Towards Online Flow-level Anomaly/Intrusion Detection System for High-speed Networks

Abstract

Traffic anomalies and attacks are commonplace in today's networks, and identifying them rapidly and accurately is critical for large network operators. We propose a High-speed Router-based Anomaly and Intrusion Detection system, HRAID, leveraging the recent work on data streaming computation and in particular, sketches. 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. In addition, we design an efficient two-dimensional sketch to distinguish different type of attacks for mitigation. We further propose several heuristics to reduce the false positives for SYN flooding detection. Simulations with several router traces and on-site experiments shows that HRAID is highly accurate, efficient, uses very small memory, and can effectively detect multiple types of attacks simultaneously. To the best of our knowledge, HRAID is the *first* online flow-level anomaly/intrusion detection system for 10s of Gigabit networks, 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 large network operators. It was estimated that malicious code (viruses, worms and Trojan horses) caused over $28 billion in economic losses in 2003, and will grow to over $75 billion in economic losses by 2007 [28]. With the rapid growth of network bandwidth and fast emergence of new attacks/viruses/worms, existing network intrusion detection systems (IDS) [20, 36–38] are insufficient for the following two reasons.

First, they are mostly host-based and not scalable to high-speed networks. Most existing IDSes reside on single host or low-end routers, examining application-level [38], system-level [20] logs, or the sniffed network packets [36, 37] because such detailed information can identify attacks on individual machines. However, today's fast propagation of viruses/worms (*e.g.*, SQL Slammer worm) can infect most of the vulnerable machines in the Internet within ten minutes [34] or even less than 30 seconds with some highly virulent techniques [42, 44]. Thus it is crucial to identify such outbreaks in their early phases, which can only possibly be achieved by detection at high speed backbone routers instead of at end hosts [53]. Also the high speed backbone is suggested as a good vantage point for worm containment [35], for which worm detection on high speed backbone is a crucial prerequisite. Furthermore, for DDoS attacks, although the end hosts can detect and block them, they still can consume much of the network bandwidth or capacity causing the network to become unusable. In fact, it is very hard to implement certain detection techniques, such as those for port scanning, at end hosts. However, the existing schemes are not scalable to the link speeds and number of flows for high-speed networks.

Given a high-speed router with, *e.g.*, an OC-192 link (10Gbps), each 40-byte TCP packet only has 32 ns to proceed [41]. Therefore, three performance features are strongly desirable for high-speed network monitoring and intrusion detection systems: 1) a small amount of memory usage (to be implemented in SRAM); 2) a small number of memory accesses per packet [13, 16]; and 3) scalabilty to a large key space size. The last constraint is especially important for the coming decade: IPv6 with its 128 bit IP address is being adopted, especially in the Asia. Thus the system should scale to a key space of 2^{128} or 2^{256}.

Second, most of the existing approaches are signature based, which cannot detect unknown network attacks. Also, attackers can easily evade such IDSes by changing the attack signatures with polymorphism. Statistical IDSes are therefore proposed to detect anomalous behaviors. Current systems are mostly designed to detect based on the overall traffic [4, 25, 32, 50, 51], therefore they tend to be inaccurate or cannot find real attack flows for mitigation even when spotting anomalies. There are also a few flow-level detection schemes [24, 36, 37], but they are not scalable and thus are themselves vulnerable to attacks. We compare these schemes with ours in Section 2.

To address these challenges, we propose a new paradigm called High-speed Router-based Anomaly and Intrusion Detection system, HRAID, leveraging the recent work on data streaming computation and in particular, sketches [39, 40]. To the best of our knowledge, HRAID is the *first* online flow-level anomaly/intrusion detection system for tens of Gigabit links, even for the worst case traffic of 40-byte-packet streams with each packet forming a flow. Essentially we want to detect as many as possible attacks. As the first step towards this ambitious goal, our threat model includes various port scans (which covers most large-scale worm propagation) and TCP SYN flooding. Our goal is to identify and distinguish them in real-time on high speed networks, and to obtain the attacks' key characteristics for mitigating the attacks. Note 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. As far as we know, none of the previous work is able to achieve that as discussed in Section 2.

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 [39, 40], sketch has not been applied to building intrusion detection systems for the following challenges.

- Sketch 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 detect or identify with such single dimensional information. For example, both horizontal scans and un-spoofed SYN flooding exhibit large number of unsuccessful connections aggregated with the (source IP, destination port) pair. However, it is very hard to differentiate them unless the distribution of the attacks on the destination IPs are also considered.
- Many network element faults, *e.g.*, congestion/failures, routing misconfigurations, and polluted DNS entries, can often lead traffic anomalies detected through the statistical algorithms. How to separate them from the attacks to reduce the false positives?

In this paper, we address these challenges with the following approaches.

- We analyze the attributes in TCP/IP headers and select an optimal small set of metrics for flow-level sketch-based traffic monitoring. Time series analysis algorithms like exponentially-weighted moving average (EWMA) is applied to detect anomalies and intrusions.
- We design an efficient two-dimensional sketches to distinguish different types of attacks for mitigation. Both analytical and empirical results show the effectiveness of the 2-dim sketches.
- We propose several heuristics to separate SYN floodings from network/server congestions and misconfigurations.

