Computer Networks 54 (2010) 1282–1299

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

HiFIND: A high-speed flow-level intrusion detection approach with DoS resiliency

Zhichun Li *, Yan Gao, Yan Chen

EECS department, Northwestern University, Evanston, IL, United States

ARTICLE INFO

Article history:
Available online 31 October 2009

Keywords:
Network monitoring
Network-level security and protection
Intrusion detection
Statistical detection
Data streaming
Attack resilience

ABSTRACT

Global-scale attacks like worms and botnets are increasing in frequency, severity and sophistication, making it critical to detect outbursts at routers/gateways instead of end hosts. In this paper, leveraging data streaming techniques such as the reversible sketch, we design HiFIND, a High-speed Flow-level Intrusion Detection system. In contrast to existing intrusion detection systems, HiFIND: (i) is scalable to flow-level detection on high-speed networks; (ii) is DoS resilient; (iii) can distinguish SYN flooding and various port scans (mostly for worm propagation) for effective mitigation; (iv) enables aggregate detection over multiple routers/gateways; and (v) separates anomalies to limit false positives in detection. Both theoretical analysis and evaluation with several router traces show that HiFIND achieves these properties. To the best of our knowledge, HiFIND is the first online DoS resilient flow-level intrusion detection system for high-speed networks (e.g. OC192), even for the worst-case traffic of 40-byte-packet streams with each packet forming a flow.

1. Introduction

Traffic anomalies and attacks are commonplace in today's networks, and identifying them rapidly and accurately is critical for operators of large networks. With the rapid growth of network bandwidth and fast emergence of new attacks, viruses and worms, existing network intrusion detection systems (IDS) [1–4] are insufficient due to lack of the following features.

1.1. First, scalability to high-speed networks

Most existing IDSes reside on a single host, only examining application-level [1] and system-level [2] logs, as such detailed information can identify attacks on individual machines. However, today's fast propagating viruses/worms (e.g., SQL Slammer worm) can infect most of the vulnerable machines in the Internet within 10 min [5] or even less than 30 s with some highly virulent techniques [6,7]. Thus, it is crucial to identify such outbreaks in their early phases, which can only be achieved by detection at high-speed routers instead of at end hosts [8]. Further, the edge network and even the backbone are suggested as good vantage points for worm containment [9]. Worm detection on those high-speed networks is a crucial prerequisite for containment. However, the existing schemes are not scalable to the link speeds and number of flows for high-speed networks.

Given a high-speed router, e.g., an OC-192 link (10 Gbps), each 40-byte TCP packet only has 32 ns to proceed [10]. For data recording in high-speed network IDSes, it is difficult for software-based approaches to keep up with the link speed. Thus, the recording part of high-speed IDSes has to be hardware implementable, and the following three performance features are strongly desirable: (1) a small amount of memory usage (to be implemented in SRAM); (2) a small number of memory accesses per-packet [11,12]; and (3) scalability to a large key space size. The

* Corresponding author. Tel.: +1 312 363 7530.
E-mail addresses: lizc@cs.northwestern.edu (Z. Li), ygao@cs.northwestern.edu (Y. Gao), ychen@cs.northwestern.edu (Y. Chen).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.10.016

last constraint is especially important for the coming decade: IPv6 with its 128 bit IP address is being adopted, especially in Asia. Thus, the system should scale to a key space of 2^{128} or 2^{256}. Meanwhile, other features are strong constraints which exclude many possible designs.

1.2. Second, attack resiliency

To bypass detection by an IDS, attackers can execute denial-of-service (DoS) attacks, or fool the IDS to raise many false positives so that real attack alerts are ignored. Thus, the attack resiliency of an IDS itself is very important. However, existing IDSes often keep per-flow states for detection, which is vulnerable for DoS attacks.

1.3. Third, attack root cause analysis for mitigation

Accurate attack mitigation usually requires IDSes to pinpoint the attack type and attack flows. To this end, we need to detect intrusions at the *flow-level* instead of based on the overall traffic [13,14]. Furthermore, we want to differentiate different types of attacks because mitigation schemes vary with the types of attacks. For example, for SYN flooding, attackers often spoof their IPs, but we can start the *SYN defender* [15], and/or change the IP address of the given domain name for the victim machines to alleviate the DoS effects. On the other hand, for port scans, we will use an ingress filter to block the traffic from the attackers' IPs.

1.4. Fourth, aggregated detection over multiple vantage points

Most existing network IDSes assume detection is on a single-router or gateway. However, as multi-homing, load balancing based routing, and policy routing become prevalent, asymmetric routing appears, and ingress and egress traffic go through different routers. Even for a connection between a certain source and destination, the packets may traverse different paths due to per-packet load balancing of routers which use a round-robin method to determine which path each packet takes to the destination [16,17]. Thus, observation from a single vantage point is often incomplete and affects detection accuracy. Meanwhile, it is very hard to copy all traffic from one-router to other routers/IDSes due to the huge volume of data.

1.5. Fifth, separating anomalies from intrusions for false positive reduction

To detect unknown attacks and polymorphic worms, statistics-based instead of signature-based intrusion detections have been adopted widely. However, many network element faults, e.g., congestion/failures, router misconfigurations, and polluted DNS entries, can lead to traffic anomalies which will be detected as attacks.

To meet the requirements above, we propose a new paradigm called DoS resilient High-speed Flow-level INtrusion Detection, HiFIND [18] leveraging recent work on data streaming computation and in particular, sketches [19,20]. Sketches are a kind of compact data streaming data structure which record traffic for given keys and are capable of reporting heavy traffic keys. Sketches are also linear, meaning we can take linear combinations of multiple sketches. (for details please refer to Section 3). Essentially, we want to detect as many attacks as possible. As the first step towards this ambitious goal, we aim to detect various port scans (port scans are an important way to detect large-scale worm propagation and botnet probing) and TCP SYN flooding. This will serve as an essential building block for the high-speed IDSes. Our goal is to identify and distinguish the port scans and SYN flooding in real-time on high-speed networks, and to obtain the attacks' key characteristics for mitigation. Note that while each of these attacks seems relatively easy to detect separately, or in an offline setting, it is in fact very hard to detect a mixture of attacks online at flow-level for high-speed networks. To the best of our knowledge, HiFIND is the *first* DoS resilient high-speed flow-level intrusion detection approach for port scans and TCP SYN flooding for high-speed networks (like 10 s of Gigabit links, e.g., OC192), even for the worst-case traffic of 40-byte-packet streams with each packet forming a flow.

To this end, we leverage and improve sketches, an efficient tool for data streaming computation, to record flow-level traffic as the basis for statistical intrusion detection. Although proposed in [19,20], sketches have not been applied to building IDSes for the following reasons:

- Sketches can only record certain aggregated metrics for some given keys. For each flow, there are numerous possible keys: source/destination IP addresses, source/destination ports, source/destination prefixes, protocols, etc., and any of these combinations. Since, it is not feasible to try all possible combinations of the metrics, given the threat model, what would be the minimal set of metrics for monitoring?
- Existing sketches are all one-dimensional, i.e., they can only record the values for a specific metric. However, various forms of attacks are often hard to identify with such single dimensional information. For example, both horizontal scans and un-spoofed SYN flooding exhibit a large number of unsuccessful connections aggregated with the (source IP, destination port) pair. However, it is difficult to differentiate these different kinds of attacks unless the distribution of the attacks on the destination IPs are also considered.

In this paper, we address these two challenges and build the HiFIND prototype system to meet the five requirements mentioned before. We make the following contributions:

- We analyze the attributes in TCP/IP headers and select an optimal small set of metrics for flow-level sketch-based traffic monitoring and intrusion detection. Based on that, we build the HiFIND prototype which is DoS resilient and can provide high-speed flow-level intrusion detection online as demonstrated by both analytical and experimental results.
- To analyze the attack root cause for mitigation, we design efficient two-dimensional (2D) sketches to dis-

tinguish different types of attacks. Both analytical and empirical results show the effectiveness of the 2D sketches.
- We aggregate the compact sketches from multiple vantage points (e.g., routers) to detect intrusion in the face of asymmetric routing and multi-path routing caused by per-packet load balancing of routers. To the best of our knowledge, HiFIND is the first system that can work in such environments.
- For false positive reduction, we propose several heuristics to separate SYN floodings from network/server congestions and misconfigurations (e.g., polluted DNS entries).

As shown in Fig. 1, HiFIND detection systems can be implemented as black boxes attached to high-speed routers (edge network routers or backbone routers) of ISPs without affecting the normal operation of the routers.

Detection on edge networks is particularly critical, powerful and efficient (without deploying IDSes on all the edge hosts), according to a recent research agenda for large-scale malicious code by DARPA [8].

For evaluation, we first test the router traffic traces collected at Lawrence Berkeley National Labs. We then apply HiFIND for on-site detection at the Northwestern University (NU) edge routers: we record each minute of traffic with reversible sketches on the fly. At the end of each minute, we use the recorded sketches for online detection. In particular, the one day experiment data consist of 239M network flows of 1.8TB total traffic. We validate the SYN flooding and port scans detected, and find the HiFIND system is highly accurate. The 2D sketches successfully separate the SYN flooding from port scans, and the heuristics effectively reduce false positives of SYN flooding. The evaluation demonstrates that HiFIND significantly outperforms existing approaches like Threshold Random Walk (TRW) [21], TRW with approximate caches (TRW-AC) [22], and Change-Point Monitoring (CPM) [13,14]. Compared with statistical detection on complete flow-level data logs, we have almost the same detection accuracy, but use much less memory.

The HiFIND system runs in real-time, and requires a small number of memory accesses per-packet. With a Pentium Xeon 3.2 GHz machine and normal DRAM memory, we record 239M flows with one reversible sketch in 20.6 s, i.e., 11.6M insertions/second. For the worst-case scenario with all 40-byte packets, this translates to around 3.7 Gbps. Our prototype single FPGA board for reversible sketches can achieve a throughput of over 16 Gbps for all 40-byte-packet streams. For the NU on-site experiments over a total of 1430 min, HiFIND on average uses only 0.34 s to detect intrusions for each minute of traffic, and the standard deviation is 0.64 s.

The organization of this paper is as follows. First, we survey related work in Section 2. In Section 3, we introduce the sketches and reversible sketches as the basis for high-speed network monitoring. We then introduce the HiFIND architecture, discuss the threat model and flow-level detection design in Section 4. The two-dimensional sketches are presented in Section 5, and evaluation methodology and results are in Section 6. Finally, we show the potential limitations of HiFIND in Section 7 and conclude in Section 8.

2. Related work on intrusion detection systems

Although some vendors claim to have multi-gigabit statistical IDSes (e.g., Arbor Networks' Peakflow Traffic [23] and Symantec's Manhunt [24]), they usually refer to *average* traffic conditions and use packet sampling [25,26] which has two shortcomings. First, sampling is not scalable, especially after aggregation; there are up to 2^{64} flows defined by source and destination IP addresses. Second, long-lived traffic flows, increasingly prevalent for peer-to-peer applications, will be split up if the time between sampled packets exceeds the flow timeout [25]. In contrast, HiFIND records *every* packet in the traffic summary, and targets *worst-case* performance of tens of gigabits, when all the packets are small, e.g., 40-byte-packet streams as in TCP SYN flooding attacks.

