A Monitoring System for Detecting Repeated Packets with Applications to Computer Worms[‡]

P.C. van Oorschot[*] J.-M. Robert[†] M. Vargas Martin[§]

Abstract

We present a monitoring system which detects repeated packets in network traffic, and has applications including detecting computer worms. It uses Bloom filters with counters. The system analyzes traffic in routers of a network. Our preliminary evaluation of the system involved traffic from our internal lab and a well known historical data set. After appropriate configuration, no false alarms are obtained under these data sets and we expect low false alarm rates are possible in many network environments. We also conduct simulations using real Internet Service Provider topologies with realistic link delays and simulated traffic. These simulations confirm that this approach can detect worms at early stages of propagation. We believe our approach, with minor adaptations, is of independent interest for use in a number of network applications which benefit from detecting repeated packets, beyond detecting worm propagation. These include detecting network anomalies such as dangerous traffic fluctuations, abusive use of certain services, and some distributed denial-of-service attacks.

Keywords: computer worms, anomaly detection, network security, intrusion detection, traffic monitoring.

1 Introduction

Worms, unlike other malicious code such as viruses and trojans, are capable of self-propagating [60]. The constant threat of worms to the network infrastructure has become one of the major concerns of network equipment designers. In this paper we propose a router-based monitoring system using Bloom filters with counters (BFWC) [17] to automatically detect repeated packets. This capability is adapted to detect worms which transmit significant numbers of packets within short time periods. The idea is to use BFWCs to count the number of times each packet has been forwarded locally. If the count for one packet exceeds a specified threshold within a given fixed period of time, the BFWC triggers an alarm. (Note that Bloom filters can indeed be implemented in hardware without considerable performance degradation in routers [13, 50].) In addition, the detection mechanism provides sufficient information – e.g., identifying a port number or packet signature – to allow some possible reactions. (In contrast, e.g., detecting only that *"there is a UDP flood attack in progress"* would not suffice to allow an appropriate reaction.) While the monitoring system can also detect abnormal behaviour in the network (by identifying the port number responsible for sending significant amounts of repeated packets), in this paper we focus our attention largely on worm detection.

A large class of worms can be characterized by self-propagating code that selects victims by generating random IP addresses, yielding a propagation pattern giving the impression of a "fan." For example, the Slammer worm [36] arrives at the victim network in one malicious UDP packet; once the infection is completed an infected host sends the malicious packet to as many random IP addresses as possible. Similar behaviour can also be observed in multi-TCP-packet worms such as Code Red [63]. This paper addresses the problem of how to efficiently detect such a propagation behaviour locally, and how to locate the monitoring mechanisms in the network infrastructure to use as few monitors as possible.

OUR CONTRIBUTIONS. Our main contributions are: (1) a monitoring system based on stateful analysis of network traffic in routers, using an extension of Bloom filters [4]; (2) a validation of the system using limited data sets of network traffic, which suggests that the number of false alarms may be reduced by filtering out a small number of common pre-identified non-worm repeated packets, such that they are not processed by the BFWC; (3) addressing the problem of where to locate the monitoring mechanisms in the network infrastructure to use as few monitors as possible, which eases deployment and reduces the impact on the network; this may be of independent interest. We furthermore (4) use simulations to show that deploying BFWCs in routers of an approximate minimum vertex cover may be effective at detecting worms in the early stages of propagation.

[‡] 30 Nov. 2005. *International J. Information Security* (to appear). A preliminary version of this paper was filed as Technical Report TR-04-02 (June 2004), School of Computer Science, Carleton University, Canada. Contact author: miguel.vargasmartin@uoit.ca.

[*] School of Computer Science, Carleton University, Canada.

[†] Research & Innovation Centre – Security Group, Alcatel Canada.

[§] University of Ontario Institute of Technology, Canada.

While our results, as presented, involve algorithms operating directly on packets – which is the context in which we have carried out this research – in practice our (packet matching) work may be of greater interest if one assumes a device carries out these algorithms after full IP and TCP stream reassembly by independent means (see [15]).

LIMITATIONS. As our monitoring system is designed to detect worm packets with identical payloads, it has limitations with respect to worms with varying payloads. BFWCs are not effective against polymorphic worms (i.e., those changing their representation on each new infection - e.g., see Nachenberg [37]), or the simpler subset of encrypted worms (which encrypt the bulk of their payloads using a different key per infection). Even a worm which changes a single byte (e.g., the Witty worm [47], which includes random padding and variable destination port in each packet) can deceive BFWCs. To deal with polymorphic worms that only mutate part of their payload, the monitoring system may use a fingerprinting technique which considers only unchanged portions of the payload such as the Rabin fingerprinting technique [45], which has been previously used in intrusion detection (see [48, 49, 27]). Identifying these unchanged portions is beyond the scope of this paper, and merits attention as an open problem. However, our approach in its present form is of use as a first step, and indeed is capable of detecting "popular" recent worms, almost all of which are non-polymorphic. (For further discussion of polymorphic worms and "Polygraph" [39], see §2.) While we believe that detecting polymorphic worms is an important problem, in this paper we focus on how BFWCs can be deployed efficiently for detecting non-polymorphic worms. Also, an attacker could take evasive action through worm instances with packets intentionally fragmented to cause recognition ambiguities [44, 23], thereby escaping our BFWC detection technique. Nonetheless, our results remain interesting in that they may lead towards an effective stream matching algorithm.

OUTLINE. §2 briefly overviews related work. In §3 we present the principles of Bloom filters, leading to the BFWC discussion of §4. In §5 we consider the deployment of BFWCs in routers of a vertex cover. §6 addresses some important implementation issues. In §7 we evaluate the performance of the BFWC under selected traffic data sets. §8 reports on the performance of the BFWC in a simulated network. §9 gives concluding remarks. The Appendix considers the selection of a threshold parameter used.

2 Related Work

In this section we present a partial summary of strategies against worm propagation. This includes intrusion detection systems (IDS) based on the principle that worms (as well as other kinds of denial-of-service attacks) perform *many* similar actions within a *short* period of time. Our approach relies on this same principle.

The CERIAS Group [8] proposes a network anomaly-based IDS based on a common propagation technique observed in worms: probing many random IP-addresses in a "short" period of time. It is assumed that legitimate traffic does not exhibit such probing behaviour. The idea is to have a sensor constantly monitoring the outbound traffic of a host (or a set of hosts). This sensor triggers an alert if the number of packets of a particular [src_addr, dst_port] pair exceeds a predefined threshold t. The counter for each [src_addr, dst_port] pair is set to zero every Q units of time. The authors indicate that the implementation of this approach is in progress; details are not provided. A similar approach has been used to detect malware scanning hosts remotely [26].

Williamson [62] pursues an approach based on the same principle. He presents a network anomaly-based intrusion detection and response system against viruses based on the belief that a virus would open many *new connections*, i.e., connections that have not been seen recently. This behaviour is also characteristic of worms. The idea is to maintain a list of n (different) destination IP addresses (the *working set*) representing the n most recently open connections. Any connection to an IP address not found in the working set is considered a new connection. Before a connection is processed the system checks for newness; if the connection is not new, it is processed normally. Otherwise, the connection is put into a delay queue, whose elements are popped off in FIFO order every Q units of time. An implementation of this approach is presented in [54].

Based on the same principle, Weaver et al. [61] present a worm containment system which throttles connections based on approximate source-destination pair reputation. In this approach, a pair gains/loses good reputation every time a connection attempt between the source and destination succeeds/fails. Based on the same principle, Venkataraman et al. [57] propose a system for detecting *superspreaders* (i.e., hosts which contact at least a given number of other hosts within a *short* period of time) at low memory and processing expense. The authors present two basic probabilistic approaches based on hash-based sampling; one of them consists of sampling distinct source-destination pairs and counting the number of times each pair is repeated within a short period of time. The other algorithm performs a two-level filtering where the first level consists of filtering out (i.e., not sampling) sources which contact a small number of distinct destinations, whereas in the second level each distinct source-destination pair resulting from the first-level filtering is counted.

