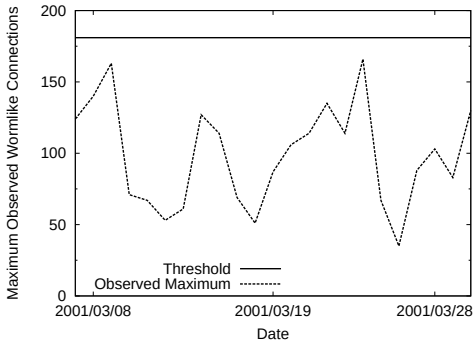
5.2 Accuracy

During the experiments outlined in Sec. 4.4, SWORD detected all tested worm types with 100% accuracy. This remarkable result holds true across a wide variety of parameters that represent a broad spectrum of both real and expected worm behavior (see Sec. 4.3 for the full list).

One of the most critical performance attributes of a worm detection system is the number of false alarms raised by the system. Our method of selecting the window-size and threshold makes false alarms very un-

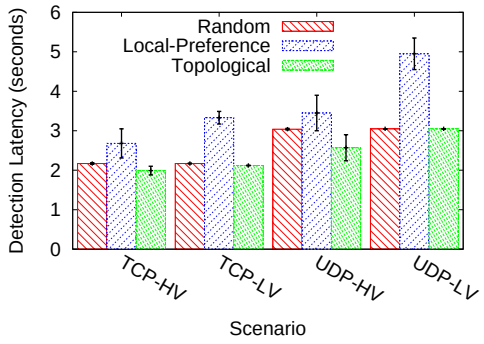


(a) TCP Connections

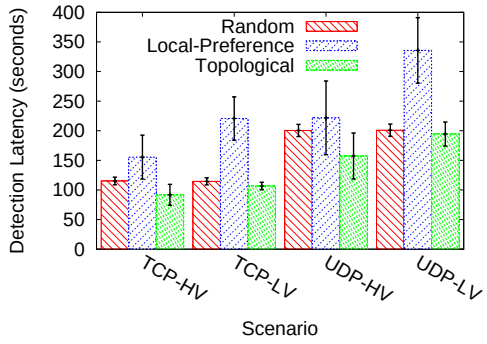


(b) UDP Connections

Figure 3: Daily maximum observed background noise compared to threshold for days other than training period.



(a) High scanning-rate worms



(b) Low scanning-rate worms

Figure 4: **Detection latencies for all worm scenarios.** *HV* and *LV* stand for high-vulnerability population and low-vulnerability population respectively. Error bars represent 95% confidence intervals.

likely, and that is verified in our experimental results. In addition to running experiments against traces with worm traffic injected, we also ran our detection algorithms against the entire month of traffic from the trace beyond what was used for training. Figures 3(a) and 3(b) show that the daily maximum of observed noise never exceeds our chosen threshold for either TCP or UDP, validating our threshold selection procedure.

5.3 Latency

SWORD must be timely in detecting worms before a worm causes serious damage. In this subsection, we first discuss SWORD's detection latency in terms of seconds, then its latency in terms of number of hosts infected at the time of detection.

5.3.1 Latency in terms of seconds

SWORD is successful at detecting both high-speed worms at 100 $scans/s$ and low-speed worms at 1 $scan/s$. The results for high-speed worms are shown in Figure 4(a). Because high-speed worms generate sufficient connection volumes to quickly overwhelm the threshold, high-speed worms of all types are detected in under 6 seconds. Note that our experiment begins at the moment when a worm infects the first internal host,

and it takes a full second for a worm-infected host to begin scanning. It then takes another second for a SWORD monitor to see enough worm-like connections to trigger a worm alarm.

Random-scanning worms (as well as permutation-scanning, partition-scanning, and sequential-scanning worms as explained in Sec. 4.3) show relatively little variation, with just a slight difference between TCP and UDP due to the different threshold. Because this type of worm is most often detected when only one host is infected, the vulnerability level of the population makes little difference.