Wagner et al. design a scan detection tool for VPN gateway [27], but scalability is not a major concern. In contrast, we target the network gateway with high bandwidth and focus on scalability and attack resilience. Gross et al. develop a distributed alert correlation scheme called Selecticast

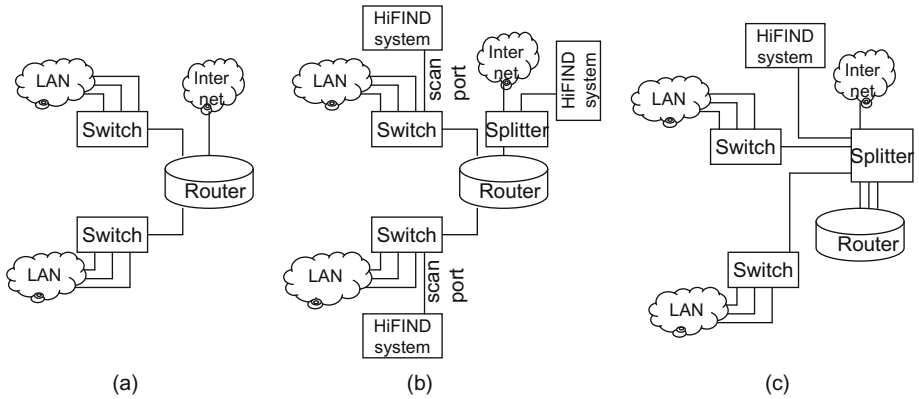

Fig. 1. Attaching the HiFIND systems to high-speed routers. (a) original configuration, (b) distributed configuration for which each port is monitored separately, and (c) aggregate configuration for which a splitter is used to aggregate the traffic from all the ports.

[28]. On the other hand, we mainly design a single detection sensor.

Many network IDSs like Bro [3] and Snort [4] check packet payload for virus/worm signatures. However, such schemes are not scalable for high-speed network links. Recent work has proposed detecting large-scale attacks, like DoS attacks, port scans, etc., based on the statistical traffic patterns. They can roughly be classified into two categories: detecting based on the overall traffic [29,13,14,30] and flow-level detection [3,4,21].

With the first approach, even when we can detect the attack, we still do not have any flow or port knowledge for mitigation. Moreover, attacks can be easily buried in the background network traffic. Thus, such detection schemes tend to be inaccurate; for example, CPM [13,14] will detect port scans as SYN floodings as verified in Section 6. For the second approach, such schemes usually need to maintain a per-flow table (e.g., a per-source-IP table for TRW [21]) for detection, which is not scalable and thus provides a vulnerability to DoS attacks with randomly spoofed IP addresses, especially on high-speed networks. TRW was recently improved by limiting its memory consumption with approximate caches (TRW-AC) [22]. However, spoofed DoS attacks will still cause collisions in TRW-AC, and leave the real port scans undetected.[1]

The existing schemes can detect specific types of attacks, but will perform poorly when facing a mixture of attacks as in the real world. People may attempt to combine TRW-AC and CPM to detect both scans and SYN flooding attacks. However, each of these two approaches can work properly only when the other one works well, which is a chicken-and-egg problem. TRW and TRW-AC are vulnerable to spoofed DoS attacks, such as SYN flooding, unless CPM can detect DoS attacks accurately and remove them from the traffic. However, with port scans in the traffic, CPM will always detect these port scans as SYN floodings. As a result, this combination cannot work.

Table 1 shows the high-level functionality comparison of our approach to the other methods. Backscatter detects the SYN flooding attacks by testing the uniform distribution of destination IPs to which the same source (potential victim) sends SYN/ACK [29]. Thus, it is another flow-level scheme and can only detect spoofed DoS attacks when the source IP addresses are randomly spoofed. We use this for validating the SYN flooding detected by HiFIND.

There is a significant amount of prior work on efficient and online heavy hitter detection [31,32,12,33–35]. However, these approaches are limited in their applicability to online intrusion detection in that: (1) they lack the ability to differentiate different types of attacks; (2) they cannot work with Time Series Analysis based detection algorithms; and (3) they cannot be applied to asymmetric routing environments.

Venkataraman et al. propose efficient algorithms to detect superspreaders, sources that connect to a large number of distinct destinations [36]. They can detect horizontal scans and worm propagation, but may have high false positives with P2P traffic where a single host may connect to many peers for download. Also, their approach cannot differentiate different types of attacks. PCF was recently proposed for scalable network detection [37]. It uses data structures similar to the original sketch, and is not reversible. So even when attacks are detected, the attacker or victim is still unknown, making mitigation impossible. Similar to the approaches discussed before, they do not differentiate among various attacks.

For intrusion classification, Lakhina et al. recently examine the traffic feature distribution with entropy, for all OD flows between a pair of point of presence (POPs) [38]. Their scheme, however, cannot give the key of culprit flows for mitigation even when spotting anomalies. Similarly, other recent works [39,40] use traffic feature distributions or patterns to classify network flows for anomaly detection, but their methods require the complete flow tables which are often unavailable for high-speed networks of 10 s of Gigabits.

Table 1
Functionality comparison of five approaches.

Approaches	Spoofed DoS	Non-spoofed DoS	Horizontal scan	Vertical scan
HiFIND	Yes	Yes	Yes	Yes
TRW (AC)	No	No	Yes	Yes (TRW-AC)
CPM	Yes, but with high FP with port scans		No	No
Backscatter	Yes	No	No	No
Superspreader	No	No	Yes	No

3. Background on sketches for high-speed network monitoring

3.1. k-ary Sketch

There are two key primitives in the analysis of a live network traffic stream: heavy hitter detection and heavy change detection. The former finds flows that constitute more than a given threshold fraction of the total traffic stream. The latter detects flows whose size changes significantly from one stream to another. There is a significant amount of prior work on efficient and online heavy hitter detection [12,33–35]. Efficient online heavy *change* detection, however, remains a challenging problem of significant interest because it is more general and powerful than heavy hitter detection. "Change" is a concept that ranges over a gamut from simple absolute or relative changes, to linear transformation changes.

The sketch, a recently proposed data structure, has proven to be useful in many data stream computation applications [41]. We have designed, implemented and evaluated a variant of the sketch, namely the *k*-ary sketch [42], and described how to detect heavy changes in massive data streams with small memory consumption, constant update/query complexity, and accurate estimation guarantees [42]. The *k*-ary sketch is similar to the count sketch

[1] As the authors mentioned in [22], when the connection cache size of 1 million entries reaches about 20% full, each new scan attempt has a 20% chance of not being recorded because it aliases with an already-established connection. Actually, during spoofed DoS attacks, such collisions can become even worse.

[43], however, the most common operations on *k*-ary sketch are more efficient than the corresponding operations defined on count sketches [44].

For most statistical approaches, we can model the network traffic as a stream of {*key*, *value*} pairs. The *key* can be IP addresses, port numbers, etc. The *value* can be any accumulatable feature, such as packet number and traffic volume. For instance, we can parse the network traffic based on this model to count the packet number of every unique source IP address. The *k*-ary sketch provides the functionality to archive this calculation with low cost. The *k*-ary sketch has three basic functions: UPDATE, ESTIMATE and COMBINE. Among them, UPDATE is used most frequently and has the most stringent real-time requirement. Table 2 shows the definition of these functions. Suppose there is a sketch with a source IP as the *key* and packet size as the *value*. As shown in Fig. 2, when a 68-byte-packet with source IP 10.0.0.5 arrives, the UPDATE function will increase the values of buckets mapped by the source IP 10.0.0.5 in the sketch data structure by 68. The ESTIMATE function will return an unbiased estimation of the value (total traffic volume) given a source IP and a sketch. The COMBINE function can linearly combine several sketches into a single one. The key feature of the COMBINE function is to support aggregate queries over multiple data streams, i.e., to find the top heavy hitters and their keys from the linear combination of multiple data streams, for temporal and/or spatial aggregation. With sketches, we can record hundreds of millions of flows with only a few hundred kilobytes of memory. See [42] for more details.

3.2. Reversible k-ary Sketch

Although the original sketches have good linear properties and can accurately estimate the value for any given key, they have one major drawback: they are not *reversible*. A sketch cannot efficiently report the set of all keys that have large values estimated in the sketch. This means that to output all the keys whose value is larger than some threshold, we would have to know which keys to query. One possible solution is to exhaustively test all possible keys. Unfortunately, this option is not scalable. Another solution, similar to other heavy hitter approaches, is to update and query each {*key*, *value*} pair. For every pair, after updating, we can query the value to see whether the value of the key is larger than some threshold, then decide whether to output the key. This requires us to do recording and heavy key inference at the same time, and we cannot take advantage of the linearity of sketches to combine them together.

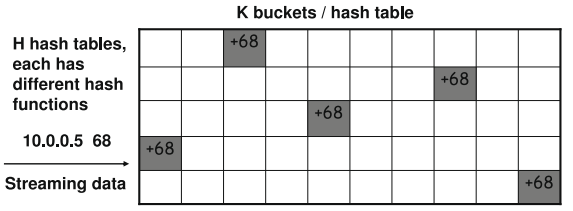

Fig. 2. A sample *k*-ary sketch structure and its UPDATE operation for a sample input stream with an item (10.0.0.5 68). The key is 10.0.0.5 and the value is 68.

To address these problems, in our previous work we propose a novel framework for efficiently *reversing* sketches [19,20], which allows us to have separate stages for update, combine and inference. The basic idea is to hash intelligently by modifying the input keys and/or hashing functions so that we can recover the keys with large estimated values, without sacrificing detection accuracy. We also use a second verifier sketch with 2-universal hash functions to reduce false positives. In fact, we obtain analytical bounds on the false positives with this scheme. In addition to the three basic functions of sketches, reversible sketches also support the INFERENCE function, which will return the keys whose values are larger than a given threshold. Given the example before, we may want to get all source IPs whose total traffic volume is larger than 1MB in a certain time period. We just need to record the traffic of that period with reversible sketches, and call the INFERENCE function to give the list of all such source IPs. Compared with original *k*-ary sketches, we only add negligible extra memory consumption (4–8 kB) and few (4–8) additional memory accesses per-packet for the UPDATE function, but we achieve an efficient and accurate INFERENCE function. The other two functions are unchanged. For more details, please refer to our paper [20].

4. Architecture of the HiFIND system

4.1. System architecture

Fig. 3 shows the architecture of the HiFIND system. First, we record the network traffic with sketches using the UPDATE function in each router. Based on linearity of the sketches, we summarize the sketches over multiple

Table 2
Function of sketches (*S*-Sketch, *v*-Value, *y*-Key, *Y*-Set of keys, *t*-Threshold).