As shown in Figure 1, HRAID detection systems can be implemented as black boxes to attach to high-speed routers (edge network routers or backbone routers) of ISPs without affecting the normal operations of the routers. They can enable the early detection of global scale attacks, which is crucial because such attacks can subvert the networking infrastructure and many hosts instantly with exponential growth.

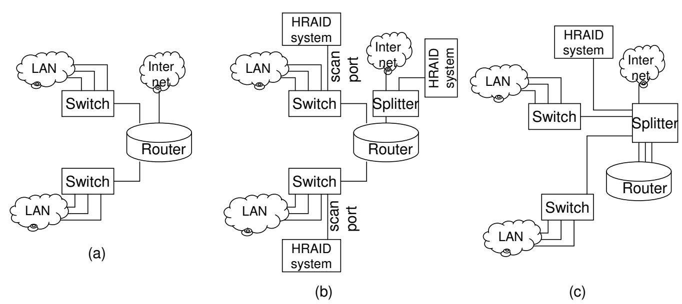

Figure 1: Attaching the HRAID systems to high-speed routers. (a) original configuration, (b) distributed configuration for which each port is monitored separately, (c) aggregate configuration for which a splitter is used to aggregate the traffic from all the ports.

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 a recent DARPA research agenda [53]. It can be naturally extended for response, especially at the edge networks. That is, when alerted, filters can be automatically constructed to filter classes of traffic, and isolate infected machines. Again, such a scheme is considered as "powerful and flexible" by [53].

For evaluation, we first use the router traffic traces collected at Lawrence Berkeley National Labs traces for simulations. Then we apply HRAID for on-site detection at the Northwestern University (NU) edge routers: we record each minute traffic with reversible sketches on the fly. Then at the end of each minute, we use the sketches recorded for online detection. In particular, the one day experiment data consists of 239M network flows of 1.8TB total traffic. We validated the SYN flooding and port scans detected, and found the HRAID system is highly accurate. The 2-dim sketches successfully separate the SYN flooding from port scans, and the heuristics effectively reduce false positives of SYN flooding. The evaluation demonstrates that HRAID significantly outperforms existing approaches like Threshold Random Walk (TRW) [24] and Change-Point Monitoring (CPM) [50, 51]. Compared with statistical detection with complete flow-level data logs, we use much less memory, but have almost the same detection accuracy.

The HRAID system runs in real-time, and requires a small number of memory access per packet. With a Pentium Xeon 3.2 GHz machine with normal DRAM memory, we record 239M flows with one reversible sketch in 20.6 seconds, *i.e.*, 11.6M insertions/second. For the worst case scenario with all 40-byte packets, this translates to around 3.7 Gbps. Even when we run the three reversible sketches of HRAID in serial, we can still archive 3.8M insertions/second. Our prototype of single FPGA board for reversible sketches can achieve a throughput of over 16Gbps for all 40-byte-packet stream. For the NU on-site experiments on a total of 1430 minutes, on average HRAID only uses 0.34 seconds to detect intrusions in each minute traffic, and the standard deviation is 0.64 seconds.

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. Then we introduce the HRAID architecture in Section 4, and discuss the threat model and flow-level monitoring and detection design in Section 5. The two-dimensional sketches are presented in Section 6, and evaluation methodology and results in Section 7. Finally, we conclude in Section 8.

2 Related Work

Although some vendors claim to have multi-gigabit statistical IDSes (*e.g.*, Arbor Networks' Peakflow Traffic [2, 3] and Symantec's Manhunt [45]), they usually refer to *average* traffic conditions and use packet sampling [14, 15] which has two shortcomings. First, sampling is still not scalable, especially after aggregation; there are up to 2^{64} flows even defined only 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 [14]. In contrast, HRAID 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. In a similar spirit, Alcatel, Cisco and Enterasys Networks recently combined IDSes with their switches to defend against worms and DoS attacks [29]. They mostly only use signature-based detection, and all their techniques are proprietary.

Many network IDSs like Bro [36] and Snort [37] check packet payload for virus/worm signatures. However, such schemes are not scalable for high-speed network links. Recently, there are some works proposed to detect large scale attacks, like DoS attacks, port scans, *etc.*, based on the statistical traffic patterns. They can roughly be classified into two categories: 1) Detecting based on the overall traffic [4, 32, 50, 51]. However, the attacks can be easily buried in the background network traffic. Thus such detection schemes tend to be inaccurate or cannot find real attack flows; For example, CPM [50, 51] will detect the port scans as SYN floodings as we verified in Section 7. 2) flow level detection [24, 36, 37], *e.g.*, TRW. However, such schemes usually need to maintain a per-flow table (*e.g.*, a per-source-IP table for TRW [24]) for detection, which are not scalable and thus themselves are vulnerable 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 (AC) [52]. However, in the spoofed DoS attacks will still cause collisions in AC, and make the real port scans undetected[1].

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

Table 1: Functionality comparison of four approaches.

In short, 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. Thus it is like a chicken-and-egg problem. TRW and 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 them 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 [32]. Thus it is another flow-level scheme and can only detect spoofed DoS attacks when

[1]As the authors mentioned in [52], when the connection cache size of 1 million entries reached 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.

the source IP addresses are randomly spoofed. We use that for validating the SYN flooding detected by HRAID.

Venkataraman *et al.*propose efficient algorithms to detect superspreaders, sources that connect to a large number of distinct destinations [48]. 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. PCF was recently proposed for scalable network detection [25]. It uses similar data structures as an original sketch, and thus 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 from various attacks.