Following the same principle, Toth et al. [53] propose a network anomaly-based IDS against worms, involving a firewall on each host, a traffic monitor per local network, and a central analyzer. Each traffic monitor listens to all packets transmitted in that network and collects tuples of the form: [*timestamp, src_ addr, src_port, dst_ addr, dst_port, z_ bytes_ of_payload*]. These tuples are stored temporarily in a connection-history table, which is renewed every given fixed period of time. The central analyzer periodically collects lists of tuples from the traffic monitors and infers attacks based on measured values which include the number of times a tuple is repeated, the number of connection attempts to non-existing hosts, and the number of connection attempts to non-existing services. Upon detection of a worm, the central analyzer automatically broadcasts "appropriate changes" to the firewalls' policies to all the hosts in the network. Given the number of issues involved overall (e.g., overhead of reassembling packets, storage capacity required to keep lists of tuples, time to retain the lists, etc.), the impact on network performance is not clear; no performance measurements or implementation is reported.

Based on the same principle, Chen et al. [9] present a worm propagation detection system employing packet matching. An enhanced router maintains two lists of counters with each counter associated with one port. One list is for incoming connections; the other is for outgoing connections. The system matches port counters in both lists and then monitors the number of packets to distinct destination addresses for each of these ports. The system relies on Williamson's observation [62] that users (as opposed to worms propagating to random IP addresses) usually make connections to a small set of addresses. Every predetermined period of time enhanced routers check the number of connections to different addresses. Worm activity is inferred if this number is "significantly" larger than the long-term average. The system is tested with simulations using different router-level topologies. The authors test the system under different scenarios including variations on scanning strategies, network topologies, and parameters of the detection system. No experimentation using real traffic or real network topologies is reported. Unless routers are vulnerable to worm infections, matching incoming and outgoing port counters is not effective in routers which have no LANs attached.

Singh et al. [48, 49] present the "EarlyBird System" for worm detection based on packet payload analysis. Their approach consists of detecting the most popular flows (flows are identified with packet payload hashes), counting the number of source and destination addresses for these flows, and counting the number of connection attempts of these flows to unused portions of IP addresses. The system can detect different malicious packets in real networks. As for almost all systems to date (including that proposed in the present paper), worms employing polymorphism or end-to-end encryption evade detection.

Wang et al. [58] propose a network IDS based on n-gram analysis (an *n-gram* is a consecutive sequence of n characters or symbols in a text document). The idea is to train the IDS with traffic with no malicious packets into categories according to destination port and size. Then an n-gram frequency distribution is computed for each category (they focus on 1-gram analysis). After the training period the IDS compares statistically the n-gram frequency distribution of each arriving packet with the n-gram frequency distribution of the corresponding category (according to destination port and packet size) using a variant of the well-known Mahalanobis distance. If the result of this comparison is above a threshold the packet is considered malicious. Matrawy et al. [32] propose a network denial-of-service mitigation system based on (p, n)-gram analysis, where a (p, n)-gram is an n-gram at byte position p. The idea is to categorize traffic according to their (p, n)-gram characteristics hoping that disruptive traffic can effectively be separated from non-disruptive traffic. In terms of network overhead, the number of tasks involved in many of these systems [49, 58, 32] raises performance issues warranting further practical validation regarding applicability in core routers.[1]

Newsome et al. [39] present "Polygraph," a system for automatic generation of signatures for polymorphic worms. This system relies on the fact that a number of polymorphic worms need to contain an unchanged portion to be able to exploit the targeted vulnerability (this may not the case for Santy worms, for example, which exploit generic flaws of web applications [29]). The authors present a group of signature generation algorithms which employ the naïve Bayes classifier, and methods for finding signatures by determining the presence of a set of tokens within a sequence of bytes. "Polygraph" relies on appropriate classification of worm and non-worm flows.

Valdes et al. [55] present a visualization technique of network activity. This technique allows visual detection of vertical and horizontal scanning through graphical combinations of source and destination IP addresses and ports. The authors indicate that appropriate entropy analysis may enable this technique for early detection of malicious traffic; see also Onut et al. [41].

Regarding the deployment of sensors in core networks, Park et al. [43] show experimentally that a vertex cover of the Internet can be constructed with approximately 20% of the total number of nodes. Their results rely on the characteristics of the network underlying the Internet [16]. Park et al. use randomly generated power-law networks to test the performance of a well-known approximation algorithm for finding a vertex cover. They deploy filtering mechanisms in the vertex cover to detect spoofed addresses. We use the vertex cover results of Park et al. to deploy our

[1] The EarlyBird system has recently been progressed commercially and advanced (including hardware implementation) by NetSift Inc., which was acquired by Cisco in June 2005.

monitoring mechanism for detecting repeated packets.

3 Bloom Filters

In this section we review Bloom filters [4]. A Bloom filter is a hash based method for testing membership of a series of items in a large given set of items, with allowable errors. Bloom filters have been used in a number of different contexts since they were introduced [46, 14, 13, 21, 28, 7, 56]. For example, Snoeren et al. [50] present a traceback system capable of tracing packets delivered by the network in the recent past using Bloom filters. They show that the system is effective (up to a certain number of false positives, inherent to Bloom filters), efficient (requiring storage proportional to 0.5% of the link capacity), and, perhaps most importantly, they show that it can be efficiently implemented in hardware. Using Bloom filters each router of the network keeps a record of the packets that it sends forward over a certain fixed time window. When a packet needs to be traced back, the system constructs an attack graph indicating the path(s) from which the packet was forwarded along the network. This graph is constructed by checking which routers forwarded this packet.

The idea of Bloom filters is to insert a given set of items into a bit-array, or *Bloom-table*, as follows. Each item is hashed by k independent hash functions h_i, for $i = 1, ..., k$. The resulting value of each of these hash functions is interpreted as an index pointing to an entry of the Bloom-table. Then the corresponding entry of the Bloom-table is set to 1 (initially, each entry is set to 0).

To test membership of an item p, a similar procedure is followed: p is hashed and if $h_i(p) = 1$, for $i = 1, ..., k$, then it is inferred that p is already in the Bloom-table. See Figure 1.

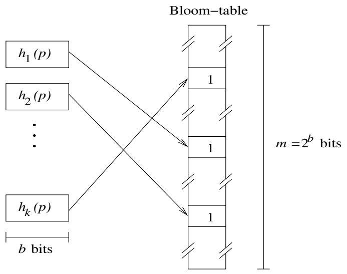


Figure 1: Illustration of a Bloom filter.

One way to assess the efficiency of a Bloom filter is by measuring the accuracy of its membership test. If a membership test is negative, we can be certain that it is correct, i.e., the item p has not been inserted into the Bloom-table. However, if the test is positive, there exists the possibility that the result is incorrect, i.e., arises due to several items $p' \neq p$ collectively causing cells $h_i(p)$ being set to 1. We define a false positive as follows.

Definition 1 (false positive). Insert n items $p_1, ..., p_n$ into a Bloom filter. If a membership test for an item $p \neq p_i$ (for $i = 1, ..., n$) succeeds, then we call this event a *false positive.*

Note that the notion of falseness here has to do with the fact that the membership test succeeds as a result of a hashing collision rather than a repeated item.