Topological-scanning worms are detected slightly faster than random-scanning worms in the high-vulnerability cases (slightly less than 2 seconds for TCP, just under 3 seconds for UDP). This is because in the high-vulnerability scenario there are many service-running hosts, and these hosts are more likely to be targeted and then quickly infected by a topological-scanning-worm-infected host. Once another internal host is infected, worm connections will be generated at a higher rate, more quickly exceeding the threshold.

Local-preference-scanning worms are detected most slowly of the types studied. Many scans generated by such a worm do not cross the gateway where SWORD can observe them. In our experiment, since about half the worm connections do not cross the gateway, one might expect them to be detected in about twice the time as a random-scanning worm, but it is generally not that slow. For the high-vulnerability cases, the large number of vulnerable internal hosts means that a local-preference-scanning worm's scanning strategy will pay off quickly and additional internal hosts will be infected. These infected hosts increase the aggregate scanning rate sufficiently so that the local-preference-scanning worm is detected in only 3.4 seconds in the UDP high-vulnerability scenario versus 3.05 seconds for the random-scanning worm.

The results from the low-speed scenarios can be seen in Figure 4(b). The low-speed worms generate connections at about $\frac{1}{100}$ of the rate of the high-speed worms, and are thus detected much slower than the high-speed worms. For instance, the random-scanning worm is detected in an average of approximately 115 seconds for the TCP high-vulnerability scenario, versus just over 2 seconds (including 1 second latency from the time a host is infected to the time it begins to infect others, as configured in the worm simulator) in the corresponding high-speed scenario.

5.3.2 Latency in terms of number of infected hosts

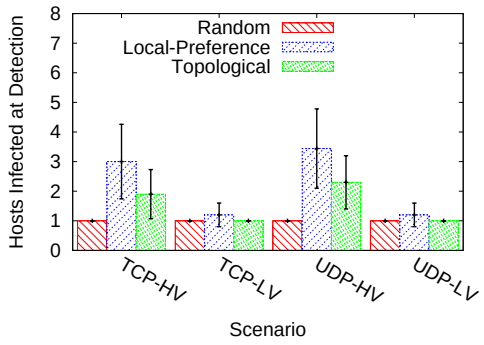
An additional and important measure of latency is the number of hosts infected when detection occurs. It does no good to detect a worm within 100 seconds if the worm has already infected your entire network. Figure 5(a) shows the number of hosts infected at detection time for the high-speed scenarios. Clearly, SWORD is effective at detecting worms before they have propagated too widely.

In all four scenarios for the random-scanning worm type, a worm is detected with only one infected host. This stems from a combination of two factors: that the random-scanning propagation algorithm is ineffective at finding additional hosts to infect, and that nearly all of the scans from an internal host will cross the gateway and be visible to the monitor.

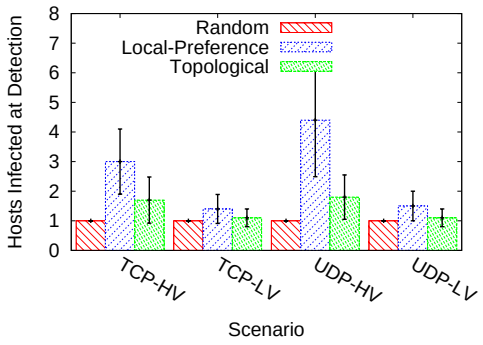
The topological-scanning worm shows a different story. In the high-vulnerability scenario, an average of over 2 hosts were infected. Every worm instance starts with a hit-list of service running neighbors, and is therefore likely to infect additional hosts early in its propagation while exploring this list. In the low-vulnerability scenario, without enough vulnerable internal hosts to be hit by worms, the effect is not visible.