3 Background on Sketches for High-speed Network Monitoring

3.1 k-ary Sketch

The sketch, a recently proposed data structure, has proven to be useful in many data stream computation applications [18]. We have designed, implemented and evaluated a variant of the sketch, namely the k-ary sketch [26], and displayed how to detect heavy changes in massive data streams with small memory consumption, constant update/query complexity, and provably accurate estimation guarantees [26]. It is also flexible and can be applied with various forecast models to detect anomalies.

The input stream arrives sequentially, item by item. Each item consists of a key and an update $value$. The k-ary sketch has three basic functions: UPDATE, ESTIMATE and COMBINE. Among them, UPDATE is most frequently used and has the most stringent real-time requirement. Table 2 shows the definition of these functions. Suppose there is a sketch with source IP as the key and packet size as the value. When a 68-byte packet arrives, the UPDATE function will increase the value of the corresponding source IP 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. Many statistical approaches, such as Time Series Analysis (TSA), need this functionality for anomaly/trend detection. Moreover, the linearity of reversible sketches enables traffic aggregation over multiple routers to facilitate distributed detection. With sketches, we can record hundreds of millions of flows with only a few hundred kilobytes of memory. More detailed mathematics analysis for k-ary sketch is in the Appendix A.1.

Functions	Descriptions	k-ary sketch	Reversible sketch
UPDATE (S, y, v)	Update the observed sketch in the monitoring module	$\surd$	$\surd$
v = ESTIMATE (S, y)	Reconstruct the signal series for statistical detection for a given key in anomaly detection module	$\surd$	$\surd$
S = COMBINE $(c_1, S_1, ..., c_k, S_k)$	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 anomaly detection module	$\surd$	$\surd$
y = INFERENCE (S, t)	Return the keys whose values are larger than the threshold		$\surd$

Table 2: Function of sketches (S-Sketch, v-Value, y-Key, t-Threshold).

3.2 Reversible k-ary Sketch

Although the sketch data structures have good linear properties and can accurately estimate the value for any given key, they have one major drawback: they are not *reversible*. That is, a sketch cannot efficiently report the set of all keys that have large value estimates in the sketch. This means that to compare two streams, we would have to know which keys to query and the streams with the largest changes. Naively, this would require either exhaustively testing all possible keys, or recording and testing all data stream keys and corresponding sketches. Unfortunately, neither option is scalable.

To address these problems, we propose a novel framework for efficiently *reversing* sketches [39, 40] (For more details, please refer to Appendix A.2). The basic idea is to hash intelligently by modifying the input keys and/or hashing functions so that we can recover the keys with certain properties intact, such as big changes in a given category, 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 are larger than 1MB in 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.

4 Architecture of the HRAID System

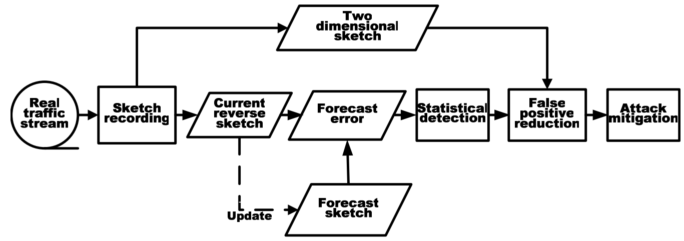

Figure 2: HRAID System architecture.

Figure 2 shows the architecture of the HRAID system. First, we record the network traffic with sketches using UPDATE function. Then we apply different time series analysis methods to obtain the forecast sketches for change detection by COMBINE function, based on the linearity properties of sketches. The forecast time series analysis method, *e.g.*, EWMA (exponentially weighted moving average) and Holt-Winter algorithm [6], can help remove noises for detections. 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 adopt the two-dimensional sketches to further distinguish different types of attacks, and apply other false positive reduction techniques as discussed in Section 5.3. Finally, we use the key characteristics of the culprit flows revealed by the reversible sketches to mitigate the attacks.

Here, we apply the EWMA algorithm as the forecast model to do change detection. The following characteristic we used for detection for EWMA algorithms is the `#SYN−#SYN/ACK` counted in a certain time interval. Denote $\mathbf{M}_0(t)$ as the `#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

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

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

5 The Threat Model and Detection Algorithms

5.1 The Threat Model

Ultimately, we want to detect as many different types of attacks as possible. As a first step, we focus on arguably the two most popular intrusions for detection: TCP flooding denial of service (DoS) attack and port scan/worm propagation. Actually, our work also can be easily extended to detect UDP or ICMP flooding: use the number of packets or traffic volume instead of `#SYN−#SYN/ACK`. We focus on TCP flooding here because it is reported that more than 90% of DoS attacks are TCP SYN flooding attacks [50, 51].

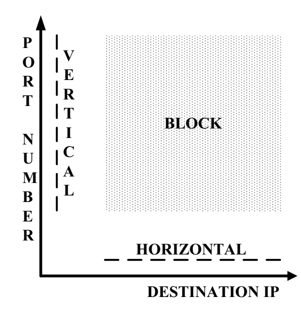

Figure 3: Three types of scans.

Scans are probably the most common type of intrusion and many of them are caused by worm propagation. According to statistics data by CERT [7], most existing worms are TCP worms. Most 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. 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* [43, 49] as illustrated in Figure 3.