The number of false positives depends on the number of hash functions k, the size of the Bloom-table m, and the total number of items to be inserted, n. It is desirable to design a Bloom filter which simultaneously minimizes the number of false positives, the size of the Bloom-table, and the number of hash functions. The number of false positives can be approximated as follows. When an item is inserted into the Bloom-table, the probability that any particular entry is set to 1 by one hash function is $1/m$, and the probability that any particular entry is unchanged is $1 - 1/m$. Hence, after inserting n random items using k independent hash functions, the probability that a particular entry is still 0 is $(1 - \frac{1}{m})^{nk}$. Thus, when we test for membership, the probability f of a false positive equals the probability that the k hash functions all point to entries with value 1, i.e., $f = (1 - (1 - \frac{1}{m})^{nk})^k$. For m large, f can be approximated by $f \approx (1 - e^{-\frac{nk}{m}})^k$ which is minimized for

$$k = (\ln 2)\,\frac{m}{n}. \qquad (1)$$

For m large, we obtain a false positive probability of $f \approx 0.6185^{\frac{m}{n}}$. (See e.g., [35, 17].)

There are several studies regarding hardware design of Bloom filters including feasibility studies of incorporating them in network devices without considerable performance degradation (e.g., [13, 50]).

4 Bloom Filters with Counters

The problem of detecting the "fan" effect produced by worm propagation (see §1) consists of detecting duplicated packets leaving a network. In particular, we focus on malicious software which employs scanning methods. As reported previously [19], a common scanning strategy consists of sending out many packets with same payload and destination port, but different destination addresses within a short period of time.

Fan et al. [17] introduced BFWC applied to web cache techniques. We use BFWC in a router-based monitoring system. The idea is to enhance the routers of a network (or a subset of them — see §5) with BFWC to analyze the outbound traffic. For each outbound packet p, an *enhanced router* (i.e., a router that implements a BFWC) tracks how many times this packet has been forwarded in the recent past. If the packet has been forwarded more than t times for an appropriate *trigger threshold* t (precisely the behaviour that one would expect from a worm) an alarm is triggered.

To measure the efficiency of a BFWC we need to analyze the expected number of alarms. The value of the threshold t directly affects the expected number of alarms. To determine an appropriate value for t we need to take into account two different factors that may legitimately increase the counters (i.e., in the absence of malicious packets). First, we give some definitions.

Definition 2 (random packet). A random packet is a concatenation of a fixed number s of bits, where each bit may be 0 or 1 with equal probability (0.5). s is the (bit) size of the random packet.

Definition 3 (Class I traffic). Class I traffic is a sequence of n random packets.

Definition 4 (Class II traffic). Class II traffic is a sequence of n packets generated by a legitimate (e.g., non-worm) communication protocol in a network and contains at least one repeated packet.

When we speak of "Class I traffic", we assume that s is sufficiently large relative to n that the probability of any two packets being identical is negligible, or at least relatively small. In real networks, although most packet sequences are not randomly generated, some may look like "Class I traffic" in the sense that no packet is repeated.

We need to consider the increments to the counters caused by "Class I traffic", which increments the counters due to different packets resulting in the same hash value (i.e., statistical hash function collisions). Secondly, we need to consider the increments caused by "Class II traffic", i.e., non-worm traffic that contains repeated packets due to "normal"[2] repeated use of particular protocols (e.g., peer-to-peer, Google visits, Yahoo connections). Figure 2 illustrates these two types of traffic. The *safety gap* is the remaining number of increments allowed or provisioned before the threshold t is reached, i.e., beyond the expected number of Bloom-table entry collisions (increments) contributed by "Class I" and "Class II traffic". This *safety gap* should be sufficiently large to avoid false alarms due to expected statistical variation in "Class I" and "Class II traffic". "Class I traffic" increments are studied in the Appendix. Increments caused by "Class II traffic" are inherent to the characteristics of the system where the BFWC is deployed and such increments are not a major focus of this paper. However, the results of testing the BFWC under selected data sets (see §7) partially clarifies how "Class II traffic" affects the counters and what needs to be done to deal with this type of traffic.

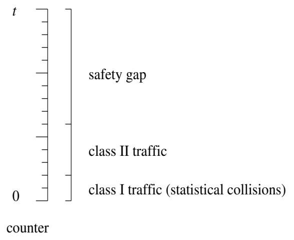


Figure 2: Types of non-worm traffic that increment a Bloom counter, and the resulting *safety gap*.

Figure 3 illustrates a Bloom filter with counters. Algorithm 1 describes the actions taken for each packet arriving at an enhanced router to be forwarded. Line 2 is a *threshold test* which triggers if a packet (worm or otherwise) is processed more than t times.

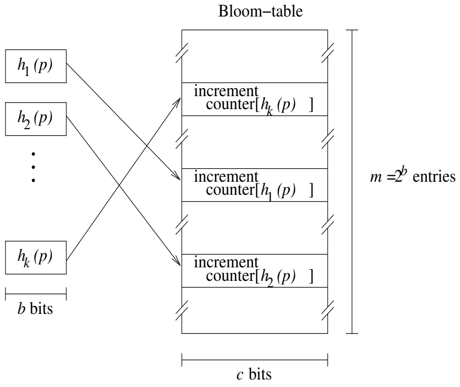

Bloom−table
$h_1(p)$
$h_2(p)$
$h_k(p)$
b bits
increment counter[$h_k(p)$]
increment counter[$h_1(p)$]
increment counter[$h_2(p)$]
$m = 2^b$ entries
c bits

Figure 3: Bloom filter with counters (BFWC).

In the Appendix we consider the probability that any k among the m counters have value greater than t each after n packets are processed.

Since memory capacity is limited, the entries of the Bloom-table are reset to 0 every fixed period of time, the *reset period*. The probability that a counter reaches its maximum value M solely based on "Class I traffic" is low because of the choice of parameters (e.g., size of Bloom-table, number of hash functions, expected number of packets). If a counter reaches its maximum value M, increments and wraps around to 0, important information is lost for the remaining time of the reset period. One way to address this is to use a design whereby the counters remain at the maximum value.

Algorithm 1 BFWC(packet p)

```
1. for $i \leftarrow 1$ to $k$
     $counter[h_i(p)] \leftarrow counter[h_i(p)]+1$
2. if $counter[h_i(p)] > t$ for all $i = 1, ..., k$ then
     trigger alarm
3. else process $p$ as usual
```

[2] "Normal" will vary depending on the nature of the network and its users.

5 Enhanced Routers

Consider BFWCs (as described in §4) with "appropriate" values for k and m according to the expected traffic volume n, and a threshold t with a reset period r. Suppose that these BFWCs are deployed in each router of the network. Worms spreading at t or more worm packets per second from a host would be detected by the first router that forward these packets. In this way, enhancing all the routers of a network would facilitate detection. However, costs and complexity of deployment might make it impossible or impractical to enhance every single router of a network. Therefore, we are interested in minimizing the number of enhanced routers while still being able to detect worms efficiently. In this section we describe how we can take advantage of Internet connectivity properties to find an appropriate small set of routers to enhance.

Consider a graph representing a network of autonomous systems where each node represents an autonomous system. Internet topology follows power-law relationships [16], including: *the number of nodes with d neighbours is proportional to d^{-c}, where $c > 1$.*

Several experimental studies have attempted to find an accurate value for c (e.g., [16, 6]). Since the power-law property induces hubs in the network (i.e., routers attached to "many" other routers – see [59]), we suggest that enhancing the routers of a small vertex cover including the hubs would be advantageous. A *vertex cover* of an undirected graph $G = (V, E)$ is a set of vertices $C \subseteq V$ such that for each link $(u, v) \in E$, u or v or both are in C. Given G and a positive integer $z \leq |V|$, the vertex cover problem is to find a vertex cover C such that $|C| \leq z$, if one exists. This problem is NP-complete [20].