The local-preference-scanning worm shows the greatest number of internal hosts infected at detection time. Again there is a more significant difference in the high-vulnerability scenarios. This is because, when there are many vulnerable hosts in the network, the local-preference-scanning strategy is very effective at finding additional hosts to infect. When this is coupled with the fact that a smaller number of the scanning connections cross the gateway and are visible to the monitor, causing a higher detection latency, the local-preference-scanning worm becomes the most effective strategy at infecting hosts.

One important result, with regards to the latency in terms of number of infected hosts, is that the low-speed results (Figure 5(b)) are very similar to the high-speed results. Regardless of the scanning rate, both



(a) High scanning-rate worms



(b) Low scanning-rate worms

Figure 5: **Number of internal hosts infected at worm detection time for all worm scenarios.** *HV* and *LV* stand for high-vulnerability population and low-vulnerability population respectively. Error bars represent 95% confidence intervals.

high-speed and low-speed worms can only be detected after generating more worm connections than a specific threshold value. In cases where a high-speed and low-speed worm make about the same number of connections, we can expect them to infect essentially the same number of hosts. The numbers do not match perfectly, however. With a slower connection rate, normal traffic is more likely to mask the worm traffic and interfere with correctly identifying connections as worm-like.

5.4 SWORD System Overhead

One potential area of concern is the system overhead required to run SWORD. However, this is not a significant problem. Our implementation is written in Java, with low enough processor and memory requirements that a dedicated PC is more than sufficient to monitor a link of the size we have studied here.

An analysis of the memory requirements illustrates the low requirements. We store only approximately 120 bytes of information for each connection stored in the causal connection graph. The graph itself is limited to no more than 20,000 connections and therefore requires less than 3 MB of memory. Regarding the connection histories for the destination address distribution algorithm, we store only 50 recent destination addresses per internal host, requiring only $\sim$2MB of storage for Auckland's domain of $\sim$10,000 hosts.

In our offline evaluation, running on a 1.73 GHz Pentium M Laptop with 512 MB of RAM, the causal similarity algorithm spent $\sim$1 millisecond per connection and the destination address distribution algorithm required only $\sim$0.001 milliseconds per connection. An entire 24 hour trace including over 3 million connections was processed in under an hour without exhausting the available memory.

6 Limitations of SWORD

The major goal of this paper is to show that it is feasible and promising to detect Internet worms based on their essential behaviors. We acknowledge that the results thus far are only one step towards accurate and fast behavior-based worm detection. Furthermore, as worm authors create smarter worms, and new legitimate traffic types show worm-like behaviors, limitations of SWORD may become an issue. Additionally, the domain-specific training period may be viewed by some as a significant obstacle to deployment.

6.1 Slow-moving or Smart Worms

One mechanism for avoiding detection that worm authors may employ is slowing their propagation down. We have shown that SWORD is capable of catching worms with scan rates of 1 scan/second which is slower than existing worms. The average scan rate for the SQLSlammer worm was $4,000$ scans/second [19] and the estimated scan rate for the Code-Red worm was 6 scans/second [20]. However, there is in fact a lower bound on worm scanning speed beneath which detection may not occur (we omit our analysis on the lower bound in this paper). The advantage here is that, if we can force worms to propagate very slowly, we have, in a way, already succeeded: we will not only dampen the damage from worms, but can also now focus on solutions for slow worms as opposed to (relatively) high-speed worms.

A smart worm may attempt to go undetected by disguising its behavior to avoid having its connections labeled as worm-like. Since SWORD does not inspect packet payload, we know it is resilient against polymorphic worms, but a worm may still try to vary its behavior. It might try to mimic the behavior of legitimate application or try to introduce polymorphic behavior, such as by becoming a multi-vector worm [1]. We have found SWORD is resilient against some such situations (*e.g.*, while a worm tries to vary itself, a subset of its connections will still look similar), but we still need to systematically investigate and improve SWORD's resiliency against worms.