Functions	Descriptions	*k*-ary sketch	Reversible sketch
UPDATE(S, y, v)	Update the corresponding values of the given key into the sketch in the monitoring module	√	√
v = ESTIMATE (S, y)	Reconstruct the signal series for statistical detection for a given key in the anomaly detection module	√	√
S = COMBINE ($c_1, S_1, \ldots, c_k, S_k$)	Compute the linear combination of multiple sketches $S = \sum_{k=1}^{l} c_k \cdot S_k$ (c_i is coefficient) to aggregate signals in the anomaly detection module	√	√
Y=INFERENCE (S, t)	Return the keys whose values are larger than the threshold in the anomaly detection module		√

routers into an aggregate sketch, and apply different time series analysis methods for aggregate sketches to obtain the forecast sketches for change detection by the COMBINE function. The forecast time series analysis method, e.g., EWMA (exponentially weighted moving average) and Holt–Winter algorithm [45], can help remove noise. By subtracting the forecast sketch from the current one, we obtain the forecast error sketches. Intuitively, a large forecast error implies there is an anomaly, thus the forecast error is the key metric for detection in our system. Moreover, we aggregate the 2D sketches in the same way and adopt them to further distinguish different types of attacks. We also apply other false positive reduction techniques as discussed in Section 4.5. Finally, we use the key characteristics of the culprit flows revealed by the reversible sketches to mitigate the attacks. Note that the streaming data recording process needs to be done continuously in real-time, while the detection process can be run in the background executing only once every interval (e.g., every second or minute) with more memory (DRAM).

As discussed in Section 1, due to multi-homing and per-packet load balancing features in routers, asymmetric routing and multi-path routing (where there are multiple routing paths between a pair of source and destination) emerged. However, many existing flow-level intrusion detection approaches have to keep the connection states (e.g., SYN and SYN/ACK) for both directions of a TCP connection. Therefore, they can only obtain one-way information with asymmetric routing, and will cause high false positives. To deal with asymmetric routing, some work proposes to use one-way traffic like {SYN, FIN} pairs to detect attacks [13,14]. However, such approaches can be easily bypassed if the attackers send a FIN packet for each SYN packet.

Thus, to deal with asymmetric routing, for many existing IDS systems, we have to transport all the packet traces or all connection states from one router to the other. Obviously this is very expensive. Moreover, if the link is congested when an attack happens, transmission of this data can be very slow. For example, TRW and TRW-AC [21,22] need the state table to determine how to process the next packet. It is almost impossible for them to record the traffic in several state tables in different locations, and then only transmit the state tables to some central site and combine these tables for detection. They have to collect and combine all packet-level traces to a centralized location for aggregated detection, which as analyzed before is very expensive and is not realistic.

Furthermore, some routers may use per-packet load balancing so that packets of the same flow may even traverse different paths. For instance, Fig. 4 shows a campus network topology where there are three edge routers, and each of them is connected to tens of different department and institute networks. Due to asymmetric routing, the incoming SYN packets and outgoing SYN/ACK packets for Department 1 go through different routers. Furthermore, the Department 1 router enables per-packet load balancing, so the SYN/ACK packets go through different edge routers. Detection can be executed on each department router, but it requires deployment of IDSes on tens or hundreds of different routers. On the other hand, none of the three edge routers has complete traffic information for any of the department subnet.

In contrast, for HiFIND, we summarize the traffic information with compact sketches at each edge router, and deliver them quickly to some central site. Then, with the linearity of the sketches, we can aggregate the sketches by applying the COMBINE function and the resulting sketch has all the traffic information as if all the traffic went through the same router.

Next, we will introduce the threat model considered in our prototype as well as the detection algorithms and time series analysis method used. After that, we present the heuristics for separating anomalies from attacks and analyze the DoS resilience of the HiFIND system.

4.2. The threat model

Ultimately, we want to detect as many different types of attacks as possible. As a first step, we focus on detecting the two most popular intrusions: TCP SYN flooding (DoS) attack and port scan/worm propagation. We focus on TCP flooding here because it is reported that more than 90% of DoS attacks are TCP SYN flooding attacks [13,14]. Actually, our work also can be easily extended to detect UDP or ICMP flooding from multiple sources by using the number of packets or traffic volume as the *value* stored in sketches.

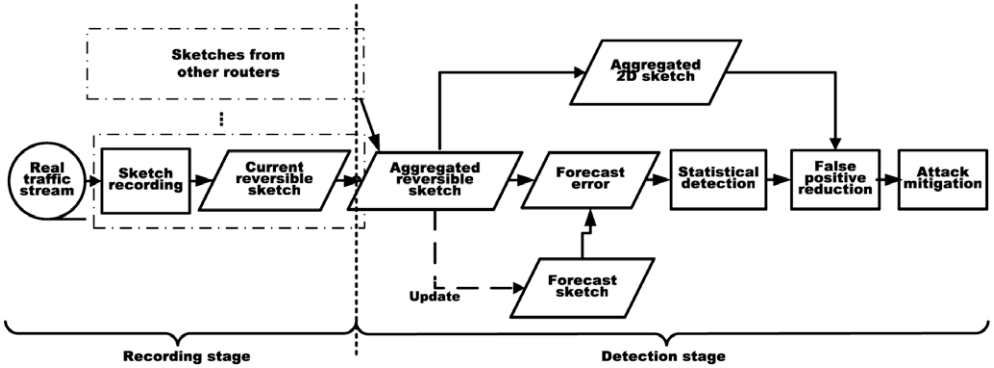

Fig. 3. HiFIND system architecture.

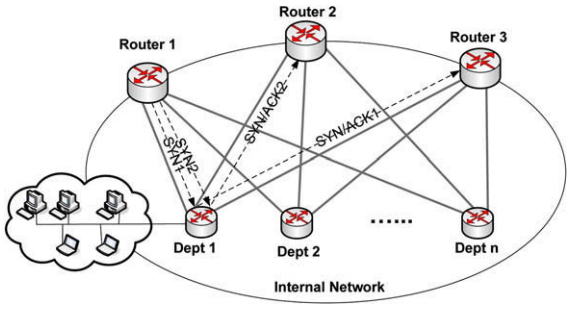

Fig. 4. Sample network topology with asymmetric routing and multi-path routing.

Scans are probably the most common type of intrusion. Most of them are caused by either worm propagation or botnet sweeps. According to statistics data by CERT [46], most existing worms are TCP worms. TCP worm propagation starts with massive TCP connection requests, i.e., port scans. Our system detects generic port scans. Thus, it can detect propagation for both known and *unknown* worms. Port scan is also a key technique employed for botnet recruitment [47].

Based on source/destination IP and the port number combinations, there are three well-known types of scans: *horizontal scan*, *vertical scan*, and *block scan* [48,49] as illustrated in Fig. 5.

Unlike DoS attacks, for port scans, the attacker needs to use a real source IP address, since he/she needs to see the result of the scan in order to know what ports are actually open [49,48]. Although the "idle scan" allows completely blind scans, it is based on predictable IP-ID sequence numbers and recent versions of operating systems, such as OpenBSD, Solaris and Linux, which have made the IP-ID sequences less predictable and rendered the attack obsolete [50]. Among the three types of scans in Fig. 5, horizontal scans are the most common type of scan, which scans a given port on IP addresses in some range of interest. The port number is often unique because it reflects what the vulnerability attackers (or the virus/worm) try to exploit. A vertical scan is a scan of some or all ports on a single host. The attacker is interested in this particular host, and wishes to characterize its active services to find which exploits to attempt [49]. The third type of scan, a block scan, is a combination of horizontal and vertical scans over numerous services on numerous hosts [49]. Block scan is not very common in practice, thus we focus on horizontal and vertical scans.

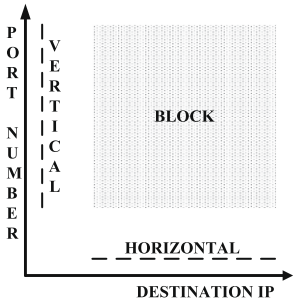

Fig. 5. Three types of scans.

It is crucial to distinguish SYN flooding, horizontal scans and vertical scans because network administrators need to apply different mitigation schemes for different attacks. For SYN flooding, we can start the *SYN defender* [15], *SYN proxy*, and/or *SYN cookies* [51] for the particular victim machines to alleviate the DoS effects. For port scans, we can use an ingress filter to block the traffic from the attacker's IP. For horizontal scans, we can also pay particular attention to popular port numbers scanned by attackers, and block scanned port numbers which are not common services. For vertical scans, we can quarantine the victim machine for further inspection, or block all the scan traffic towards the victims if necessary.

4.3. Sketch-based detection algorithm

The challenge is to detect and differentiate attacks above scalably and accurately, and to get the key characteristics (e.g., source IPs for scans) of such attacks in order to mitigate them efficiently.

We denote the key of a sketch as `K`, the feature value recorded as `V`, and the reversible sketch as `RS(K,V)`. We also denote the number of SYN packets as #SYN, and the number of SYN/ACK packets as #SYN/ACK. Since normally we only extract fields in the IP header, the possible fields we can use are shown in Table 3.

Here, we only consider the attacks in TCP protocol, i.e., the TCP SYN flooding attacks and TCP port scans. Normally, attackers can choose source ports arbitrarily, so `Sport` is not a good metric for attack detection. For the other three fields, we can consider all the possible combinations of them, but the key (`SIP,DIP,Dport`) can only help detect non-spoofed SYN flooding, so we do not use it in the detection process. Table 5 shows the other combinations and their uniqueness. Here, we define the *uniqueness* of a key as the capability of differentiating between different types of attacks, which is measured by the types of attacks that the key metric is related to and can help detect. For example, the count of unsuccessful connections aggregated by {`SIP`} can be used to detect non-spoofed SYN flooding attacks (we count it as 0.5), horizontal scans and vertical scans, so its value of uniqueness is 2.5. The best key would ideally correspond to only one type of attack. Normally a key can be related to several types of attacks, so we need to use more than one dimension to differentiate these attacks as shown in Section 5.

From Table 5 we can tell that the combinations of two fields have better uniqueness than single fields, so we use the three combinations of two fields as keys for the

Table 3
The fields in IP header used in detection.

The destination IP	`DIP`
The source IP	`SIP`
The destination port	`Dport`
The source port	`Sport`

reversible sketches. Our detection has the following three steps:

Step 1 We use RS({DIP, Dport}, #SYN-#SYN/ACK) to detect SYN flooding attacks because it usually targets a certain service as characterized by the Dport on a small set of machine(s). The value of #SYN-#SYN/ACK means that for each incoming SYN packet, we will update the sketch by incrementing one, while for each outgoing SYN/ACK packet, the sketch will be updated by decrementing one. In fact, similar structures can be applied to detect any partial completion attacks [37]. The reversible sketch can further provide the victim IP and port number for mitigation as in Section 4.2. We denote this set of DIPs as *FLOODING_DIP_SET*.

Step 2 We use RS({SIP, DIP}, #SYN-#SYN/ACK) to detect any intruder trying to attack a particular IP address. The detected attacks can be non-spoofed SYN flooding attacks or vertical scans. For each {SIP, DIP} entry, if DIP $\in$ *FLOODING_DIP_SET*, we put the SIP into the *FLOODING_SIP_SET* for the next step; otherwise the {SIP, DIP} is the attacker's IP and victim IP of a vertical scan.