Park et al. [43] show experimentally that a vertex cover of the autonomous-system-level Internet can be constructed with approximately 20% of the total number of nodes, and use random power-law networks to test the performance of a standard greedy algorithm [10, §35.1] – which we call *greedyVC* – for finding an approximated minimum vertex cover. This algorithm iteratively selects a node of highest degree and adds the node to the vertex cover, deleting the associated edges until all links are covered. We expect that enhancing the routers as selected by algorithm *greedyVC*, will be efficient in detecting a worm since the routers of the vertex cover tend to be the ones with most neighbours, and therefore the ones that will likely forward most of the traffic in the network (see results of §8).

6 Implementation Issues

In this section we discuss a few important implementation issues regarding BFWCs, four in particular: the packet-subset that is input to the hash functions, the nature of the hash functions themselves, the amount of memory required to store the Bloom-table, and the length of the reset period. While additional issues arise in an actual implementation, these are the most fundamental ones.

To be able to detect the "fan" produced by worm propagation we must look only at those portions of the packet that do not change. Our selection criteria for extracting a *packet-subset* is to retain those portions of the packet which we expect to be identical in a propagating worm (see §9 for comments regarding polymorphic worms). For example the destination IP address would vary, therefore this field should be excluded. For most worms, the destination *port* however, is expected to be constant. At the same time we want the packet-subset to contain enough information to avoid having non-identical packets which have the same packet-subset. Thus, we propose assembling packet-subsets of the form: [*dst_port, payload*]. Taking packet-subsets of this form prevents worms from evading detection by simply spoofing (varying) source IP addresses which would result in distinct hashes; i.e., worm packets not being counted by the same k counters of the Bloom-table. In addition, this packet-subset allows the detection of worm packets coming from different IP sources. However, by omitting the source IP address in the packet-subset, *flashcrowds* (i.e., sudden transmission of a significant number of packets from different IP addresses) of legal packets yielding equal packet-subsets may trigger false alarms; e.g., the same http request made at the same time by a considerable number of users in a corporation network, or any broadcasting application as on-line gaming. The destination port helps us to identify the vulnerable application.

It is important to use appropriate hash functions according to the requirements on performance and security. Several papers have discussed performance measures of hash functions implemented in both hardware and software. For example Grembowski et al. [22] show experimentally that SHA-512 implemented in hardware performs surprisingly well (i.e., with not much additional performance degradation) in terms of speed compared to other widely-known cryptographic hash functions. Nevelsteen et al. [38] compare several MAC algorithms against universal hash functions implemented in software and find that universal hash functions perform better than MAC algorithms in general. However, for our present purposes, we expect that even simple linear hash functions may suffice and provide performance advantages (e.g., see [24, page 151]). For the purposes of our present research also, we are less concerned about advanced attacks such as those proposed by Crosby et al. [11]; depending on his objectives, a worm writer might achieve greater success by spending additional time crafting better worms than by trying to exploit details of hash functions employed in a detection mechanism deployed in some subset of a large network. An attacker, however, could carefully fabricate packets that provoke false alarms due to hash collisions (i.e., different packets resulting in the same hash value, as described by Crosby et al.). This attack represents a potential problem requiring further consideration, and argues against the use of linear hash functions.

As discussed in §3, the efficiency of a BFWC depends on the total number of packets n expected to be processed over a fixed reset period r, the number of hash functions k, the number of counters m in the Bloom-table, and the bit size of each counter. The objective is to achieve an acceptable level of false alarms with reasonable memory size. Several factors need to be considered to design realistic and efficient Bloom filters, as illustrated in the following paragraphs.

Suppose we have a relatively low bandwidth OC-3 router interface[3] and 520 KBytes of memory available for the Bloom-table.[4] First, if the threshold t can fit within 4 bits (see Appendix) then we can allocate two counters per byte, giving a Bloom-table with $m = 1\,040\,000$ counters. Secondly, considering an average packet size of 1000 bits, this router can forward at most about 150 000 packets per second. Now, suppose we want to detect worm propagation within one-second windows, then a reset period of $r = 1$ second yields $n = 150\,000$ packets. Thirdly, the number of hash functions k can be obtained from equation (1): $k = \ln(2) \cdot 1\,040\,000/150\,000 = 4.8$ (to reduce overhead we can choose $k = 4$). Finally, an appropriate trigger threshold t under "Class I traffic" can be selected as explained in the Appendix. With these parameters and $t = 15$ the probability of a false alarm after one second of "Class I traffic" is approximately 1.52×10^{-64} (see Appendix), or 4.79×10^{-57} in one year (3.15×10^7 seconds), leaving ample room to accommodate links of sufficiently higher bandwidth.

Similarly, for an OC-192 router interface (with a capacity of $n = 10\,000\,000$ packets per second, considering an average packet size of 1000 bits) with 4 MBytes of available memory, a BFWC with $m = 8\,000\,000$ counters (i.e., two counters per byte), $k = 4$ hash functions, and $t = 15$, would have a false alarm probability of approximately 2.27×10^{-17} under one second of "Class I traffic," or 7.15×10^{-9} over one year. Using more memory, for example 64 MBytes (vs. 4 MBytes), would decrease this rate dramatically.

7 Performance of BFWCs under Specific Traffic Samples

In our evaluation we use packet-subsets of the form [*dst_port, payload*], as discussed in §6. It is clear that the BFWC would detect packets with identical packet-subsets forwarded at more than t packets per *reset period*. However, we expect that the number of non-worm packets with identical packet-subsets is not sufficiently large to trigger false alarms.

In our preliminary analysis, we use three different data sets to test a BFWC with the parameters described in Table 7: one hour of incoming traffic to our internal lab (*outside*), one day of traffic generated within our internal lab (*inside*), and one day of traffic of the well known DARPA data set [34] (the last two are known to contain no worm packets). The data sets consist of 113 463, 37 379, and 1 476 391 TCP and UDP packets, with average packet size of 10464, 7769 and 3011 bits per packet (respectively). The destination port frequency of the data sets are presented in Tables 1, 2, and 3. The predominant destination ports in the outside traffic (ports 80 and 32777) correspond to http (web) and rpc (remote service requests) packets, whereas the most frequent ones in the inside traffic (53, 631, and 9100) correspond to DNS (domain names address resolution) and printing protocols. As for the DARPA data set, the traffic is predominantly composed of telnet port 23 (remote login) and http packets on port 80. It is important to note that the incoming traffic to our internal lab had been pre-screened by one or more university firewalls, and thus a number of malicious packets may not have reached our lab. Although our data sets fall well short of the traffic volume in core routers – and further evaluation with representative traffic is necessary for higher confidence – we believe they validate the overall design principles of our approach. Similarly, despite known flaws in the DARPA data set [33], it suffices for a preliminary analysis.

Port	Freq.	Port	Freq.	Port	Freq.
80	43 582	137	382	513	40
32777	38 815	138	346	1715	27
1985	7 771	21	291	1667	26
16080	6 455	34852	124	1699	26
32771	3 111	32781	113	1674	25
32787	3 060	53	108	1683	25
32788	2 713	22	106	1706	25
520	2 429	427	71	1689	24
631	2 324	3127	47	32772	22
135	770	445	47	1096	21

Table 1: Most frequent destination ports of the outside traffic.

To avoid false alarms due to repeated non-worm packets we need to adjust the BFWC according to the predominant (non-worm) repeated traffic in the data sets. For our lab data set, we inserted all the outside 113 463 and the inside 37 379 packets into the BFWC. For the outside traffic we found that packets with the following destination ports triggered false alarms: 1985 (Hot Standby Router Proto-

[3] OC stands for optical carrier. One OC-3 link has a capacity of 155.52 Mbps.

[4] An OC-3 router interface is not realistic compared to core routers which may have several OC-192 interfaces, however one OC-3 router interface can be well used as an illustrative example.