6.2 Worm-like Legitimate Traffic

It is important to closely watch new legitimate applications that may be worm-like and continue to discover ways to distinguish them from worm traffic. For instance, peer-to-peer traffic tends to be self-similar and reaches many destinations, both of which are properties that SWORD looks for. However, current types of peer-to-peer traffic are different from worm traffic in important ways as well. Peer-to-peer systems generally employ existing connections for sustained communications rather than establishing new, short-lived connections for each message. Because SWORD works at the connection level, it can detect the difference between these two types of connections. Nonetheless, there may be new applications that generate completely worm-like traffic; in this case, SWORD will likely misclassify the connections from this application as worm-like, unless SWORD introduces a whitelisting mechanism.

6.3 Training Against Normal Traffic

As we touched upon in Sec. 2, SWORD requires a training period against normal traffic before being able to accurately distinguish between legitimate connections and worm connections. Without knowing the normal behavior of the hosts within a domain, we cannot know what to consider as *ab*normal behavior. This limitation is not unique to SWORD; any behavior-based detection system, which must know what normal behavior is, will also have this limitation.

Fortunately, this limitation is not a large obstacle to deploying SWORD on a domain. The training can be done in a very straightforward manner. The values for the threshold and window-size of the continuity algorithm can be found by calculating the expected maximum value and standard deviation, as described in Sec. 2.3. The history size and trending requirements for the destination address distribution algorithm are similarly easy to compute (Sec. 2.2.

7 Positioning Our Research Among Others

Research into detecting self-propagating worms can be roughly categorized into two primary classes: content-based detection and behavior-based detection. Both of these have already been extensively researched, but SWORD approaches the problem from a new angle and is different from previous works in important ways.

7.1 Related Work

Content-based detection relies on examining the payload of worm communications to look for the most prevalent substrings or suspicious flows, in order to generate a *signature* to identify a specific worm. In doing so, researchers have attempted different ways to isolate worm traffic, including: correlating host-based detection with network monitoring ([21]); monitoring traffic toward unused IP addresses ([20, 22]) or a honeypot system ([9]); setting up a center to collect suspicious traffic ([1, 23]); or simply inspecting *all* traffic (such as Earlybird [2], Autograph [3], WEW [11], PAYL [24], and an FPGA-based system [4]). However, regardless of the mechanism for isolating traffic, all content-based detection schemes suffer from the same basic flaw: *They are vulnerable to polymorphic worms which can change their payload as they spread.* More recent advances such as [25, 26] attempt to address this issue, but research shows that such schemes will not be sufficient for detecting all polymorphic worms [5].

Behavior-based detection approaches are more closely related to ours. *However, existing solutions mostly rely on non-essential behaviors that worms can circumvent.* Work focusing on network-level behaviors includes monitoring unexpected traffic toward unused IP addresses [27, 20, 28, 29, 6, 7] or honeypots [8, 9, 30, 31, 32, 33], watching for error messages (*e.g.,* ICMP Unreachable [10] or TCP Reset [11]), or tracking the lack of DNS queries [12]. While well suited for their stated purpose, these solutions look for behaviors smarter worms can avoid: a worm can choose to only contact hosts in actual use, not unused IP addresses or honeypots; a worm can try to confuse the defense by deliberately sending traffic that carries characteristics of legitimate traffic; and a worm can avoid error messages as well as force DNS queries.

Existing host-level behavior-based detection solutions attempt to identify more essential behaviors of worms. *Unfortunately, they are still unable to accurately capture behaviors essential and unique to worms.* Research in [13] and [14], for example, tries to correlate outgoing connections toward a port with a preceding incoming connection on that port. This correlation fails to detect multi-vector worms; and worse, certain legitimate applications (web proxies [34, 35], SMTP servers, DNS servers, or Gnutella [36]) can also have similar inputs and outputs. The work in [14] also looks for payload similarity, and is thus vulnerable to polymorphic worms. Moreover, while needing to watch host behavior closely, *host-level behavior monitoring is often impractical to deploy.* It has to watch *all* internal traffic of a network (as in [13] and [14]); or, it must be deployed at end hosts (such as [37] where system calls made on multiple hosts are compared for similarity) yet it is unreasonable to expect *all* end-hosts to install a system no matter how effective it is.