Step 3 We use RS({SIP, Dport}, #SYN-#SYN/ACK) to detect any source IP which causes a large number of uncompleted connections to a particular destination port. For each {SIP, Dport} entry, if SIP $\in$ *FLOODING_SIP_SET*, it is a non-spoofed SYN flooding; otherwise, it is a horizontal scan.

As a whole, we need three reversible sketches to record the traffic characteristics. The classification rules are in Table 4. We then use time series analysis in Section 4.4 and detect intrusions when there is a big forecast error. After the detection, we use the heuristics described in Section 4.5 and 2D sketches in Section 5 to reduce the false positives.

4.4. Forecast with time series analysis methods

Here we apply both EWMA and the Holt–Winter algorithm as the forecast models to do change detection. By default, we use #SYN-#SYN/ACK as the forecast value in these time series analysis methods below in our system.

4.4.1. Holt–Winter forecasting algorithm

Holt–Winter Forecasting relies on the premise that the observed time series can be decomposed into three components: a baseline, a linear trend, and a seasonal effect. The algorithm assumes that each component evolves over time. This is modeled by applying exponential smoothing to incrementally update the components.

For each client, we define y_t as the #SYN-#SYN/ACK in time interval t. Given that attack is a relatively fast process, we use a minute interval for detection. The prediction of y in the next time interval $t+1$ is the sum of the three components: $\hat{y}_{t+1} = a_t + b_t + c_{t+1-m}$.

The update formulae for the three components, or coefficients a, b, c are:

- Baseline ("intercept"): $a_t = \alpha(y_t - c_{t-m}) + (1-\alpha)(a_{t-1} + b_{t-1})$.
- Linear Trend ("slope"): $b_t = \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1}$.
- Seasonal Trend: $c_t = \gamma(y_t - a_t) + (1-\gamma)c_{t-m}$.

In our system, we respectively use COMBINE $(\alpha, y_t - c_{t-m}, 1-\alpha, a_{t-1} + b_{t-1})$, COMBINE $(\beta, a_t - a_{t-1}, 1-\beta, b_{t-1})$ and COMBINE $(\gamma, y_t - a_t, 1-\gamma, c_{t-m})$ to implement baseline, linear trend and seasonal trend. Following the guidelines in [45], we set α, β and γ all to 0.1. We use the INFERENCE function to output the set of keys whose forecast error, $\mathbf{e}_t = y_t - \hat{y}_t$, is larger than some given threshold for detection.

4.4.2. EWMA

We denote $\mathbf{M}_0(t)$ as the current #SYN-#SYN/ACK at the time interval t, and $\mathbf{M}_f(t)$ as the forecasted #SYN-#SYN/ACK at the time interval t, we have

$$\mathbf{M}_f(t) = \begin{cases} \alpha\mathbf{M}_0(t-1) + (1-\alpha)\mathbf{M}_f(t-1) & t > 2, \\ \mathbf{M}_0(1) & t = 2. \end{cases} \quad (1)$$

In HiFIND, we use COMBINE $(\alpha, S_0(t-1), 1-\alpha, S_f(t-1))$ for implementation, where $S_0(t)$ and $S_f(t)$ are the current sketch and the forecast sketch at the time interval t, respectively. The difference between the forecasted value and the actual value is then used for detection. That is, we use the INFERENCE function to output the set of keys whose forecast error, $\mathbf{e}_t = \mathbf{M}_0(t) - \mathbf{M}_f(t)$, is larger than some given threshold.

4.5. Reducing false positives for SYN flooding detection

While port scans are relatively easy to identify with the algorithms above, there can be a number of factors other than SYN flooding that may cause a particular destination IP and port with a large number of unacknowledged SYNs. For instance, flash crowds, network/server congestions/failures, and even polluted or outdated DNS entries may cause a large number of SYNs without SYN/ACK at the edge routers. These may cause high false positives in our detection scheme. For the flash crowds, it is difficult, if not impossible, to differentiate it from the SYN flooding attacks without payload information as discussed in [52]. Thus we aim at reducing the false positives caused by the other two behaviors listed above.

4.5.1. Filters to reduce false positives caused by bursty network/server congestions/failures

We believe a real SYN flooding should deny service to a certain degree, e.g., the number of successful connections

Table 4
Different attacks are detected in different reversible sketches.

Attack types	{DIP, Dport}	{SIP, DIP}	{SIP, Dport}
SYN flooding	Yes	Yes	Yes
Vertical scans	No	Yes	No
Horizontal scans	No	No	Yes

Table 5
The uniqueness of different types of keys.

Types of Keys	SYN flooding	Horizontal scan	Vertical scan	Uniqueness
{SIP, Dport}	Non-spoofed	Yes	No	1.5
{DIP, Dport}	Yes	No	No	1
{SIP, DIP}	Non-spoofed	No	Yes	1.5
{SIP}	Non-spoofed	Yes	Yes	2.5
{DIP}	Yes	No	Yes	2
{Dport}	Yes	Yes	No	2

is less than the unsuccessful ones, in other words, $\frac{\#SYN-\#SYN/ACK}{\#SYN/ACK} > 1$. Moreover, the typical TCP SYN flooding attacks observed in the Internet last for a certain period, such as 10 min [29,14], while most network congestions/failures are bursty, which may only last a few seconds or minutes. Based on these two observations, we introduce two filters to reduce the false positives of SYN flooding detection. The first one is $\frac{\#SYN-\#SYN/ACK}{\#SYN/ACK} > 1$, and the second one is $lifetime > Threshold_{life}$ (e.g., 10 min). The system administrators can adjust their threshold to get the trade-off between sensitivity and the number of alerts.

To implement the first filter, we add an original sketch OS({DIP, Dport}, SYN) to record the number of SYNs for each {DIP, Dport} at each time interval. When a SYN flooding suspect is detected, we estimate the number of its SYNs from the original sketch. Then, based on $\frac{\#SYN-\#SYN/ACK}{\#SYN/ACK} > 1 \iff \frac{\#SYN-\#SYN/ACK}{\#SYN} > \frac{1}{2}$, we use the equivalent condition to decide whether the attack should be filtered.

To implement the second filter, we put any SYN flooding suspect key into a suspect array of counters. Then, for each subsequent interval, we query the #SYN and #SYN-#SYN/ACK value from the related sketches, and increment the counter if the above inequality holds.

4.5.2. Filter to reduce the false positives caused by misconfigurations or related problems

Usually, a SYN flooding attack will be launched towards some server which provides certain (popular) services. However, we find that some victims of detected SYN flooding are non-existing IPs or some hosts without any services on the ports being flooded. Moreover, the size of packets is small, so they are not bandwidth consumption attacks. Such anomalies look more like mistakes than real attacks. These may be caused by misconfigurations. For example, some administrators may misconfigure the DNS entry for a popular server as a non-existing IP or some IP which does not run the particular service. Some automatic connection software may then keep trying to connect to that server. Another possibility is that a DNS entry is updated, but the TTLs for the DNS entries are set/misconfigured to be very long, so the clients may still try to connect to the old IP which may not have the service open anymore.

When evaluating HiFIND with the LBL and NU data as discussed in Section 6, we do find several such scenarios. For instance, in the LBL data, we find there is a SYN flooding like anomaly towards a certain IP with large #SYN-#SYN/ACK. The anomaly lasts for 3 h, but there are no good connections towards that IP during the day, before or after the anomaly. Thus, this is unlikely to be a real SYN flooding attack. Based on such observations, we can design another false positive reduction heuristic by filtering those anomalies for which the victim does not have any response to SYN requests during the recent 24 h period. Due to the linearity of sketches, we can add up the sketches in the past 24 h, and then estimate the number of SYN/ACK for the suspect {DIP, Dport}. If the number is less than a given threshold, e.g. 100, it will not be considered as an attack. Note that we already have sketches to record #SYN and #SYN-#SYN/ACK, so no extra sketches are needed to record SYN/ACK for this monitoring phase.

4.6. DoS resilience analysis

As we mentioned before the TRW is vulnerable to spoofed IP attack. The attacker can send a lot of SYN packets with spoofed source IP address and random destination IP address (within the edge network). This will cause the TRW to use too much memory and possibly crash, since the TRW needs to keep states for each source IP address. The TRW-AC uses an AC table with fixed memory to improve the scalability of the TRW, so the attack cannot crash the TRW-AC. However, as mention in the TRW-AC paper [22] itself, the more source spoofed packets, the more collisions occur in the AC table. Hence, this makes the TRW-AC suffer high false negatives. For example, in their paper, they use the connection cache size of 1 million entries, and the $D_{conn} = 10$ min.[2] If the attacker periodically sends 1 million IP spoofed packets in 10 min (1667 packets/s, 533 kb/s for 40 bytes SYN packets), he can fully pollute the connection cache with half-open connections so that none of the real attack can be detected.

The HiFIND system is resilient to such attacks. If an attacker sends the source spoofed SYN packets to a fixed destination, our system will treat this as a SYN Flooding attack to the particular IP address. If the attacker sends the source spoofed packets to random destinations within the edge network, the SYN count will be evenly distributed in the buckets of each of the hash tables in sketches. Even if there is a real attack, the SYN count for that attack is still sufficiently significant to be detected.

One possible attack is to introduce false positives or false negatives by creating collisions in the hash tables of sketches. To create collisions, the attacker needs to reverse engineer all the hash functions of sketches and search exhaustively offline. Section 4.6.1 shows such reverse engineering is impossible unless they can compromise the HiFIND system and obtain the intermediate execution results. Even if they can somehow get the hash functions, or they exhaustively try all attack keys (e.g., IPs) for an online search assuming that they can get the detection results, the probability of having a collision in sketch is very small (see Section 4.6.2). Finally, even if the attacker can somehow create collisions for sketches, when we monitor both ingress and egress traffic for detection, they can at most create some false positives, but not false negatives

[2] The TRW-AC uses a background process to remove any connection idle for more than D_{conn} min.

(Section 4.6.3). Overall, it is extremely difficult to attack the HiFIND system.

4.6.1. Reverse engineering of hash functions is very difficult

In general, if attackers know the input to a hash function and the hashed value output, it is possible to reverse engineer the hash function. However, in HiFIND, although attackers may possibly know the detection result, it is very difficult for them to infer the hash functions because the direct hash output of each hash function is unknown and is impossible to infer.

Essentially, sketches are composed of multiple hash tables, each with different hash functions. There are some parameters determining the hash functions. Given any key, the estimated value from the sketch is the median of the estimated value from all the hash tables. Thus the estimated value does not have any correlation to the buckets (i.e., the direct hash output) of each hash table in which the value of such key is stored. Thus we cannot infer the output value of hash functions, let alone the hash function themselves. So, unless attackers can observe the internal states of algorithm execution, it is impossible for them to reverse engineer the hash functions used in sketches.

Moreover, the parameters are all generated randomly when the HiFIND system starts, and can be reset and thus change all the hash functions when we restart the system.