Port	Freq.	Port	Freq.	Port	Freq.
53	7 830	34295	552	33298	270
631	6 118	32886	420	3932	263
9100	5 489	33175	414	68	192
32795	2 575	33788	414	67	188
138	2 088	1028	319	3179	177
123	1 544	33300	315	40216	156
38729	1 455	38833	300	3213	126
137	1 198	39135	287	1298	126
32775	615	32770	285	3159	114
32782	600	50566	272	34296	112

Table 2: Most frequent destination ports of the inside traffic.

Port	Freq.	Port	Freq.	Port	Freq.
23	398 734	123	10 511	15901	6 395
80	202 921	21262	8 983	16510	6 143
22	85 028	24638	8 580	18486	6 087
25	40 639	520	8 124	30865	5 563
53	28 354	17258	7 963	20551	5 334
161	19 992	23308	7 536	20	5 156
32770	19 413	16933	6 856	14942	5 024
15574	14 177	16586	6 757	15037	4 798
12862	10 891	29810	6 626	4702	4 356
1132	10 884	21	6 554	17782	4 349

Table 3: Most frequent destination ports of DARPA data set.

col), 520 (local routing process), and 631 (Internet Printing Protocol). We filtered out packets with these destination ports, such that they are not processed by the BFWC.[5] After this filtering, the BFWC triggered no false alarms (on these data sets)[6] and moreover was able to detect Slammer packets [36] trying to infect new targets. We were not expecting these latter packets in our network, and thus were pleasantly surprised (by their detection, rather than their presence; the absence of other worms generally present in today's Internet traffic likely resulted from enterprise-level filtering of worm traffic prior to this traffic reaching our outside lab link, as noted earlier). The detection of incoming Slammer packets confirms the effectiveness of BFWCs since incoming worm packets are more difficult to detect, as they may arrive at slower rates than if they were generated from within our lab.

For the inside traffic, no packets had to be filtered out. The BFWC did not trigger false alarms under this traffic; this was as expected, in traffic with no worms.

As for the DARPA data set, we filtered out several destination ports: 520, 161 (Simple Network Management Protocol), 6667 (Internet Relay Chat), and 1270 (OPSMAN). We inserted sequences of 155 550 packets each (the maximum number of packets per second that can be forwarded by a router with 3 OC-3 interfaces, considering an average packet length of 3000 bits per packet). After filtering packets with these destination ports, the BFWC triggered no false alarms, which is what we expected since the data set is known to have no malicious packets.

While obviously additional testing is required on larger data sets representing true Internet traffic, our preliminary results show the effectiveness of our system and suggest that the approach may indeed be effective in core routers.

8 Simulations

BFWCs have to be tuned up to take into account environment variables and characteristics inherent to the network, such as type of traffic (e.g., predominantly peer-to-peer), network topology fluctuations (e.g., constant failures or changes will trigger routing control packets), network speed and volume of traffic, available memory in routers, etc. The parameters of the BFWC used in the simulations are determined as discussed in §6. We note that while our motivation here was for packet matching, the simulation results appear to apply equally for stream matching, although more tuning may be required to address repeated context in streams. (We thank an anonymous referee for this observation.)

The goals of the simulations are (1) to determine an approximation of how efficient the BFWC is in detecting a worm when the BFWC is installed in the routers of an approximate minimum vertex cover and when all the routers of the network are enhanced, and (2) to determine an approximation of how many infections of a particular worm have been perpetrated by the time the worm is first detected by one BFWC.

Several network and worm simulators were considered [42, 52, 30, 40]. In our simulations we use NS-2 [40]. NS-2 provides flexibility on the topology of the network, making it possible to define "large" networks with a reasonable level of detail. The simulations were performed over the backbone topologies of three real ISPs. These topologies were made available by the Rocketfuel project [51, 31, 1], an Internet measurement tool. The simulation also uses real link delays, also made available by the Rocketfuel project. The characteristics of the ISP topologies used in our simulations are summarized in Table 4.

In our simulations, each node represents a router connected to a local area network (LAN). Internally, the simulator treats each router as a collection of hosts (e.g.,

[5] These adjustments may vary for different networks depending on factors such as configuration settings, predominant applications, network disruptions, etc.

[6] We do not claim zero false alarms for arbitrary Internet traffic, e.g., as might be seen by a backbone router. Again, testing on additional data sets would greatly improve our evaluation, although no data set is representative of all possible classes of real world traffic.

	ISP 1	ISP 2	ISP 3
minimum link delay	1ms	1	1
maximum link delay	17ms	29	44
number of routers	108	87	79
number of links	153	161	147
routers in vertex cover	33	46	37

Table 4: Characteristics of networks used in simulation.

clients, servers) on a LAN, as illustrated in Figure 4. To simplify the simulations, we assume that each router has a LAN associated with it.[7] Each router has a disjoint set of IP addresses associated, which represent the hosts of the LAN accessible through this router. To simulate IP addresses that do not reside within this ISP (i.e., foreign hosts), we assume that these addresses are reachable through arbitrary routers (assigned at the beginning of the simulation) of this ISP (these foreign hosts can also be thought of as non-vulnerable hosts residing within this ISP).

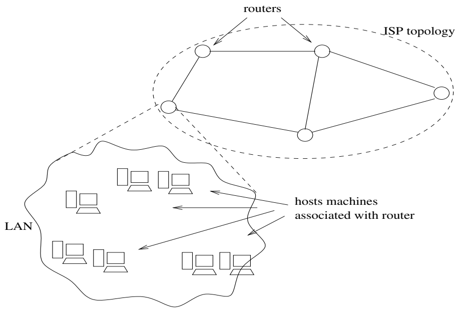


Figure 4: Example of network topology used in simulations.

We simulated a UDP worm that propagates in a single 400-byte UDP packet (not including the required headers) to a non-predetermined number of random IP addresses. The operation of this worm is described in Algorithm 2.

Algorithm 2 Simulated_UDP_Worm()

```
Upon reception of a worm packet, repeat forever:
  1. Send copy of the worm packet to a random IP
  address (i.e., host) using a pseudo-random number
  generator.
  2. Wait for $D$ seconds (the delay $D$ determines
  the stealthiness of the worm; a larger delay makes
  detection more difficult).
```

The propagation of a worm can be modelled with the "generic epidemic model" [12]. This model divides the hosts into two disjoint sets: *susceptible* and *infectious*. Infected hosts remain infected for an undetermined period of time (i.e., hosts do not transition from infectious to susceptible). Assuming that the network topology does not change (i.e., there are no new hosts or links, nor failures), and that there are no delays in the transmission lines, the number of infectious hosts at time T, $i(T)$ can be modelled with the following equation: $\frac{di(T)}{dT} = \beta \cdot i(T) \cdot s(T)$, where $s(T)$ is the number of susceptible hosts at time T and β is the propagation rate of the worm (i.e., the number of victims a single infectious host attempts to infect per unit of time).

Initially, in our simulations, all the hosts of the network are susceptible, having a process listening at port 0, which is assumed to be the service being exploited by the worm. The propagation rate, β, depends primarily on processor power and bandwidth available. We assume that hosts from other ISPs do not try to infect hosts of the ISP in question; in other words, we assume other ISPs are effectively preventing the spread of the infection.

To simulate multiple infected hosts within a router, whenever a host becomes infected, the propagation rate of worm packets leaving this router is increased accordingly to reflect the number of infected hosts within the LAN attached to this router. If a host attempts to infect another host residing within the same LAN, the attached router will not be able to see the worm packet, and consequently, even if this router was enhanced, the BFWC would not be able to count the worm packet. Thus, we do not detect worm packets propagating within a LAN.

Constant-rate UDP packets representing non-worm traffic are transmitted between random pairs of hosts. The rate of UDP packets transmitted by a host is fixed to 200 packets per second. The payload of each packet is a string of 500 random bytes (see Definition 3). This traffic was generated using the NS-2 built-in generator.