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 [43, 49]. 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, have made the IP-ID sequences less predictable and rendered the attack obsolete [21]. Among the three types of scans in Figure 3, horizontal scans are the most common type of scan, and scan a given port on IP addresses in some range of interest. The port number is often unique because it reflects 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 [43]. The third type of scan, a block scan, is a combination of horizontal and vertical scans over numerous services on numerous hosts [43]. Block scan is not very common in practice, thus we focus on horizontal and vertical scans.

5.2 Sketch-based Detection Algorithm

The challenge then is to detect and differentiate these attacks 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 sketch as `K`, the feature value recorded in sketch as `V`, and the reversible sketch as `RS(K,V)`. Since normally we only extract fields in the IP header, the possible fields we can use are shown in Figure 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 detection. The Figure 4 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. The best key would ideally correspond to only one type of attack. Nevertheless, normally a key can be related to several types of attacks, so we need use more dimensions to differentiate these attacks as shown in Section 6.

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

Table 3: The fields in IP header used in detection

Types of Keys	SYN flooding	horizontal scan	vertical scan	block scan
{`SIP`,`Dport`}	non-spoofed	Yes	No	Yes
{`DIP`,`Dport`}	Yes	No	No	No
{`SIP`,`DIP`}	non-spoofed	No	Yes	Yes
{`SIP`}	non-spoofed	Yes	Yes	Yes
{`DIP`}	Yes	No	Yes	Yes
{`Dport`}	Yes	Yes	No	Yes

Table 4: The uniqueness of different types of keys.

From Figure 4 we see that, the combinations of two fields have better uniqueness than single fields, so we use the 3 combinations of two fields as keys for the 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 by one, while for each outgoing SYN/ACK packet, the sketch will be updated by decrementing by one. In fact, similar structures can be applied to detect any partial completion attacks [25]. The reversible sketch can further provide the victim IP and port number for mitigation as in Section 5.4. We denote this set of `DIP`s 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. They 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 and victim 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.

Thus altogether, we need three reversible sketches to record the traffic characteristics. The classification rules are depicted in Table 5. In addition to {`SYN, SYN/ACK`} pairs, other features like {`SYN, FIN`} pairs or {`SYN/ACK, FIN`} pairs can also be applied for detection as discussed in [50, 51]. After the detection, we use the heuristics described in Section 5.3 and 2-dim sketches in Section 6 to reduce the false positives.

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: Different attacks are detected in different reversible sketches.

5.3 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 unresponded SYNs. For instance, flash crowds, network/server congestions/failures, and even polluted or outdated DNS entries may exhibit a large number of SYNs without SYN/ACK at the edge routers. These will cause relatively high false positives in our detection scheme. For the flash crowds, it is very hard, if not impossible, to differentiate it from the SYN flooding attacks without payload information as discussed in [23]. Thus we target reducing the false positives caused by the other two behaviors listed above.

Filters to reduce the false positives caused by bursty network/server congestions/failures We believe a real SYN flooding should deny the service to certain degree, *e.g.*, the number of successful connections is less than those unsuccessful ones. That is, $\frac{SYN-SYN/ACK}{SYN/ACK}$ is greater than one. Moreover, the typical TCP SYN flooding attacks observed in the Internet last for a certain period, such as 10 minutes [33, 51]. On the other hand, 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 mins). The system administrators can adjust their threshold to get the tradeoff between sensitivity and the number of alerts.

To implement the first filter, we added 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, when we found any SYN flooding suspect key, we put it into a suspect array with a counter for each one. And then for each subsequent interval, we query their `SYN` and `SYN-SYN/ACK` value from the related sketches, and increment the counter if the above inequality holds.

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 found 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 misconfigrations. 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 then may 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 HRAID with the LBL and NU data as discussed in Section 7, we did find several such scenarios. For instance, in the one-day LBL data, we found there was a SYN flooding like anomaly towards a certain IP with large `#SYN-#SYN/ACK`. The anomaly lasted for 3 hours, but there were 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 heuristics by filtering those anomalies of which the victim does not have any response to SYN requests during the most recent 24 hour period. Due to the linearity of sketches, we can add up the sketches in the past 24 hours, and then estimate the number of SYN/ACK for the suspect `{DIP, Dport}`. If the number is less than certain 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 the monitoring phase.

5.4 Intrusion Mitigation

The HRAID system can instruct routers or system administrators to mitigate the attacks based on the attack type and their key characteristics as obtained from reversible sketches. For distributed SYN flooding, we can start the *SYN defender* [8], *SYN proxy* and/or *SYN cookies* [5] for the particular victim machines and applications to alleviate the DoS effects. Other DoS attack mitigation schemes [31] can also be applied. For port scans and point-to-point SYN flooding, we can use an ingress filter to block the traffic from the attacker IP. For vertical scans, we can quarantine the victim machine for further inspection. For horizontal scans, we can pay particular attention to any suspicious internal or outgoing traffic with the same port number because if any internal machine is compromised, it will probably try to infect more machines within the network.

6 Intrusion Classification with Two Dimensional Sketch

One major challenge for anomaly 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 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/ACK packets from one source to one destination. That is, one mode corresponds to the SYN flooding, and the other, vertical scans. This bi-modal assumption is verified with real network traffic in Section 6.1. 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 6.2.