4.6.2. The possibility of finding collisions through exhaustive search is very low

In this subsection, we show that regardless of online or offline search, the probability of finding collisions in sketches is very small. Typically we use $H = 6$ hash functions in both reversible sketches and original sketches. To make a collision, the attacker needs to make two keys collide in at least $H - r$ hash functions, where typically $r = 1$. Given each hash function has at least 2^{12} buckets, and all the hash functions are independent, the probability of a collision of two random keys will be $P = \sum_{i=0}^{r} \binom{H}{H-i} S^{H-i}(1-S)^i$, where $S = \frac{1}{2^{12}}$. For $H = 6, r = 1$, the possibility is $P = 5.2 \times 10^{-18}$.

4.6.3. Attacks are limited even with collisions

In this subsection we analyze what attackers can do, even if they can create collisions. In the HiFIND system, we monitor both ingress and egress traffic for detection. Unless the attacker has already compromised some internal machines in the victim network, the attacker can only control one-way traffic. For example, the attacker can only send more SYN packets but cannot make HiFIND receive more SYN/ACK packets which have to come from the victim network. That is, he can only increase the count of `#SYN-#SYN/ACK` in sketches but cannot decrease it. Thus, he can only create false positives by using collisions, but cannot create false negatives to hide attacks.

As mentioned in Section 3.2, for a reversible sketch, we always use another original sketch for false positive reduction. The attacker must create collisions for both sketches to create a false positive, which is even harder.

5. Intrusion classification with two-dimensional sketch

It is crucial to distinguish different types of attack to take the most effective mitigation scheme. However, one major challenge for intrusion detection is that the traffic anomalies are often multidimensional i.e., they can only be identified when we examine traffic with specific combinations of IP addresses, port numbers, and protocols. For example, if the port distribution of a particular attack is unknown, it becomes very hard to distinguish non-IP-Spoofing SYN flooding attacks from vertical scans because both of them will exhibit a single (or a small number) of source IPs sending a large number of un-responded SYN packets to the destination IP. The key difference lies in whether the attacker sends to a small number, e.g., one or two, of the ports (SYN flooding) or many different ports (vertical scan) on the destination. In other words, there is a bi-modal distribution for the number of unique ports visited when there are a large number of un-responded SYN packets from one source to one destination. One mode corresponds to the SYN flooding, and the other, vertical scans. This bi-modal assumption is verified with real network traffic in Section 5.1. Thus, it is essential to know the port distribution, given a specific source IP and destination IP pair {`SIP`, `DIP`}, to distinguish between these two attacks. However, existing sketches are all of single dimension. To address this challenge, we design a novel two-dimensional (2D) *k*-ary sketch and apply it for intrusion classification in Section 5.2.

5.1. Verification of the bi-modal distribution for SYN flooding vs. port scans

We tested the bi-modal assumption with one-month edge router traffic from Northwestern University (NU) and Fermi Lab. With samples of two-hour NU data and one-day Fermi data, Fig. 6 shows the distribution of the number of attacks with respect to the number of unique ports visited when there is more than 50 un-responded SYNs in a 1-minute interval from one source to one destination. There is a clear bi-modal distribution to distinguish SYN flooding attacks from vertical scans. The former usually have the number of ports visited at less than three. Similarly, there is also a bi-modal distribution between SYN flooding attacks and horizontal scans. SYN flooding attacks have a very small number of destination IPs, while horizontal scans have a much larger amount. Fig. 7 shows the distribution of the number of attacks with respect to the number of unique destination IP addresses visited when there are more than 100 un-responded SYNs in a 1-min interval from one source IP to one destination port. Note that in the graph for NU, there is only one mode because there is no SYN flooding attack in the 2 h NU traffic.

Theoretically, in the worst-case, the attackers can create an attack with a port distribution just in between of the typical distributions of vertical scans and SYN flooding. In such an attack, the attackers send large number of SYN packets to multiple ports. The attack can be viewed as vertical scans or SYN flooding or even both. Our approach will fail to differentiate such extreme cases, such cases will be

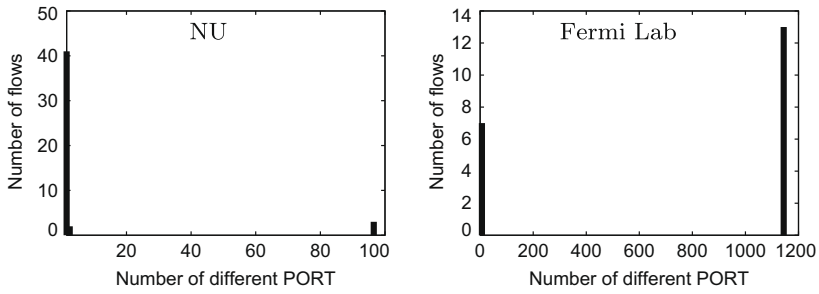

Fig. 6. The distribution of the number of attacks with respect to the number of unique ports visited when there are more than 50 un-responded SYNs in 1-min interval between one {SIP, DIP} pair.

very hard for manual analysis as well. Although we might not be able to differentiate such cases, our system still can detect such attacks and label them as "undecided". In practice, we believe such attacks are unlikely to happen, since they cannot help attackers to fully avoid detection and make the intended attacks less efficient.

5.2. Two-dimensional sketch design and intrusion classification

For the 2D k-ary sketch, instead of using H independent one-dimensional hash tables, we use H independent 2D hash tables (matrices), as shown in Fig. 8. Let K_x and K_y denote the number of buckets for each dimension, respectively. For each 2D hash matrix, we hash two groups of fields into it. Consider the previous example of separating SYN flooding attacks from vertical scans. The x dimension represents the {SIP, DIP}, and the y dimension corresponds to Dport. For each packet, we locate its corresponding entry in the matrix by two independent hash mappings as shown in Fig. 9, and update the bucket in the same manner as for reversible sketches in Section 4.3. Similarly, we can update all H matrices for data stream recording.

In the detection stage, after finding an attack by using reversible sketch or any other method (i.e., the {SIP, DIP} is known), we can use the column of buckets in the hash matrix selected by the {SIP, DIP} to infer the distribution of Dport and pinpoint the type of attack, e.g., a SYN flooding or a vertical scan. The algorithm is as follows. For each 2D hash matrix, the {SIP, DIP} pair selects a column of buckets. We define B to be the total sum of all buckets in the column. We then obtain the sum S_p of the top p buckets with the largest values (e.g., 5 out of 64). If $S_p > \phi \times B$ for some $\phi < 1$, e.g., 0.8., then we regard it as a SYN flooding. If the majority of the H hash matrices of the 2D sketch

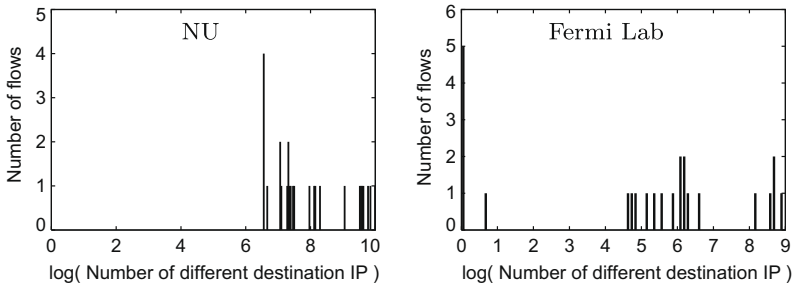

Fig. 7. The distribution of the number of attacks with respect to the number of unique destination IPs visited when there are more than 100 un-responded SYNs in 1-min interval with one SIP, Dport pair.

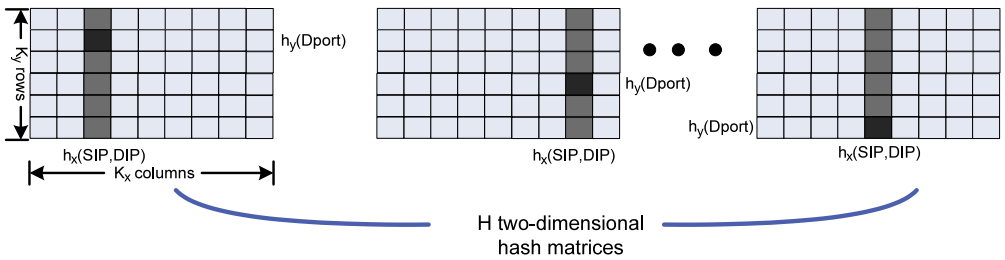

Fig. 8. Diagram of the two-dimensional k-ary sketch.

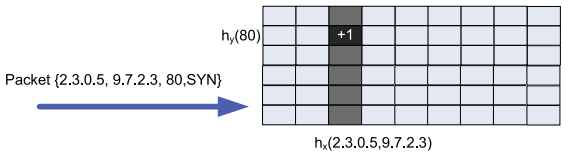

Fig. 9. An example of UPDATE operation for two-dimensional sketch.

imply it is a SYN flooding, we conclude it is a SYN flooding attack; otherwise we conclude it is a vertical scan. Similarly, we can differentiate horizontal scans from the SYN flooding attacks.

5.3. Accuracy analysis

An important issue with using the bi-modal distribution to separate SYN flooding attacks versus vertical scan attacks is the possibility of misclassification. The concern is that a vertical scan attack could be hashed among the K_y buckets of a given column in a hash matrix such that a large number of attacked ports hash to just a single bucket, while the remaining attacked ports are spread out among many different buckets. Such a clumping would cause the detection algorithm to report a SYN flooding attack rather than the correct vertical scan attack (note that the reverse problem of misclassification should not be an issue). Alternatively, it is possible that a key of a SYN flooding attack and a key of vertical scan attack happen to hash to the same column. In such a case, since the hash table can only return SYN flooding or vertical scan for the given column, it will be wrong. To address these issues, we first bound the probability of the first type of misclassification assuming that the given column has been hit by only one type of attack. We will then bound the likelihood of such a column being hit by two separate attacks to get a final bound on the probability of a misclassification.

To bound the probability of the first type of misclassification, we use the *Generalized Increment-Decrement Counter Model* [37]. If one SYN packet hashes to the bucket (counter), the counter increases by one; if one SYN/ACK packet hashes to the counter, the counter decreases by one. At the end of a sufficiently large amount of time T called a *measurement interval*, e.g., 1 min, the counter should accumulate all the packets hashed to it to get the outcome value N. Now, for all normal traffic, there will be an equal number of SYN's and SYN/ACK's, thus contributing a total of zero to the hash matrix.[3] Now, consider a column of the hash matrix that is hashed to by one single vertical scan key. We will assume that a vertical scan sends exactly 1 SYN to B different ports.

Lemma 1. *Given a column of a hash matrix whose total SYN value comes from a single vertical scan key, the probability of misclassification is* $Pr[S_p > \phi B] \leqslant K_y e^{\frac{-2\delta^2}{B}}$, *where* $\delta = \frac{B}{p}\left(\phi - \frac{p}{K_y}\right)$.