The traffic generated by the underlying routing mechanism should also be considered as part of non-worm traffic. The NS-2 simulator comes with several built-in routing algorithms, including distance vector algorithms [3]. Our simulation was configured to use the built-in implementation of the Distributed Bellman-Ford routing algorithm [3]. Route information packets are thus inherently added to the set of non-worm packets.

The simulation let us easily adjust the parameters of the network, the BFWC and the simulator itself. The configuration of the three ISP networks used in the simulations is shown in Table 5. The parameters of the worm and non-worm network traffic are listed in Table 6. The non-worm traffic can be classified as "Class I traffic" (see §4). The parameters of the BFWC were set up as discussed in §6, assuming that the routers have about 520 KBytes of memory available for the Bloom-table and that their maximum throughput is around 150 000 packets per second; a network operator should be able to adjust the BFWC parameters according to the characteristics of the underlying network

[7] In simulations, routers themselves are not considered to be hosts.

and its predominant traffic (see §6). The parameters of the BFWC appear in Table 7.

Parameter	Value or Configuration
address space	$\sim 3 \times 2^{16}$ (approx. 3 class B networks)
number of hosts associated with each router (see Figure 4)	100 (empirically set; Rocketfuel does not provide a value for this parameter)
which routers to "enhance"	routers as selected by an approximate minimum vertex cover

Table 5: Network parameters used in the simulations.

Parameter	Value
worm payload	fixed sequence of 400 bytes within one UDP packet
infection attempts per infected host	1 per worm delay period, D
worm delay, D	0.5 seconds
non-worm payload	random sequence of 500 bytes within a single UDP packet
rate of non-worm packets	200 packets/second, each 500 bytes

Table 6: Worm and non-worm traffic used in the simulations.

Parameter (see Figure 3)	Value
size of hash values, b	20 bits ($b = \log_2 m$)
entries in Bloom-table, m	1 048 576 ($m = 2^b$)
bits per counter, c	4
hash functions, k	4
threshold t	15
reset period r	1 second

Table 7: BFWC parameters used in the simulations.

The simulations were performed over three real ISP topologies with real link delays (see Table 4). The infection is started by a random host within the network. Figure 5 shows how the BFWC is able to detect the worm at its early stage of propagation. The curves show the number of infected hosts (cf. Figure 4) over time. The arrows show the first detection of the worm by an enhanced router.

The simulations results suggest that enhancing the routers of an approximate minimum vertex cover with BFWCs may be effective in detecting a worm within a reasonably short period of time, e.g., before the worm spreads to 1% (e.g., per Figure 5, 20 infections out of 79×100 hosts for ISP 3) of the (vulnerable) hosts in the entire network.

Also with additional experimentation we found that, with the parameters of our particular simulations, enhancing all the routers of the network is essentially no more effective than enhancing only those in an approximate minimum vertex cover. This fact becomes clear when we analyze the scenario. For example, Figure 6 depicts a portion of the (non-geographic) topology of ISP 1. According to algorithm *greedyVC*, the routers with most neighbours are most likely to be enhanced, and these routers are the ones forwarding most traffic in the network. For instance, assume that enhanced router 93 (top right) in the figure has reset period $r = 1$ second and threshold $t = 15$. Then it is unable to detect a worm propagating from within its associated LAN at less than 15 packets per second because its Bloom-table is reset every second. The alarms are triggered due to high volumes of worm packets collectively sent by a significant number of hosts. Therefore, even if all the routers were enhanced, a worm is most likely first detected by one of the routers with most neighbours, which most probably belongs to the vertex cover.

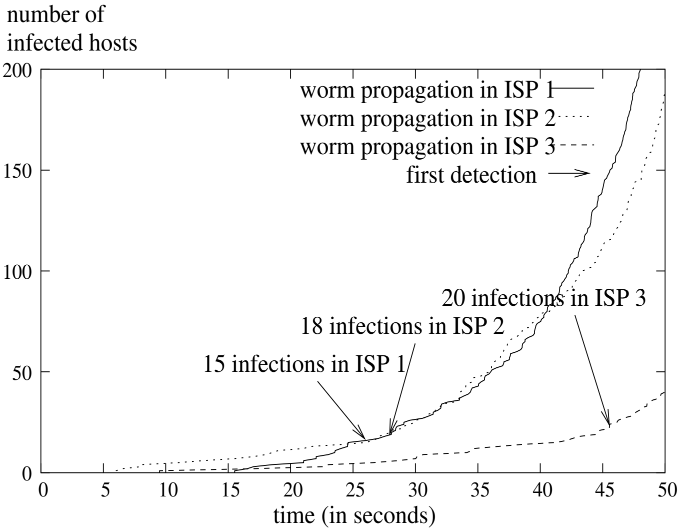
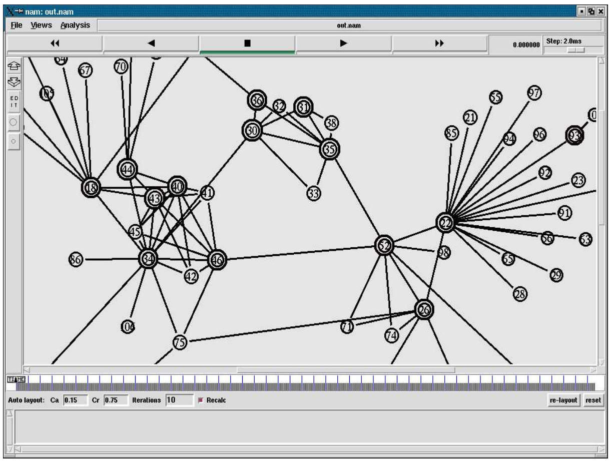


Figure 5: Detection of the worm enhancing the routers of an approximate minimum vertex cover.



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

Figure 6: Partial topology of ISP 1. Each node represents a router. Enhanced routers are illustrated with double circle.

9 Concluding Remarks

We have presented a router-based monitoring system using Bloom filters with counters (BFWC) for detecting worms in the propagation phase. It is based on stateful analysis of network traffic in the routers of a network of LANs, and the principle that worms send out many similar packets within a short period of time. A variation of Bloom filters is used to count the number of times that a packet is repeated. Our simulation results suggest that this approach may be effective in detecting a worm at the early stages of propagation. Further testing under real network conditions and additional data sets is necessary for a rigorous evaluation to answer critical questions about the utility and efficiency of the approach. As noted in §1, BFWCs are not effective against polymorphic or variable-key encrypted worms.

Our system cannot distinguish between repeated worm packets and repeated non-worm packets. Though this situation must be appropriately addressed (see next paragraph), this "feature" allows our approach to have applications beyond worm detection such as detection of dangerous traffic fluctuations, abusive use of certain services, and distributed denial-of-service attacks. This characteristic of our system does not prevent it from being useful for worm detection since we expect the frequency of repeated worm packets to dramatically exceed that of repeated non-worm packets (see safety gap in Figure 2), except for "stealth" worms.

An additional solution to preventing repeated non-worm packets from triggering false alarms is to pre-identify such packets where possible, and decrement corresponding Bloom-table counters every fixed period of time. This might be viewed as part of the network-specific "tuning".

Further evaluation should include testing the BFWC technique on data sets with significant numbers of packets of particular legitimate protocols (e.g., peer-to-peer) – and other legitimate traffic including e.g., flashcrowd or slashdot-effect behaviour at core router rates – which we expect to be susceptible to false alarms; and determining the tuning required under such traffic. Other follow-on work may involve: exploring how inclusion of more fields in the packet-subset affects the rate of undetected worms; running simulations under different worm propagation rates, and with fewer enhanced routines than those of a vertex cover; and enhancing only routers with particular characteristics (e.g., capacity, type of predominant traffic, number of neighbouring routers, etc.).