6.1 Verification of the Bi-modal Distribution for SYN flooding vs. Port Scans

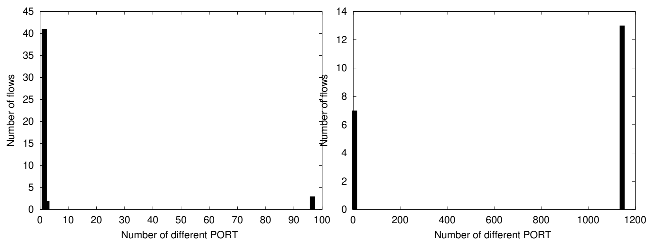

Figure 4: The distribution of the number of attacks with respect tothe number of unique ports visited when there are more than 50 un-responded SYNs in 1-minute interval between one {`SIP,DIP`} pair. Left: NU, right: Fermi Lab

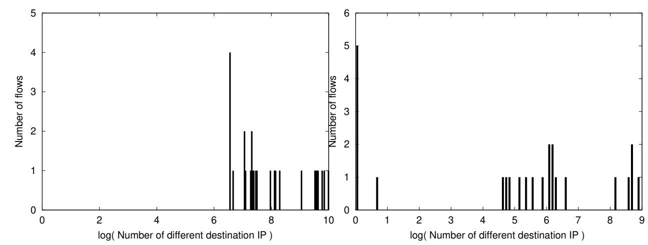

Figure 5: The distribution of the number of attacks with respect tothe number of unique destination IPs visited when there are more than 100 un-responded SYNs in 1-minute interval with one `SIP,Dport` pair. Left: NU, right: Fermi Lab

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, Figure 4 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. That is, SYN flooding attacks have a very small number of destination IPs, while horizontal scans have a much larger amount. Figure 5 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-minute interval from one source 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 two hour NU traffic.

Lakhina *et al.*recently also examine the traffic feature distribution with entropy for anomaly detection, for all OD flows between a pair of point of presence (POPs) [1]. Their scheme, however, cannot give the key of culprit flows for mitigation even when spotting anomalies.

6.2 Two-dimensional Sketch Design and Intrusion Classification

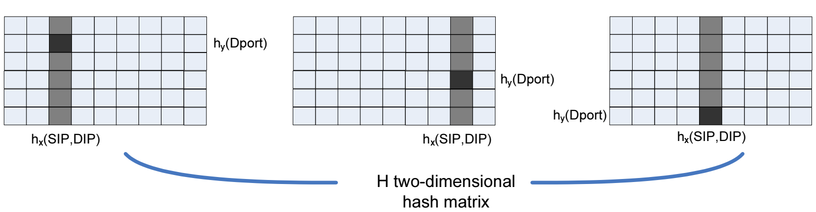

Figure 6: Architecture of the two-dimensional k-ary sketch

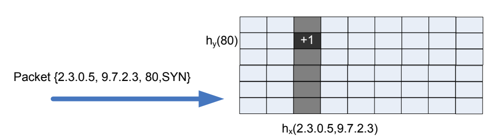

Figure 7: An example of UPDATE operation for two-dimensional sketch

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 Figure 6. 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 is the concatenation of `SIP` and `DIP`, and the y dimension is `Dport`. For each packet, we locate its corresponding entry in the matrix by two independent hash mappings as shown in Figure 7, and update the bucket in the same manner as for reversible sketches in Section 5.2. Similarly, we can update all H matrices for data stream recording.

In the detection stage, when we find an attack by using reversible sketch or any other methods (*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. Then we obtain the sum of the top p (*e.g.*, 5 out of 64) buckets with the largest values. If the sum is larger than ϕ ($\phi < 1$, *e.g.*, 0.8) times the detection threshold D *and* the sum is larger than ϕ' (*e.g.*, 0.8) times the total sum in all buckets, then we regard it as a SYN flooding. Only when all H hash matrices of the 2D sketch imply it as a SYN flooding, we can conclude it is a SYN flooding attack; otherwise it is a vertical scan. Similarly, we can differentiate horizontal scans from the SYN flooding attacks.

Accuracy Analysis When we use the bi-modal distribution to separate the two attacks, with random variation, there can be some false positives and false negatives for classification. We use *Generalized Increment-Decrement Counter Model* [25] to obtain the bound on the false positives and negatives due to the randomness of normal traffic background. For the detailed bound analysis, please refer to Appendix B.1. We found that for even a small size of 2D sketch (*e.g.*, 3 matrices, each matrix has $2^{12} \times 2^{6}$ buckets), we can get a misclassification possibility of less than 2.88×10^{-6} even for 10 million packets.

Given that the number of buckets in the 2D sketches is much less than the key space size of $\{$`SIP, DIP`$\}$, (*i.e.*, 2^{64}), different $\{$`SIP, DIP`$\}$ pairs may collide into the same column of a hash matrix. Then we may misclassify a SYN flooding as a vertical scan. Similarly, it is possible that a SYN flooding attack and a vertical scan may collide to the same columns of all 2D hash matrices, for which we will misclassify some vertical scans as a SYN flooding attack. In the Appendix B.2, we proved that the misclassification possibility is also very small, less than 2.274×10^{-7} for the parameters we used.