[3] This is not always true, but given a large enough measurement window and high volume traffic, it is approximately true.

Proof. Let X_i be a random variable denoting the total traffic in bucket i of a given column. As each X_i is a binomial variable with mean $\frac{B}{K_y}$, from the Chernoff bound we get that $Pr\left[X_i > \frac{B}{K_y} + \delta\right] < e^{\frac{-2\delta^2}{B}}$. For the largest X_i, X_{max}, we thus get $Pr\left[X_{max} > \frac{B}{K_y} + \delta\right] < K_y e^{\frac{-2\delta^2}{B}}$. Further, we know that if the sum of the top p buckets S_p exceeds $p\left(\frac{B}{K_y} + \delta\right)$, then by the pigeon hole principle at least one X_i must exceed $\frac{B}{K_y} + \delta$, which yields $Pr\left[S_p > \frac{B \cdot p}{K_y} + p\delta\right] < K_y e^{\frac{-2\delta^2}{B}}$. By setting $\delta = \frac{B}{p}\left(\phi - \frac{p}{K_y}\right)$ we get the desired inequality. □

Having bounded the possibility of misclassification assuming only one attack for a given bucket, we now consider the probability of actually getting this scenario. Let f and v denote the total number of SYN flood attacks and vertical scan attacks within a measurement interval T, respectively. First, we will bound the probability of misclassifying a vertical scan. For a given key that constitutes a vertical scan, the key will be correctly categorized in any given hash matrix according to Lemma 1 unless a SYN flood attack key j also hashes to the same bucket (multiple vertical scans in a single bucket will still register as a vertical scan). Combining the probability of such a collision with Lemma 1 yields the following lemma.

Lemma 2. *The probability that a given hash matrix h correctly classifies a vertical scan key k is* $P[correct_h] \geqslant S = \left(\frac{K_x - 1}{K_x}\right)^f (1 - error)$, *where* $error = K_y e^{\frac{-2\delta^2}{B}}$ *as stated in Lemma 1.*

Proof. The probability of one or more of the f SYN flood keys hashing to a single given bucket is $\left(\frac{K_x - 1}{K_x}\right)^f$. Combining this event with Lemma 1 yields the result. □

We now use this lemma to derive the probability of failure that takes the majority result of the H hash tables.

Theorem 3. *Given a key k of a vertical scan, the majority of the H hash matrices will classify k as a vertical scan attack with probability at least* $\sum_{r=\lfloor \frac{H}{2} \rfloor + 1}^{H} \binom{H}{r} S^r (1 - S)^{H-r}$ *where* $S = \left(\frac{K_x - 1}{K_x}\right)^f (1 - error)$ *where error is as stated in Lemma 1.*

Proof. The result follows from observing that the total number of correct hash matrices is a binomial variable. □

Theorem 4. *Given a key k of a SYN flooding, the majority of the H hash matrices will classify k as a SYN flooding attack with probability at least* $\sum_{r=\lfloor \frac{H}{2} \rfloor + 1}^{H} \binom{H}{r} S^r (1 - S)^{H-r}$ *where* $S = \left(\frac{K_x - 1}{K_x}\right)^{f+v}$.

Proof. The number of correct hash matrices is a binomial variable, so it is sufficient to show that the probability of successful classification for any given matrix is at least S. □

As an example of the bounds provided by Theorems 3 and 4, consider the following typical values. For $H = 5$, $p = 5, \phi = 0.8, K_x = 4096, K_y = 64$ (these configurations are also adopted in the evaluation of Section 6), $B = 200, f = 5$, and $v = 5$, we get that a vertical scan will be detected with probability at least 99.9956% and a SYN attack will be classified correctly with probability at least 99.99999%. Note that the larger B is, the larger these probabilities are. We can achieve similar misclassification bounds for reporting a horizontal scan as a SYN flooding and vice versa.

6. Evaluation

6.1. Evaluation methodology

In this section, we evaluate HiFIND with both simulation and on-site experiment.

- We use the router traffic traces collected at the Lawrence Berkeley National Laboratory (LBL) for simulation. The one-day trace consists of about 900M netflow records. Unfortunately, the sampling rate is unknown.
- We apply HiFIND for on-site detection at the Northwestern University (NU, which has several Class *B* networks) edge routers. The router exports netflow data continuously which is recorded with sketches of HiFIND on the fly. At the end of each minute, HiFIND detects intrusions online as shown in Fig. 3. We also record the complete netflow records for detection comparison. The one day experiment in May 2005 consists of 239M netflow records. The total traffic is 1.8TB. The average packet rate is 37 k/s and the peak packet rate is 79 k/s. The flows were constructed from packet sampling at a 1:1 rate.

Unless denoted otherwise, the default time interval for constructing the time series for detection is 1 min.

The data recording part of the HiFIND system consists of: (1) three reversible sketches (RS), one for {SIP, Dport}, one for {DIP, Dport}, and the other for {SIP, DIP}, (2) one original sketch (OS) for {DIP, Dport}, and (3) two 2D sketches for {SIP, Dport} × {DIP} and {SIP, DIP} × {Dport}. For all the RS and 2D sketches we update #SYN-#SYN/ACK as the value, and only for the OS, we use #SYN as the value.

The following parameters are chosen based on systematic study as in [20,42]. We adopt 6 stages for each RS and OS, and 5 stages for each 2D sketch in our system. We use 2^{12} buckets for each stage in 48-bit RS, 2^{16} buckets for each stage in the 64-bit RS, and 2^{14} buckets for all their verification sketches. 2^{14} buckets are applied for each stage in OS. We also use $2^{12} \times 64$ buckets for each stage of the 2D sketches. Therefore, the total memory is 13.2 MB.

Both NU and LBL have a large amount of traffic, so we set the detection threshold to be one scan per second for both horizontal and vertical scans, and one un-responded SYN packet per second for SYN flooding attacks. The thresholds can be adjusted by network administrators.

The evaluation metrics include *detection accuracy* (in terms of the true positive and false positive percentages), *execution speed, the amount of memory access per-packet*, and *the amount of memory used in the recording stage*. To evaluate the accuracy of the HiFIND system, we take the following steps. First, we solely evaluate the error introduced by the sketches, which is very small as shown in Section 6.2. Second, we compare the HiFIND system with two existing network intrusion detection approaches for both single-router detection and aggregated detection in Section 6.3 and find that HiFIND significantly outperforms others. Thirdly, we manually validate the attacks we detected in Section 6.4. Regarding online performance, in Section 6.5, we demonstrate the small memory consumption, small number of memory accesses per-packet and high-speed of the HiFIND system.

We implement both EWMA and non-seasonal Holt–Winter time series analysis algorithms in the HiFIND system. The two methods give very similar results for the one day NU data and the LBL data. We thus use EWMA as the default method to show all evaluation results.

6.2. Sketches highly accurate in recording traffic for detection

Table 6 shows the three phases of our detection results. We first detect attacks using reversible sketches with algorithms described in Section 4.3. The results are shown as "Raw results" ("Phase 1") in Table 6. 2D sketches reduce the false positives for port scans introduced by SYN flooding attacks ("Phase 2") of Table 6. The heuristics in Section 4.5 reduce false positives of SYN flooding attacks ("Phase 3").

To evaluate the errors introduced by sketches, we compare the results obtained from the same detection algorithm but with two different types of traffic recording: (1) sketches; (2) accurate flow table to hold per-flow information (we call it non-sketch method). We find that we detect exactly the same attacks for the two configurations with very different amounts of memory (see memory consumption discussion in Section 6.5). This shows sketches are highly accurate in recording the traffic for detection.

6.3. HiFIND outperforms other existing network IDSes

6.3.1. Detection over a single-router

We compare the HiFIND with other state-of-the-art network intrusion detection approaches as introduced in Section 2: the TRW [21] for port scan detection and the CPM [13,14] for SYN flooding detection.

Table 6
Detection results under three phases.

Traces	Attack type	Phase 1: Raw results	False positive reduction	
			Phase 2: Port scan	Phase 3: Flooding
NU	SYN flooding	157	157	32
	Hscan	988	936	936
	Vscan	73	19	19
LBL	SYN flooding	35	35	0
	Hscan	736	699	699
	Vscan	40	1	1

Table 7
Horizontal scans detection comparison of HiFIND and TRW aggregated by source IP.

Data	TRW	HiFIND	Overlap number
NU	497	512	488
LBL	695	699	692

Table 8
TCP SYN flooding detection comparison of HiFIND and CPM.

Data	CPM	HiFIND	Overlap number
NU	1422	1427	1422
LBL	1426	0	0

For TRW experiments, we choose similar parameters as those mentioned in their paper, $P_D = 0.99, P_F = 0.01$, $\theta_1 = 0.2$ and $\theta_0 = 0.8$. We then apply the TRW method on NU data and LBL data with a 1 min time interval, and the same threshold as we set in our system previously. After obtaining the TRW results, we filter out the repeated alerts from the original reports and get the final result, as well as what we do in our HiFIND system. Table 7 shows the comparison results of our methods with TRW for horizontal scan detection. In TRW, the horizontal scans from the same source IP but to different ports are counted as only one attack. Thus, we aggregate the horizontal scans detected by HiFIND with the source IP address. We observe that the scans detected by these two methods have very good overlap, except for a few special cases. There are a small number of horizontal scans detected by HiFIND but not TRW due to the following reasons. Some attacks have both successful connection attempts and unsuccessful connection attempts. TRW gives each attempt different weights and multiplies them together for detection. If the product value is higher than a certain threshold, it will be identified as a port scan. In some cases, although the number of unsuccessful connections is fairly large, more than 500, they still have several successful connections, such as 100–150. TRW is not sensitive to this kind of abnormal behavior. In fact, we should consider these behaviors as abnormal ones, or at least suspicious ones.

Meanwhile, there are a very small number of scans detected by TRW but not HiFIND. They are caused by some scenarios as follows. The attackers scan multiple hosts, and for different hosts, they scan at different ports. When aggregated by either {SIP, DIP} or {SIP, Dport}, the number of scans is relatively small and less than our threshold. But TRW aggregates the scan by source IP and counts the number of unique destination IP's for detection, and thus will detect such behavior. We are not aware of any worms using such scans for propagation. It seems more like a combination of multiple small scans. Since the number of scans to each port and the number of scans to each host are small, the overall effect of such scans is still relatively small, i.e., not a major attack. It is part of our future work to further investigate such problems.

From the results above, we find the TRW and the HiFIND have a good overlap, however, under the worst-case the HiFIND system is resilient to DoS attacks, while the TRW will run out of memory. The detailed analysis is described in Section 6.5.1. Moreover, in practice HiFIND is hardware implementable for online data recording, so that it can be deployed on high-speed networks.

Next, we compared our method with CPM for SYN flooding attack detection. The results are shown in Table 8. Note that CPM detection is based on overall traffic, but not on each flow, so it can only report for each interval whether there is an attack or not. For a fair comparison, we also count the number of intervals that SYN flooding attacks detected by HiFIND span on (not the real number of attacks) and include them in Table 8.