In summary, there is yet to be a reliable and fast worm detection system that is behavior-based, not content-based; that captures essential worm behaviors, not superficial symptoms; and that is easily deployable, not cumbersome to adopt. SWORD achieves this goal.

7.2 Advancing State-of-the-art

Below we discuss more specifically how each of SWORD's algorithms advances the state-of-the-art in self-propagating worm detection, along with some open issues we still face.

Similarity Comparison: Existing solutions often compare traffic payload for similarity (*e.g.,* [14, 38, 39]), which is vulnerable to polymorphic worms. Research in [37] compared the similarity between system calls on different hosts, but is not deployable at the gateway of a domain. In identifying connections for similarity comparison, existing solutions can trace their heritage to GrIDS [40], which introduced the idea of using communication graphs for worm detection. Toth and Kruegel advanced the line in [39] by identifying TCP connections with potential causal relationships; but, its similarity analysis is based on payload, relies on connections to unused IP addresses, and is for deployment at end hosts. Ellis *et al.* formalized some ideas in GrIDS using a descendant relation between infected hosts [14], but this concept is not about the relationship between *connections* and won't help identify connections for similarity comparison. A more recent work for identifying worm connections is the random moonwalk algorithm in [41], but it is mainly

an after-the-fact forensic analysis tool.

Our similarity comparison is not payload-based, but uses certain connection attributes instead. One open issue here is to identify more accurately and broadly the metrics for similarity comparison, and ensure these metrics are robust against smart worms. For example, not only can we compare the similarity between individual connections, but also that between several sets of connections that may display similar propagation patterns. In maintaining the causal connection graph, we also need to consider how to cache connections, how to deduce causal relationships even though a monitor cannot see *all* connections, and how to perform well for domains at different scales.

Destination Address Distribution: Destination address distribution has been studied in the context of traffic management and caching [42, 43], as well as for detecting traffic anomalies [44]. It has also been used to analyze suspicious payload [2], throttle suspicious traffic [45, 46], or detect "super-spreaders" with a very large number of destinations [47]. Sekar *et al.* [46] further shows the promise of monitoring the rate of unique destinations a host contacts for worm detection.

We have shown in this paper a ranking-based destination address distribution analysis algorithm. It is likely, however, that other approaches may work even better (faster and/or more accurate) in differentiating the destination distribution with and without the presence of a worm—especially worms of various speeds—and thus detect worm-like connections.

If a worm wants to disguise itself, it must ensure that, when a host is infected and sends out worm connections, the distribution of the destination addresses of *all* connections from the host, legitimate or not, still follow the norm. Most of the time, infection attempts must go to a small number of fixed targets (and such connections, when too numerous, will also look suspicious). Nonetheless, how effective this constraint can slow down the worm warrants further study.

Worm Continuity: Moore *et al.* showed that random-scanning worms such as Code-Red will infect fewer than 1% of vulnerable hosts if we can stop them before they send roughly $12,000$ connections [48]. But, to our knowledge, no one has used this feature to lower the false positives in detecting worms.

In contrast, our sliding-window-based mechanism is shown to be effective in having low false positives as well as low false negatives. One open issue here is that every domain may need a specific configuration for the sliding window size and threshold values. Factors such as the size of a domain, time of the day, or the type of popular applications running inside a domain, may all affect the configuration. As all these factors evolve over time, so will the most suitable configuration. Although we could learn these values by running SWORD over a period known to be worm-free, it is subject to becoming obsolete. To solve the problem, an automatic training process that helps learn the best configuration over time would be necessary.

8 Conclusions

Accurate detection of Internet worms in their early stages remains an unsolved problem. One could scan Internet traffic for worm signatures or suspicious byte patterns, but worm signatures are often useless for unknown worms that exploit new software vulnerabilities, and suspicious byte patters are problematic because worms can carry virtually arbitrary payloads. Processing and analyzing traffic payloads is also expensive. These limitations have motivated investigations into the behaviors that self-propagating worms may display. However, for a behavior-based approach to succeed, researchers must now identify the precise or essential behaviors that worms will display.