7 Evaluation

7.1 Evaluation Methodology

In this section, we evaluate HRAID 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 HRAID for on-site detection at the Northwestern University (NU, which has several Class B networks) edge routers. The router exports netflow data continuously which are recorded with sketches of HRAID on the fly. Then at the end of each minute, HRAID detects intrusions online as shown in Figure 2. 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 37K/s and the peak packet rate is 79K/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 one minute.

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*.

The HRAID system consists of three reversible sketches (RS), one for $\{$`SIP,Dport`$\}$, one for $\{$`DIP,Dport`$\}$, and the other for $\{$`SIP,DIP`$\}$, one original sketch (OS) for $\{$`DIP,Dport`$\}$, and two 2D sketches for $\{$`SIP,Dport`$\}$ and $\{$`SIP,DIP`$\}$. For all the RS and 2D sketches we update `SYN - SYN/ACK` as value, and only for the one original sketch we use `SYN` as value. For the number of buckets used in the sketches, we chose two set of parameters according to the different memory requirements. They used a total memory of 9.4MB and 44.5MB respectively. For both settings, we use 6 stages for each RS and OS, and 3 stages for each 2-dim sketch.

For the small memory setting, we use 2^{12} buckets for each stage in 48-bit RS, 2^{14} buckets for each stage in OS, and 2^{16} buckets for each stage in the 64-bit RS. 2^{14} buckets are applied for all their verification sketches. We use $2^{12} \times 64$ buckets for each stage of the 2D sketches.

For the medium memory setting, we use 2^{18} buckets for each stage in the 48-bit RS, 2^{16} buckets for each stage in the OS, and 2^{16} buckets for each stage in the 64-bit RS. 2^{16} buckets are adopted for all their verification sketches. $2^{14} \times 64$ buckets are used for each stage of the 2D sketches.

Both NU and LBL have quite 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 attack. The thresholds can be adjusted by network administrators.

To evaluate the accuracy of the HRAID system, we take the following steps. First, we solely evaluate the error introduced by the sketches (Section 7.2). Second, we compare the HRAID system with several existing network intrusion detection approaches in Section 7.3. Finally, we manually validate the attacks we detected in Section 7.4.

7.2 Sketches Highly Accurate in Recording Traffic for Detection

Table 6 shows the results when using EWMA as the forecasting method. There are three phases. We first detect the attacks through using reversible sketches with algorithms described in Section 5.2. The results are shown as "Raw results" as in Table 6. Then we use 2-dim sketches to reduce the false positives for port scans introduced by SYN floodings, and get results in the column of "Phase 2" of Table 6. Finally, we use heuristics in Section 5.3 to reduce false positives of SYN flooding and the results are presented as "Phase 3".

To evaluate the errors introduced by sketches, we compare the results obtained from the same detection algorithm but with three different types of traffic recording: 1)small-size sketches; 2) medium-size sketches; and 3) accurate flow table to hold per-flow information (we call it non-sketch method). We found that we detect exactly the same attacks for the three configurations with very different amount of memory (see memory consumption discussion in Section 7.5). This shows that sketches are highly accurate in recording the traffic for detection.

Traces	Attack type	Phase1: Raw results	False positive reduction	
			Phase2: Port scan	Phase3: 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 6: Detection results under three phases.

7.3 HRAID Outperforms Other Existing Network IDSes

We compare our approach with other state-of-the-art network intrusion detection approaches as introduced in Section 2: TRW [24] for port scan detection and CPM [50, 51] for SYN flooding detection.

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 HRAID with the source IP address. We observe that the scans detected by these two methods have very good overlap, except a few special cases. There are a small number of horizontal scans detected by HRAID 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 multiply 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 to 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 HRAID. 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 $\{\texttt{SIP, DIP}\}$ or $\{\texttt{SIP, Dport}\}$, the number of scans is relatively small and less than our threshold. But TRW aggregates the scan by source IP, and count the number of unique destination IP 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 are still relatively small, *i.e.*, not major attacks. It is part of our future work to further investigate such problems.

Next, we compared our method with CPM for SYN flooding attacks 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 that for each interval whether there is an attack or not. For a fair comparison, we also count the number of intervals that SYN flooding detected by HRAID spans 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 and has very high false positives. On the other hand, CPM and HRAID 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 when SYN flooding is buried in the rest of normal traffics, which have large number of normal SYN and SYN/ACK packets from the traffic of other hosts, and thus for total traffic $\frac{SYN-SYN/ACK}{SYN/ACK} < 1$ so that CPM cannot detect them.

Traces	TRW	HRAID	Overlap number
NU	497	512	488
LBL	695	699	692

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

Traces	CPM	HRAID	Overlap number
NU	1422	1427	1422
LBL	1426	0	0

Table 8: TCP SYN flooding detection comparison of HRAID and CPM.

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 9: Top 10 of Horizontal scans in NU trace.

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 10: Bottom 10 of Horizontal scan in NU 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

Table 11: Top 10 of Horizontal scan in 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

Table 12: Bottom 10 of Horizontal scan in LBL trace.

7.4 Detected Intrusions Successfully Validated

In the previous section, we compare HRAID 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.

7.4.1 SYN Flooding

We validate our SYN flooding detection results with backscatter [32] 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 by the victim, which is not valid in many cases (observed from our netflow data as well). Furthermore, our evaluation is to detect attacks in the incoming traffic. 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 flooding 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 [32]. Another three have IP spoofed in some regular manner. For instance, one has 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 because ISPs tend to drop randomly spoofed packets through ingress filtering.