In the LBL traces, there is no SYN flooding, but a very large number of scans. As discussed before, CPM cannot differentiate them, so it treats all the scans as SYN flooding producing a large number of false positives. On the other hand, CPM and HiFIND have very similar results for the NU data because the port scans are mixed with SYN flooding for each interval, and thus the port scans do not cause any false positives for CPM. Meanwhile, there is a small number of intervals in which SYN flooding is buried in the rest of the normal traffic, which have large numbers of normal SYN and SYN/ACK packets from the traffic of other hosts, and thus for the total traffic $\frac{SYN-SYN/ACK}{SYN/ACK} < 1$, meaning CPM cannot detect them.

6.3.2. Aggregated detection over multiple routers

In this section, we consider the network topology of Fig. 4 discussed in Section 4 and evaluate the performance of HiFIND and TRW under such scenarios. To simulate asymmetric routing and multi-path routing caused by per-packet load balancing on routers, we split the packet-level traffic from a Northwestern University edge router into three routers randomly, for both inbound and outbound packets. For each packet, we randomly select an edge router to deliver, i.e., for any single connection, the incoming SYN packet and the outgoing SYN/ACK packet have 2/3 probability to go through different routers.

For HiFIND, we obtain the same results as those when the traffic goes through the same router, i.e., the results in Section 6.3.1. In comparison, we apply TRW to the data on each router for detection. Thus we get three sets of results. We put them together and remove the redundant alerts as the final results. In the previous single-router experiments, we set the alert threshold as 60 for TRW, and find 497 horizontal scans. For the multi-path experiments, we run two experiments with different thresholds. We first keep the same threshold 60 in each router, and we obtain 461 alerts in the final results. Compared to one-router experiments of TRW, there are 36 false negatives in this result. Then, considering that each router may only get 1/3 of the attack and response traffic, we reduce the threshold to 20 correspondingly. In this case, TRW reports 578 horizontal scans, which has 81 false positives and no false negatives.

6.4. Detected intrusions successfully validated

In the previous section, we compare the HiFIND with other statistical detection methods. But, it does not show

whether the attacks detected are the real ones. In this section we manually examine a certain number of attacks for validation.

6.4.1. SYN flooding

We validate our SYN flooding detection results with backscatter [29] because the Anderson–Darling A2 uniformity test in backscatter is a very strong one and can hardly reach the significance level 0.05 if they are not uniformly distributed. However, backscatter has one limitation: it only checks the reply traffic from the victim, and assumes that each attack packet is replied to the victim, which is not valid in many cases (observed from our netflow data as well). Furthermore, in our evaluation, we only detect incoming attacks. Thus, we tweak backscatter a bit, to check the incoming SYN traffic, and test the uniform distribution of the source IPs if they are all towards the same destination IP and the total number of such source IPs are large enough. Among the 32 SYN floodings detected, there are 21 matched with backscatter results. For the other 11 attacks, three have uniformity values very close to the threshold, Anderson–Darling A2 test with significance level 0.05 [29]. Another three have IP's spoofed in some regular manner. For instance, one has the IP address spoofed as: 1.0.0.*, 1.0.1.*, 1.0.255.*, 2.0.0.*, etc.. But the distribution of such addresses does not satisfy the uniformity test. The remaining five attacks are hard to validate with only packet headers. But nowadays many DoS attacks are launched with botnet using un-spoofed IP's because ISPs tend to drop randomly spoofed packets through ingress filtering.

6.4.2. Horizontal scans

We manually validate horizontal scans, in particular, the top 10 and bottom 10 attacks in terms of their change difference. Table 9 shows the top 10 horizontal scans for the NU experiment, whose #SYN–#SYN/ACK change count ranges from 24,000 to 60,000. Table 10 shows the bottom 10 horizontal scans, whose #SYN–#SYN/ACK change count ranges from 60 to 64. Table 11 and Table 12 demonstrate the top 10 and bottom 10 horizontal scans for the LBL trace. Through this evaluation table, we find that they are all possible attacks, including MSBlast worm (Nachi), SQLSnake worms, SSH scans, etc.. There are even some unknown worm scans detected by us, and also confirmed in the Dshield [53], the largest worldwide intrusion log repository.

Table 9
Top 10 of Horizontal scans in NU experiment.

Anonymized SIP	Dport	#Unique DIP	Cause
204.10.110.38	1433	56275	SQLSnake scan
5.4.247.103	1433	54788	SQLSnake scan
15.38.124.150	1433	46586	SQLSnake scan
5.64.27.226	1433	45981	SQLSnake scan
109.132.101.199	22	45014	Scan SSH
95.30.62.202	3306	25964	MySQL Bot scans
162.39.147.51	6101	24741	Unknown scan
15.192.50.153	4899	23687	Rahack worm
15.82.184.106	4899	19794	Rahack worm
5.55.96.69	1433	19217	SQLSnake scan

Table 10
Bottom 10 of horizontal scan in NU experiment.

Anonymized SIP	Dport	#Unique DIP	Cause
98.198.251.168	135	64	Nachi or MSBlast worm
3.66.52.227	445	64	Sasser and Korgo worm
20.128.114.246	445	64	Sasser and Korgo worm
2.0.28.90	139	64	NetBIOS scan
91.115.92.212	445	64	Sasser and Korgo worm
98.198.0.101	135	64	Nachi or MSBlast worm
162.8.171.169	139	62	NetBIOS scan
82.203.230.86	135	62	Nachi or MSBlast worm
165.5.42.10	5554	62	Sasser worm
189.75.216.218	139	62	NetBIOS scan

Table 11
Top 10 Horizontal scans in the LBL trace.

Anonymized SIP	Dport	#unique DIP	Cause
352841	1433	16014	SQLSnake scan
251293	21	10360	FTP scan
199707	80	7893	HTTP scan
295205	80	7536	HTTP scan
346376	80	6980	HTTP scan
347333	80	6935	HTTP scan
312475	21	5892	FTP scan
447394	1433	5875	SQLSnake scan
285091	4000	5775	SkyDance worm
227176	554	5660	RTSCP scan

6.4.3. Vertical scans

We also manually validate vertical scans for both the LBL trace and the NU experiment. In the LBL trace, we found one vertical scan. It scanned some well-known service ports, such as HTTPS(81), HTTP-Proxy(8000,8001,8081).

In the NU experiment, we found in total 19 vertical scans. We manually checked the top 5 and bottom 5 vertical scans in terms of the ports they scanned. We found the vertical scans are mostly interested in the well known service ports and Trojan/BackDoor ports, for the well know services ports, most of them scanned: HTTP(80), FTP(21), RTSP(554), Cache(3128) etc.. Moreover, most attacks are also interested in Trojan/BackDoor ports, for example, WinHole(808,1082), Slapper(1978), SubSeven 2.1(7000), Tini(7777), OpwinTRojan(10,000), etc.. All these characteristics drive us to believe that these vertical scans are real scans.

6.5. Evaluation results for online performance constraints

6.5.1. Small memory consumption

It is very important to have small memory consumption for online traffic recording over high-speed links, to make use of the fast SRAM and for potential implementation in specialized hardware e.g., FPGA or ASIC. In our experiments, we only use a total memory of 13.2 MB for traffic recording. Note that such settings work well for a large range of link speeds, as we tried on different network traces. Sketch can report heavy changes and heavy hitters with the statistical accuracy bounds given in [42,19].

Table 12
Bottom 10 Horizontal scans in the LBL trace.

Anonymized SIP	Dport	#Unique DIP	Cause
191136	135	67	Nachi or MSBlast worm
4698	135	67	Nachi or MSBlast worm
432940	135	66	Nachi or MSBlast worm
219400	135	66	Nachi or MSBlast worm
253855	135	66	Nachi or MSBlast worm
445247	135	66	Nachi or MSBlast worm
81248	135	66	Nachi or MSBlast worm
218603	135	64	Nachi or MSBlast worm
98802	135	64	Nachi or MSBlast worm
208650	139	26	NetBIOS scan

On the other hand, if hash tables are used to record every flow, much larger memory is required as shown in Table 13. We consider the worst-case traffic of all-40-byte-packet streams with 100% utilization of the link capacity. There is a spoofed SYN flooding attack with a different source IP (and maybe even different destination IP) for each packet. For the method without sketch, it needs at least three hash tables corresponding to the three reversible sketches in our detection methods. For each packet, at least a new entry will be added to both the {SIP, DIP} table and the {SIP, Dport} table. Every entry of the hash table needs 6–8 bytes to store the key (e.g., SIP and DIP) and another 4 bytes to store the value. For example, when the bandwidth is 2.5 Gbps, for 1 s, the total memory consumption is $(2.5G/(8 \times 40)) \times (12 + 10) = 171.875$ MB. For TRW, for each {SIP, DIP} pair, it maintains a likelihood ratio which is updated based on observed SYN and SYN/ACK. Thus, for the example above, the total memory consumption is $(2.5G/(8 \times 40)) \times 12 = 93.75$ MB. Thus, both of the methods will run out of memory very quickly in these scenarios.

6.5.2. Small memory access per-packet for online monitoring

There are 15 memory accesses per-packet for 48 bit reversible sketches and 16 per-packet for 64-bit reversible sketches (see [20] for details). For each two-dimensional sketch, we only need 5 memory accesses per-packet, one for each 2D hash matrix. Thus, when recording these sketches in parallel or in pipeline, the HiFIND system has a very small number of memory accesses per-packet and is capable of online monitoring.

6.5.3. Traffic monitoring and intrusion detection with high speeds

The HiFIND system is composed of the three reversible sketches and two 2D sketches. The speed of 2D sketches is much faster than that of the reversible sketches. Thus, the speed is dominated by the latter. For a real HiFIND system, we will implement it in hardware so that multiple sketches can be updated in parallel. With our prototype single FPGA board implementation, we are able to sustain 16.2 Gbps throughput for recording all-40-byte packet streams (the worst-case) with a reversible sketch.

We can also use multi-processors to record multiple sketches simultaneously in software. With a Pentium Xeon 3.2 GHz machine with normal DRAM memory, we record 239M items (1.8TB, 1-day NUIT data) with one reversible sketch in 20.6 s, i.e., 11M insertions/s. For the worst-case scenario with all-40-byte packets, this translates to around 3.7 Gbps. These results are obtained from code that is not fully optimized and from a machine that is not dedicated to this process. If we update the three reversible sketches serially, we can still archive 3.8M insertions/s.

For the on-site NU experiments covering a total of 1430 min, the HiFIND system used 0.34 s on average to perform detection for each one-minute interval, and the standard deviation is 0.64 s. The maximum detection time (for which the interval contains the largest number of attacks) is 12.91 s, which is still far less than 1 min.

In order to show the scalability of HiFIND, we further do some stress experiments. We compress the NU data by the factor of 60, and detect the top 100 anomalies in each interval. The HiFIND system used 35.61 s on average in detection for each interval. The maximum detection time is 46.90 s.