Toward this end, we presented the SWORD worm-detection framework, together with its algorithms that monitor worm-related behaviors of the connections that cross the border of a domain: The causal similarity identification algorithm detects whether a connection is similar to earlier connections that may have caused it, the destination address distribution algorithm captures connections deviating from the regular destination visiting pattern, and the continuity analysis algorithm discovers whether enough worm-like connections have

occurred to trigger a worm alert. These algorithms do not inspect the payload of traffic, thus are resilient to content polymorphism or encryption of any given worm.

By exploiting invariant behavior common across different worms, our research is an important step toward effectively detecting zero-day self-propagating worms—even if those worms change or disguise their payload. This research can be further extended to detect which hosts inside a domain are infected by a worm as well as classify newly detected worms according to their behavior, *i.e.*, behavior-based worm classification. As demonstrated through our comprehensive evaluation, SWORD is fast and accurate in detecting self-propagating worms of different speeds. No matter the worm's scanning method or whether it uses TCP or UDP, it can be detected in its early stage of propagation.

References

[1] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in your spare time," in *Proc. USENIX Security Symposium*, August 2002.

[2] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proc. Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[3] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. USENIX Security Symposium*, August 2004, pp. 271–286.

[4] B. Madhusudan and J. W. Lockwood, "A hardware-accelerated system for real-time worm detection," *IEEE Micro*, vol. 25, no. 1, pp. 60–69, 2005.

[5] J. R. Crandall, Z. Su, S. F. Wu, and F. T. Chong, "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits," in *Proc. Conference on Computer and Communications Security*, 2005.

[6] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of Internet sinks for network abuse monitoring," in *Proc. Symposium on Recent Advances in Intrusion Detection*, September 2004.

[7] M. A. Rajab, F. Monrose, and A. Terzis, "On the effectiveness of distributed worm monitoring," in *Proc. USENIX Security Symposium*, 2005.

[8] N. Provos, "A virtual honeypot framework," in *Proc. USENIX Security Symposium*, 2004, pp. 1–14.

[9] C. Kreibich and J. Crowcroft, "Honeycomb: Creating intrusion detection signatures using honeypots," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 51–56, 2004.

[10] V. Berk, G. Bakos, and R. Morris, "Designing a framework for active worm detection on global networks," in *IEEE International Workshop on Information Assurance*, 2003, pp. 13–24.

[11] S. Chen and S. Ranka, "Detecting Internet worms at early stage," *IEEE J. Selected Areas in Communications*, vol. 23, no. 10, 2005.

[12] D. Whyte, E. Kranakis, and P. C. van Oorschot, "ARP-based detection of scanning worms in an enterprise network," in *Proc. Annual Computer Security Applications Conference*, December 2005.

[13] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley, "Worm detection, early warning and response based on local victim information," in *Proc. Computer Security Applications Conference*, 2004.

[14] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia, "A behavioral approach to worm detection," in *Proc. Workshop on Rapid Malcode*, 2004.

[15] S. Stafford, J. Li, and T. Ehrenkranz, "On the performance of SWORD in detecting zero-day-worm-infected hosts," in *Proc. Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, vol. 38, no. 3, July 2006, pp. 559 – 566, track 8 (Traffic Characterization).

[16] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proc. Workshop on Rapid Malcode*. New York, NY, USA: ACM Press, 2003, pp. 11–18.

[17] WAND Network Research Group, "WAND WITS: Auckland-IV trace data," http://wand.cs.waikato.ac.nz/wand/wits/auck/4/, April 2001.

[18] J. Li, S. Stafford, T. Ehrenkranz, and P. Knickerbocker, "GLOWS: A high-fidelity worm simulator," University of Oregon, Tech. Rep. CIS-TR-2006-11, 2006.