7.4.2 Horizontal Scans

We manually validate those horizontal scans, in particular, top 10 and bottom 10 attacks in terms of their change difference. Table 9 shows the top 10 horizontal scans for NU trace, whose SYN-SYN/ACK change count ranges from 24,000 to 60,000, and Table 10 shows the bottom 10 horizontal scans, whose SYN-SYN/ACK change count ranges from 60 to 64. Table 9 and Table 9 demonstrate the top 10 and bottom 10 horizontal scans for LBL trace. Through these evaluation table, we find that they are all possible attacks, including Nachi or MSBlast worm, SQLSnake worms, W32.Sasser, SSH scans, *etc.*. There are even some unknown worm scans detected by us, and also confirmed in the Dshield [22], the largest worldwide intrusion log repository.

7.4.3 Vertical Scans

We also manually validate vertical scans for both LBL trace and NU trace. In LBL trace, there is one vertical scan we found. It scanned some well-known service ports, such as HTTPS(81), HTTP-Proxy(8000,8001,8081). In NU trace, totally we found 19 vertical scans. We manually check 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 also are interested in Trojan/BackDoor ports. For example, WinHole(808,1082), Slapper(1978), SubSeven 2.1(7000), Tini(7777), OpwinTRojan(10000), *etc.*. All these characteristics drive us to believe those vertical scans we detected have the high probability to be real scans.

7.5 Evaluation Results for Online Performance Constraints

7.5.1 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, the small memory setting only needs a total of 9.4MB. If the system administrators are only interested in the most severe attacks, *e.g.*top 100 attacks, they can choose this setting. Otherwise, if more stealthy attacks need to be detected, the medium memory setting can be applied with 44.5MB. Note that such settings works well for a large range of link speeds, as we tried on different network traces. Sketch can always report the relative heavy changes and heavy hitters, as the statistical accuracy bound are given in [26, 40].

On the other hand, if hash tables are used to keep the accurate per flow table, it may use much larger memory 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 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 an new entry will be added to both the {`SIP,DIP`} table and the {`SIP,Dport`} table. Every entry of the hash table needs 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 2.5Gbps, for one second, the total memory consumption is $(2.5G/(8 \times 40)) \times 12 \times 2$=187.5MB. For TRW, for each {`SIP,DIP`} pair, it maintains a likelihood ratio which is updated based SYN and SYN/ACK observed. Thus for the example above, the total memory consumption is $(2.5G/(8 \times 40)) \times 12$=93.75MB. Thus both of the methods will run out of memory very fast on these scenarios.

Methods	2.5Gbps			10Gbps		
	1sec	1min	5min	1sec	1min	5min
HRAID with small sketch	9.4M			9.4M		
HRAID with medium sketch	44.5M			44.5M		
HRAID with complete info	187.5M	11.25G	56.25G	750M	45G	225G
TRW	93.75M	5.63G	28G	375M	22.5G	112.5G

Table 13: Memory comparison(bytes).

7.5.2 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 [39] for details). For each two-dimension sketches, we only need 3 memory accesses per packet, one for each 2-dim hash matrix. Thus, when recording these sketches in parallel or in pipeline, the HRAID system has very small number of memory access per packet and is capable of online monitoring.

7.5.3 Traffic Monitoring and Intrusion Detection with High Speeds

The HRAID 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 HRAID system, we will implement it in hardware so that multiple sketches can be updated in parallel. With our prototype a single FPGA board implementation, we are able to sustain 16.2Gbps 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 seconds, *i.e.*, 11M insertions/sec. 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/sec.

For on-site NU experiments over a total of 1430 minutes, on average HRAID only uses 0.34 seconds to detect for each one-minute interval, and the standard deviation is 0.64 seconds. The maximum detection time (for which the interval contains the largest number of attacks) is 12.91 seconds, which is still far less than one minute.

8 Conclusion

In this paper, we propose a High-speed Router-based Anomaly and Intrusion Detection system, HRAID, leveraging the recent work on sketches. Experiments with several router traces show that HRAID is highly accurate, efficient, uses very small memory, and can effectively detect multiple types of attacks simultaneously.

References

[1] ANUKOOL LAKHINA, MARK CROVELLA, C. D. Mining anomalies using traffic feature distributions. In *Proc. of ACM SIGCOMM* (2005).

[2] ARBOR NETWORKS. Intelligent Network Management with Peakflow Traffic. `http://www.arbornetworks.com/download.php` (registration required).

[3] ARBOR NETWORKS. Service Provider Infrastructure Security: Detecting, Tracing, and Mitigating Network-Wide Anomalies with Peakflow DoS. `http://www.arbornetworks.com/download.php` (registration required).

[4] BARFORD, P., ET AL. A signal analysis of network traffic anomalies. In *ACM SIGCOMM IMC* (November 2002).

[5] BERNSTEIN, D. SYN Cookies. http://cr.yp.to/syncookies.html.

[6] BRUTLAG, J. D. Aberrant behavior detection in time series for network service monitoring. In *Proc. of the USNEIX Systems Administration Conference (LISA)* (2000).

[7] CERT COORDINATION CENTER. Current scanning activity. `http://www.cert.org/current/scanning.html`, 2004.