Acknowledgements. We thank Javier Govea and Tomás Pospíchal for their valuable help on the simulations and the Poisson approximation to the distribution of counters' values of the Bloom-table (respectively), and anonymous reviewers for their comments which considerably improved the presentation and positioning of this paper. We acknowledge the support of NCIT, MITACS, and Alcatel Canada. The first author is Canada Research Chair in Network and Software Security and acknowledges the support of an NSERC Discovery Grant, and the Canada Research Chairs Program. The third author acknowledges the support of an NSERC Discovery Grant; major parts of this research were carried out while he was a post-doctoral student in the School of Computer Science, Carleton University.

References

[1] T. Anderson, R. Mahajan, N. Spring, and D. Wetherall. Rocketfuel: An ISP topology mapping engine, 2003. `http://www.cs.washington.edu/research/networking/rocketfuel/` [Accessed: August 2, 2003].

[2] A.D. Barbour, L. Holst, and S. Janson. *Poisson Approximation.* Oxford University Press, New York, USA, 1992.

[3] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, Englewood Cliffs, USA, 1992.

[4] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(17):422–426, July 1970.

[5] M.V. Boutsikas and M.V. Koutras. On the number of overflown urns and excess balls in an allocation model with limited urn capacity. *Statistical Planning and Inference*, 104:259–286, 2002.

[6] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the Web. *Computer Networks*, 33(1–6):309–320, June 2000.

[7] A. Broder and M. Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2003–2004.

[8] The CERIAS Intrusion Detection Research Group. Digging for worms, fishing for answers. In *Proceedings of the Annual Computer Security Application Conference (ACSAC'02)*, Las Vegas, USA, December 9–13 2002.

[9] X. Chen and J. Heidemann. Detecting early worm propagation through packet matching. Technical Report ISI-TR-2004-585, University of Southern California, February 2004.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, McGraw-Hill, New York, USA, second edition, 2001.

[11] S.A. Crosby and D.S Wallach. Denial of service via algorithmic complexity attacks. In *Proceedings of the 12th USENIX Security Symposium*, Washington, USA, August 4–8 2003.

[12] D.J. Daley and J. Gani. *Epidemic Modelling: An Introduction.* Cambridge University Press, Cambridge, UK, 1999.

[13] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. Deep packet inspection using parallel Bloom filters. In *Symposium on High Performance Interconnects (HotI)*, pages 44–51, Stanford, USA, August 2003.

[14] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor. Longest prefix matching using Bloom filters. In *Proceedings of the Special Interest Group on Data Communication (SIGCOMM'03)*, pages 201–212, Karlsruhe, Germany, August 25–29 2003.

[15] S. Dharmapurikar and V. Paxson. Robust TCP stream reassembly in the presence of adversaries. In *Proceedings of the 14th USENIX Security Symposium*, Baltimore, Maryland, USA, August 2005.

[16] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the Internet topology. In *Proceedings of the Special Interest Group on Data Communication (SIGCOMM'99)*, pages 251–262, Boston/Cambridge, USA, August 31 – September 1 1999.

[17] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: A scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, June 2000.

[18] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley and Sons, New York, USA, 3rd edition, 1968.

[19] Fyodor. The art of port scanning. *Phrack Magazine*, 7(51), 1997. URL: `http://www.phrack.org` [Accessed: March 6, 2003].

[20] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, USA, 1979.

[21] E.-J. Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. URL: `http://eprint.iacr.org/2003/216/` [Accessed: January 7, 2004].

[22] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott. Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In *Proceedings of Information Security Conference (ISC 2002)*, volume 2433 of *Lecture Notes in Computer Science*, pages 75–89, Sao Paulo, Brazil, Sept. 30 – Oct. 2 2002. Springer.

[23] M. Handley, C. Kreibich, and V. Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium*, Washington, USA, August 13–17 2001.

[24] B. Horne, L. Matheson, C. Sheehan, and R.E. Tarjan. Dynamic self-checking techniques for improved tamper resistance. In *Proceedings of the First ACM Workshop on Digital Rights Management (DRM 2001)*, volume 2320 of *Lecture Notes in Computer Science*, pages 141–159. Springer, 2002.

[25] K. Joag-Dev and F. Proschan. Negative association of random variables, with applications. *The Annals of Statistics*, 11(1):286–295, 1983.

[26] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *IEEE Symposium on Security and Privacy 2004*, Oakland, CA, May 2004.

[27] H.-A Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of 13th USENIX Security Symposium*, San Diego, USA, August 9–13 2004.

[28] A. Kumar, J. Xu, L. Li, and J. Wang. Space-code Bloom filter for efficient traffic flow measurement. In *Proceedings of IMC*, Miami Beach, USA, October 27–29 2003.

[29] E. Levy. Worm propagation and generic attacks. *IEEE Security & Privacy*, 3(2):63–65, 2005.

[30] M. Liljenstam. Modeling of security and systems. A network worm modeling package for SSFNet, 2003. URL: `http://www.crhc.uiuc.edu/~mili/research/ssf/worm/` [Accessed: September 10, 2004].

[31] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weight using end-to-end measurements. In *Proceedings of the Internet Measurement Workshop 2002 (IMW'02)*, Marseille, France, 2002.

[32] A. Matrawy, P.C. van Oorschot, and A. Somayaji. Mitigating network denial-of-service through diversity-based traffic management. In *Proceedings of the 3rd Annual Conference on Applied Cryptography and Network Security (ACNS 2005)*, volume 3531 of *Lecture Notes in Computer Science*, pages 104–121, New York, USA, 2005. Springer.

[33] J. McHugh. Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information System Security (TISSEC)*, 3(4):262–294, 2000.

[34] MIT Lincoln Laboratory. DARPA intrusion detection evaluation: Data sets, 1999. URL: `http://www.ll.mit.edu/IST/ideval/data/data_index.html` [Accessed: April 1, 2004].

[35] M. Mitzenmacher. Compressed Bloom filters. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing (PODC 2001)*, pages 144–150, Newport, USA, August 26–29 2001.

[36] D. Moore, V. Paxon, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[37] C. Nachenberg. Computer virus-antivirus coevolution. *Communications of the ACM*, 40(1):46–51, January 1997.

[38] W. Nevelsteen and B. Preneel. Software performance of universal hash functions. In *Proceedings of Eurocrypt'99*, pages 24–41, Prague, Czech Republic, May 2–6 1999.

[39] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, Oakland, USA, May 8–11 2005.

[40] NS-2. The network simulator - NS-2, 2003. URL: `http://www.isi.edu/nsnam/ns/` [Accessed: September 10, 2003].

[41] I.-V. Onut, B. Zhu, and A.A. Ghorbani. A novel visualization technique for network anomaly detection. In *Proceedings of the 2nd Annual Conference on Privacy, Security and Trust*, Fredericton, Canada, October 13–15 2004.

[42] OPNET Technologies Inc. Opnet modeler, 2003. URL: `http://www.opnet.com` [Accessed: September 10, 2003].

[43] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. In *Proceedings of the Special Interest Group on Data Communication (SIGCOMM'01)*, San Diego, USA, August 27–31 2001.

[44] T. H. Ptacek and T. N. Newsham. Insertion, evasion and denial of service: eluding network intrusion detection. Technical report, Secure Networks, Inc. (Calgary), January 1998. URL: `http://www.aciri.org/vern/Ptacek-Newsham-Evasion-98.ps` [Accessed: November 6, 2005].

[45] M.O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[46] K. Shanmugasundaram, H. Brönnimann, and N. Memon. Payload attribution via hierarchical Bloom filters. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, Washington, USA, October 25–29 2004.