7. Potential limitations of the HiFIND system

There are roughly two types of stealthy attacks: small rate attacks and slow ramping attacks. Depending on the detection threshold, HiFIND may not be very sensitive to stealthy scans for small rate attacks. On the other hand, small rate attacks in general are not very interesting for high-speed network gateways/routers because a low detection threshold tends to produce large volumes of scan alerts and will overwhelm network administrators. Such attacks usually also have limited effects. To deal with them, we can potentially combine the HiFIND system with existing network and host-based detection schemes, like TRW which is more sensitive to detect stealthy scans. We deploy the HiFIND system at edge routers of an edge network to detect most serious attacks while for some subnets or servers which require more secure protection, we apply TRW to detect stealthy scans.

For the "slow ramping" attack, attackers gradually increase the attack rate so that no significant request pattern changes are exhibited. Here we can aggregate the traffic at a coarser time interval, e.g. hourly, daily, so the traffic

Table 13
Memory comparison (bytes).

Methods	2.5 Gbps			10 Gbps		
	1 s	1 min	5 min	1 s	1 min	5 min
HiFIND with sketch	13.2M			13.2M		
HiFIND with complete info	171.875M	10.3G	51.6G	687.5M	41.25G	206G
TRW	93.75M	5.63G	28G	375M	22.5G	112.5G

changes become obvious and will be detected. The compact storage size and the linearity of sketch fully support such an aggregated detection scheme.

8. Conclusion

It is crucial to detect the outburst of global-scale attacks at high-speed routers/gateways. In this paper, we propose, implement and evaluate a DoS resilient High-speed Flow-level Intrusion Detection system, HiFIND, leveraging recent data streaming techniques such as reversible sketches. We analyze the TCP/IP headers and select an optimal small set of metrics for monitoring and detection. In addition, we design efficient 2D sketches to distinguish different types of attacks for effective mitigation. We further aggregate the compact sketches recorded over multiple edge routers to deal with the emerging asymmetric routing and multi-path routing enabled by the per-packet load balancing of routers. Experiments with several router traces show that HiFIND is highly accurate, efficient, uses very small memory, and can effectively detect multiple types of attacks simultaneously.

References

[1] T. Ryutov, C. Neuman, D. Kim, L. Zhou, Integrated access control and intrusion detection for web servers, IEEE Transactions on Parallel and Distributed System 14 (9) (2003) 841–850.
[2] S. Hofmeyr, S. Forrest, Intrusion detection using sequences of system calls, Journal of Computer Security 6 (1998) 151–180.
[3] V. Paxson, Bro: a system for detecting network intruders in real-time, Computer Networks 31 (23–24) (1999) 2435–2463.
[4] M. Roesch, Snort: the lightweight network intrusion detection system, 2001. <http://www.snort.org/>.
[5] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver, The spread of the Sapphire/Slammer worm, 2003. <http://www.caida.org>.
[6] S. Staniford, V. Paxson, N. Weaver, How to own the Internet in your spare time, in: Proceedings of the 11th USENIX Security Symposium, 2002.
[7] S. Staniford, D. Moore, V. Paxson, N. Weaver, The top speed of flash worms, in: Proceedings of the ACM CCS WORM Workshop, 2004.
[8] N. Weaver, V. Paxson, S. Staniford, R. Cunningham, Large scale malicious code: a research agenda, Tech. Rep. DARPA-sponsored report, 2003.
[9] D. Moore, C. Shannon, G.M. Voelker, S. Savage, Internet quarantine: requirements for containing self-propagating code, in: Proceedings of the IEEE Infocom, 2003.
[10] S. Sikka, G. Varghese, Memory-efficient state lookups with fast updates, in: Proceedings of the ACM SIGCOMM, 2000.
[11] G. Cormode, S. Muthukrishnan, What's new: finding significant differences in network data streams, in: Proceedings of the IEEE Infocom, 2004.
[12] C. Estan, G. Varghese, New directions in traffic measurement and accounting, in: Proceedings of the ACM SIGCOMM, 2002.
[13] H. Wang, D. Zhang, K.G. Shin, Detecting SYN flooding attacks, in: Proceedings of the IEEE INFOCOM, 2002.
[14] H. Wang, D. Zhang, K.G. Shin, Change-point monitoring for detection of DoS attacks, IEEE Transactions on Dependable and Secure Computing 1 (4) (2004).
[15] Checkpoint Software Technologies, TCP Flooding Attack and Firewall-1 SYNDefender. <http://checkpoint.com/press/1996/synattack.html>.
[16] Cisco Inc., Per-Packet Load Balancing, 2003. <http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s21/pplb.pdf>.
[17] Cisco Inc., Load balancing with Cisco Express Forwarding, 2003. <http://www.cisco.com/en/US/products/hw/modules/ps2033/prod_technical_reference09186a00800afeb7.html>.
[18] Y. Gao, Z. Li, Y. Chen, A dos resilient flow-level intrusion detection approach for high-speed networks, in: The International Conference on Distributed Computing Systems, 2006.
[19] R. Schweller, A. Gupta, E. Parsons, Y. Chen, Reversible sketches for efficient and accurate change detection over network data streams, in: ACM SIGCOMM/USENIX Internet Measurement Conference (IMC), 2004.
[20] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M. Kao, G. Memik, Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications, in: Proceedings of the IEEE Infocom, 2006.
[21] J. Jung, V. Paxson, A. Berger, H. Balakrishnan, Fast portscan detection using sequential hypothesis testing, in: Proceedings of the IEEE Symposium on Security and Privacy, 2004.
[22] N. Weaver, S. Staniford, V. Paxson, Very fast containment of scanning worms, in: USENIX Security Symposium, 2004.
[23] Arbor Networks, Intelligent Network Management with Peakflow Traffic. <http://www.arbornetworks.com/download.php>.
[24] Symantec Inc., Symantec ManHunt, <http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=156&EID=0>.
[25] N. Duffield, C. Lund, M. Thorup, Properties and prediction of flow statistics from sampled packet streams, in: Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW), 2002.
[26] N. Duffield, C. Lund, M. Thorup, Flow sampling under hard resource constraints, in: Proceedings of the ACM SIGMETRICS, 2004.
[27] A. Wagner, T. Dubendorfer, C.G.R. Hiestand, B. Plattner, A fast worm scan detection tool for vpn congestion avoidance, Lecture Notes in Computer Science (2006) 4064–4181.
[28] P. Gross, J. Parekh, G. Kaiser, Secure Selecticast for collaborative intrusion detection systems, in: Workshop on Distributed Event-Based System, 2002.
[29] D. Moore, G.M. Voelker, S. Savage, Inferring Internet denial of service activity, in: Proceedings of the 2001 USENIX Security Symposium, 2001.
[30] P. Barford, J. Kline, D. Plonka, A. Ron, A signal analysis of network traffic anomalies, in: ACM SIGCOMM IMC, 2002.
[31] K. Keys, D. Moore, C. Estan, Robust system for accurate real-time summaries of internet traffic, in: Proceedings of the ACM SIGMETRICS, 2005.
[32] Q.G. Zhao, A. Kumar, J.J. Xu, Joint data streaming and sampling techniques for detection of super sources and destinations, in: Proceedings of the ACM/USENIX Internet Measurement Conference, 2005.
[33] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, in: Proceedings of the International Conference on Very Large Data Bases VLDB, 2002.
[34] G. Cormode, F. Korn, S. Muthukrishnan, D. Srivastava, Finding hierarchical heavy hitters in data streams, in: International Conference on Very Large Data Bases VLDB, 2003.
[35] G. Cormode, F. Korn, S. Muthukrishnan, T. Johnson, O. Spatscheck, D. Srivastava, Holistic UDAFs at streaming speeds, in: Proceedings of the ACM SIGMOD, 2004.
[36] S. Venkataraman, D. Song, P. Gibbons, A. Blum, New streaming algorithms for superspreader detection, in: the Annual Network and Distributed System Security Symposium (NDSS), 2005.
[37] R.R. Kompella, S. Singh, G. Varghese, On scalable attack detection in the network, in: Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference (IMC), 2004.
[38] C.D. Anukool Lakhina, Mark Crovella, Mining anomalies using traffic feature distributions, in: Proceedings of the ACM SIGCOMM, 2005.
[39] M.F. Thomas Karagiannis, Dina Papagiannaki, Blinc: multilevel traffic classification in the dark, in: Proceedings of the ACM SIGCOMM, 2005.
[40] S.B. Kuai Xu, Zhi-Li Zhang, Profiling internet backbone traffic: behavior models and applications, in: Proceedings of the ACM SIGCOMM, 2005.
[41] A.C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, M.J. Strauss, QuickSAND: quick summary and analysis of network data, Tech. Rep., DIMACS Technical Report 2001-43, 2001.
[42] B. Krishnamurthy, S. Sen, Y. Zhang, Y. Chen, Sketch-based change detection: methods, evaluation, and applications, in: Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference (IMC), 2003.
[43] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, in: Proceedings of the 29th International Colloquium on Automata Languages and Programming, 2002.
[44] M. Thorup, Y. Zhang, Tabulation based 4-universal hashing with applications to second moment estimation, in: Proceedings of the ACM–SIAM Symposium on Discrete Algorithms (SODA '04), 2004.
[45] J.D. Brutlag, Aberrant behavior detection in time series for network service monitoring, in: Proceedings of the USNEIX Systems Administration Conference (LISA), 2000.

[46] CERT Coordination Center, Current scanning activity, 2004. <http://www.cert.org/current/scanning.html>.
[47] M. Rajab, J. Zarfoss, F. Monrose, A. Terzis, A multifaceted approach to understanding the botnet phenomenon, in: Proceedings of the ACM IMC, 2006.
[48] V. Yegneswaran, P. Barford, J. Ullrich, Internet intrusions: global characteristics and prevalence, in: Proceedings of the ACM SIGMETRICS, 2003.
[49] S. Staniford, J.A. Hoagland, J.M. McAlerney, Practical automated detection of stealthy portscans, Journal of Computer Security 10 (1–2) (2002).
[50] Insecure.org, Idle scanning and related ipid games, 2004. <http://www.insecure.org/nmap/idlescan.html>.
[51] D. Bernstein, SYN Cookies. <http://cr.yp.to/syncookies.html>.
[52] J. Jung, B. Krishnamurthy, M. Rabinovich, Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites, in: Proceedings of the Eleventh International World Wide Web Conference, 2002.
[53] S. Institute, Dshield.org: Distributed intrusion detection system. <http://www.dshield.org>.

Zhichun Li is a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at Northwestern University. He got his master degree in Computer Science at Tsinghua Univ at 2000. His research interests are on network security, network measurement, and data streaming.

Yan Gao is a Ph.D. candidate in the Department of Electrical Engineering and Computer Science at Northwestern University. Her research interests include network security, network measurement, and monitoring. Gao has a BE in electrical engineering and an MS in system engineering, both from Xian Jiaotong University, China.

Yan Chen is an Assistant Professor in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, IL. He got his Ph.D. in Computer Science at the University of California at Berkeley in 2003. His research interests include network measurement, monitoring and security, and P2P systems. He won the DOE Early CAREER award in 2005 and the Microsoft Trustworthy Computing Awards in 2004 and 2005.