[19] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The spread of the Sapphire/Slammer SQL worm," http://www.caida.org/analysis/security/sapphire/, CAIDA, Tech. Rep., 2003.

[20] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for Internet worms," in *Proc. Conference on Computer and Communications Security*, October 2003, pp. 190–199.

[21] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: A basis for building self-protecting servers," in *Proc. Conference on Computer and Communications Security*, 2005.

[22] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet motion sensor: A distributed blackhole monitoring system," in *Proc. Network and Distributed System Security Symposium*, 2005.

[23] "Internet storm center," http://isc.sans.org.

[24] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proc. Symposium on Recent Advances in Intrusion Detection*, September 2004.

[25] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. IEEE Symposium on Security and Privacy*, 2005.

[26] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in *Proc. IEEE Symposium on Security and Privacy*, 2006.

[27] D. Moore, "Network telescopes: Observing small or distant security events," Presentation at Usenix Security conference., August 2002.

[28] J. Wu, S. Vangala, L. Gao, and K. Kwiat, "An effective architecture and algorithm for detecting worms with various scan techniques," in *Proc. Network and Distributed System Security Symposium*, 2004.

[29] V. Yegneswaran, P. Barford, and S. Jha, "Global intrusion detection in the DOMINO overlay system," in *Proc. Network and Distributed System Security Symposium*, 2004.

[30] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen, "HoneyStat: Local worm detection using honeypots," in *Proc. Symposium on Recent Advances in Intrusion Detection*, Sophia Antipolis, France, September 2004.

[31] X. Jiang and D. Xu, "Collapsar: A VM-based architecture for network attack detention center," in *Proc. USENIX Security Symposium*, August 2004, pp. 15–28.

[32] J. Levine, R. LaBella, H. Owen, D. Contis, and B. Culver, "The use of honeynets to detect exploited systems across large enterprise networks," in *Proceedings of the Information Assurance Workshop*, June 2003, pp. 92–99.

[33] "NetBait, Incorporated," http://www.netbaitinc.com.

[34] "Squid web proxy cache," http://www.squid-cache.org.

[35] The Apache Software Foundation, "mod_proxy - Apache HTTP server," http://httpd.apache.org/docs/2.2/mod/mod_proxy.html.

[36] "Gnutella," http://www.gnutella.com/.

[37] D. J. Malan and M. D. Smith, "Host-based detection of worms through peer-to-peer cooperation," in *Proc. Workshop on Rapid Malcode*, 2005.

[38] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Proc. Symposium on Recent Advances in Intrusion Detection*, 2005.

[39] T. Toth and C. Kruegel, "Connection-history based anomaly detection," in *Proc. IEEE Workshop on Information Assurance and Security*, June 2002, pp. 30–25.

[40] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "GrIDS: A graph based intrusion detection system for large networks," in *Proc. National Information Systems Security Conference*, 1996.

[41] Y. Xie, V. Sekar, D. A. Maltz, M. K. Reiter, and H. Zhang, "Worm origin identification using random moonwalks," in *Proc. IEEE Symposium on Security and Privacy*, 2005, pp. 242–256.

[42] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proc. ACM SIGCOMM*, 2003.

[43] R. Jain, "Characteristics of destination address locality in computer networks: A comparison of caching schemes," *Computer Networks and ISDN Systems*, vol. 18, no. 4, pp. 243–254, May 1990.

[44] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proc. ACM SIGCOMM*, Philadelphia, PA, August 2005.

[45] M. Williamson, "Throttling viruses: Restricting propagation to defeat malicious mobile code," in *Proc. Annual Computer Security Applications Conference*, 2002.

[46] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "A multi-resolution approach for worm detection and containment," in *Proc. International Conference on Dependable Systems and Networks*, 2006.

[47] S. Venkataraman, D. Song, P. B. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proc. Network and Distributed System Security Symposium*, February 2005.

[48] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Proc. IEEE INFOCOM*, 2003.