[47] C. Shannon and D. Moore. The spread of the Witty worm, 2004. URL: `http://www.caida.org/analysis/security/witty/` [Accessed: June 18, 2004].

[48] S. Singh, C. Estan, G. Varghese, and S. Savage. The EarlyBird system for real-time detection of unknown worms. Technical Report CS2003-0761, University of California, San Diego, August 4 2003.

[49] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design & Implementation (OSDI'04)*, San Francisco, USA, December 5 2004.

[50] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S.T. Kent, and W.T. Strayer. Hash-based IP traceback. In *Proceedings of the Special Interest Group on Data Communication (SIGCOMM'01)*, San Diego, USA, August 27–31 2001.

[51] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proceedings of the Special Interest Group on Data Communication (SIGCOMM'02)*, Pittsburgh, USA, August 19–23 2002.

[52] SSFNet. Scalable simulation framework network models, 2003. URL: `http://www.ssfnet.org/homePage.html` [Accessed: September 10, 2003].

[53] T. Toth and C. Kruegel. Connection-history based anomaly detection. In *Proceedings of the 2002 IEEE Workshop on Information Assurance and Security*, New York, USA, June 17–19 2002.

[54] J. Twycross and M.M. Williamson. Implementing and testing a virus throttle. In *Proceedings of the 12th USENIX Security Symposium*, Washington, USA, August 4–8 2003.

[55] A. Valdes and M. Fong. Scalable visualization of propagating Internet phenomena. In *Proceedings of the ACM Workshop on Visualization and Data Mining for Computer Security*, Washington, USA, October 29 2004.

[56] M. Vargas Martin. A monitoring system for mitigating fast propagating worms in the network infrastructure. In *Proceedings of the 18th IEEE Canadian Conference on Electrical and Computing Engineering (CCECE'05)*, Saskatoon, Canada, May 1–4 2005.

[57] S. Venkataraman, D. Song, P.B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *the Internet Society Proceedings of the Network and Distributed System Security Symposium (NDSS'05)*, San Diego, USA, February 3–4 2005.

[58] K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, Sophia Antipolis, France, September 15–17 2004.

[59] D.J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, Princeton, New Jersey, USA, 1999.

[60] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proceedings of ACM WORM'03*, Washington, D.C., USA, October 27 2003.

[61] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, USA, August 9–13 2004.

[62] M.M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the Annual Computer Security Application Conference (ACSAC'02)*, Las Vegas, USA, December 9–13 2002.

[63] Cliff C. Zou, Weibo Gong, and Don Towsley. Code Red Worm Propagation Modeling and Analysis. In *9th ACM Conference on Computer and Communication Security (CCS'02)*, Washington DC, USA, November 2002.

Appendix: Determining the Repeated Packet Threshold in BFWCs

In §4 we describe Bloom filters with counters (BFWC). Here, we describe a method for generating a table to allow the appropriate selection of a threshold t of repeated packets – at which an alarm is triggered – under "Class I traffic". We first review some basic statistical concepts.

Definition 5 (Bernoulli trials). *Bernoulli trials are repeated independent trials with only two possible outcomes for each trial and their probabilities remain the same over time: outcome* success *has probability p, and outcome* failure *has probability $q = 1 - p$.*

Often one is interested in the probability of obtaining s successes out of n Bernoulli trials regardless of the order of occurrence. This probability has a *Binomial distribution.*

Theorem 1. [18] *Let $b(s; n, p)$ be the probability that n Bernoulli trials with probabilities p for success and $q = 1-p$ for failure result in s successes and $n - s$ failures ($0 \leq s \leq n$). Then $b(s; n, p) = \binom{n}{s} p^s q^{n-s}$.*

For large n and small p, a direct evaluation of $b(s; n, p)$ may be impractical. In this case, we can use the Poisson approximation to $b(s; n, p)$:

$$b(s; n, p) \approx p(s; \lambda) = e^{-\lambda} \left(\frac{\lambda^s}{s!} \right) \tag{2}$$

where $\lambda = np$.

A generalization of the Binomial distribution for m possible outcomes for each trial is the *multinomial distribution* [18]. The possible outcomes for each trial are denoted E_i (for $i = 1, ..., m$) and the probability of each outcome is denoted p_i (for $i = 1, ..., m$, $\sum_{i=1}^{m} p_i = 1$, and $p_i \geq 0$). In a multinomial distribution, the probability that in n trials E_1 occurs s_1 times, E_2 occurs s_2 times, etc., is:

$$\frac{n!}{s_1! s_2! \cdots s_m!} p_1^{s_1} p_2^{s_2} \cdots p_m^{s_m} \tag{3}$$

where s_i are arbitrary non-negative integers and $s_1 + s_2 + \cdots + s_m = n$.

Observe that the number of successes for each possible outcome E_i after n trials can be approximated using (2).[8] Now we return to BFWCs by stating the following.

Observation 1. *Inserting N packets of "Class I traffic" into a BFWC with m counters and k independent hash functions that distribute the packets equiprobably into the m counters,[9] is equivalent to equiprobably throwing $n = Nk$ packets into m bins.*

Below we use the terms counter and bin interchangeably. Consider the following experiment.

Experiment 1. *Throw equiprobably n random packets into m bins.*

We are interested in finding a threshold t such that, after performing Experiment 1, the probability that k random bins have more than t packets is "sufficiently small" to be accepted according to our context (i.e., a tolerable number of false alarms due to Class I traffic). Using (3) may be impractical as m and n may be large. Instead, we use (2) to approximate the number of packets in each bin.

Example 1. *Consider a software implementation of a BFWC with $m = 1\,048\,576$ counters and $k = 4$ hash functions. After inserting $N = 150\,000$ random packets into the BFWC, the mean (expected) number of packets per bin is 0.572205. We want to set the value of t such that the probability that $k = 4$ counters taken at random each hold more than t packets is "sufficiently small". The probability that k random bins all have counters exceeding t, denoted $P[k, t, m, N]$, is approximated using (2) as:*

$$P[k, t, m, N] \approx \left(1 - \sum_{i=0}^{t} e^{-n/m} \frac{(n/m)^i}{i!} \right)^k . \tag{4}$$

Table 8 shows this approximation for several values of t. Based on the table, a network operator can choose an appropriate value for t according to the number of false alarms tolerable over some time period, under "Class I traffic".

t	$P[k, t, m, N]$	t	$P[k, t, m, N]$
2	1.75×10^{-7}	9	1.45×10^{-37}
3	6.50×10^{-11}	10	1.04×10^{-42}
4	1.03×10^{-14}	11	5.31×10^{-48}
5	8.02×10^{-19}	12	1.98×10^{-53}
6	3.43×10^{-23}	13	5.93×10^{-59}
7	8.69×10^{-28}	14	2.43×10^{-63}
8	1.38×10^{-32}	15	1.52×10^{-64}

Table 8: Poisson approximation to $P[k, t, m, N]$ after inserting $N = 150\,000$ packets into a BFWC with $k = 4$ hash functions and $m = 1\,048\,576$ counters.

The expected number of false alarms while inserting N packets is $\sum_{i=1}^{N} P[k, t, m, i]$. For large values of i, $P[k, t, m, i]$ can be estimated using (4). For small values of i, $P[k, t, m, i]$ becomes negligibly small. Observe that $P[k, t, m, i] > P[k, t, m, i']$ for $i > i'$. Therefore, a very coarse upper bound for the expected number of false alarms is $\sum_{i=1}^{N} P[k, t, m, i] < N \cdot P[k, t, m, N]$.

[8] The accuracy of the Poisson approximation to the multinomial distribution is well-studied (e.g., [25, 2, 5]). A Normal distribution may also be used as an approximation to $b(s; n, p)$ for large n [18].

[9] i.e., the probability of throwing a packet into counter i is $1/m$, for all $1 \leq i \leq m$.