[8] CHECKPOINT SOFTWARE TECHNOLOGIES. TCP Flooding Attack and Firewall-1 SYNDefender. `http://checkpoint.com/press/1996/synattack.html`.

[9] CHEN, C., AND LIU, L. Forecasting time series with outliers. *Journal of Forecasting 12* (1993), 1335.

[10] CHEN, C., AND LIU, L.-M. Joint estimation of model parameters and outlier effects in time series. *Journal of the American Statistical Association 88* (1993), 284297.

[11] CORMODE, G., KORN, F., MUTHUKRISHNAN, S., JOHNSON, T., SPATSCHECK, O., , AND SRIVASTAVA, D. Holistic UDAFs at streaming speeds. In *Proceedings of ACM SIGMOD* (2004).

[12] CORMODE, G., KORN, F., MUTHUKRISHNAN, S., AND SRIVASTAVA, D. Finding hierarchical heavy hitters in data streams. In *VLDB 2003* (2003).

[13] CORMODE, G., AND MUTHUKRISHNAN, S. What's new: Finding significant differences in network data streams. In *Proc. of IEEE Infocom* (2004).

[14] DUFFIELD, N., LUND, C., AND THORUP, M. Properties and prediction of flow statistics from sampled packet streams. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMW)* (2002).

[15] DUFFIELD, N., LUND, C., AND THORUP, M. Flow sampling under hard resource constraints. In *Proc. of ACM SIGMETRICS* (2004).

[16] ESTAN, C., AND VARGHESE, G. New directions in traffic measurement and accounting. In *Proc. of ACM SIGCOMM* (2002).

[17] FRALEIGH, C., ET AL. Packet-level traffic measurements from the sprint IP backbone. *IEEE Network* (2003).

[18] GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. J. QuickSAND: Quick summary and analysis of network data. Tech. Rep. 2001-43, DIMACS, 2001.

[19] HADLOCK, C. R. *Field Theory and its Classical Problems*. Mathematical Association of America, 1978.

[20] HOFMEYR, S., AND FORREST, S. Intrusion detection using sequences of system calls. *Journal of Computer Security 6* (1998).

[21] INSECURE.ORG. Idle scanning and related ipid games, 2004. `http://www.insecure.org/nmap/idlescan.html`.

[22] INSTITUTE, S. Dshield.org: Distributed intrusion detection system. `http://www.dshield.org`.

[23] JUNG, J., ET AL. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proc. of WWW* (2002).

[24] JUNG, J., ET AL. Fast portscan detection using sequential hypothesis testing. In *Proc. of the IEEE Symposium on Security and Privacy* (2004).

[25] KOMPELLA, R. R., SINGH, S., AND VARGHESE, G. On scalable attack detection in the network. In *Proc. of ACM/USENIX IMC* (2004).

[26] KRISHNAMURTHY, B., SEN, S., ZHANG, Y., AND CHEN, Y. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. of ACM SIGCOMM Internet Measurement Conference (IMC)* (2003).

[27] MANKU, G. S., AND MOTWANI, R. Approximate frequency counts over data streams. In *Proc. of IEEE VLDB* (2002).

[28] MARS, E., AND JANSKY, J. D. Email defense industry statistics. `http://www.mxlogic.com/PDFs/IndustryStats.2.28.04.pdf`.

[29] MESSMER, E. Switches taking on new security roles. `http://www.nwfusion.com/news/2004/0614switchsecurity.html?docid=2441`.

[30] MIRKOVIC, J., AND REIHER, P. Data streams: algorithms and applications. In *Proc. of ACM SODA* (2003).

[31] MIRKOVIC, J., AND REIHER, P. A taxonomy of DDoS attacks and defense mechanisms. In *ACM CCR* (April 2004).

[32] MOORE, D., ET AL. Inferring Internet denial of service activity. In *Proceedings of the 2001 USENIX Security Symposium* (Aug. 2001).

[33] MOORE, D., ET AL. Inferring Internet denial-of-service activity. In *Proceedings of the USENIX Security Symposium* (2001).

[34] MOORE, D., ET AL. The spread of the Sapphire/Slammer worm. http://www.caida.org, 2003.

[35] MOORE, D., SHANNON, C., VOELKER, G. M., AND SAVAGE, S. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of the IEEE Infocom* (2003).

[36] PAXSON, V. Bro: A system for detecting network intruders in real-time. *Computer Networks 31*, 23-24 (1999), 2435–2463.

[37] ROESCH, M. Snort: The lightweight network intrusion detection system, 2001. `http://www.snort.org/`.

[38] RYUTOV, T., NEUMAN, C., KIM, D., AND ZHOU, L. Integrated access control and intrusion detection for web servers. *IEEE Trans. on Parallel and Distributed Systems 14*, 9 (2003), 841–850.

[39] SCHWELLER, R., GUPTA, A., PARSONS, E., AND CHEN, Y. Reverse hashing for sketch-based one-pass change detection for high-speed networks. Tech. Rep. 2003-31, Northwestern University, May 2004.

[40] SCHWELLER, R., GUPTA, A., PARSONS, E., AND CHEN, Y. Reversible sketches for efficient and accurate change detection over network data streams. In *IMC* (2004).

[41] SIKKA, S., AND VARGHESE, G. Memory-efficient state lookups with fast updates. In *Proc. of ACM SIGCOMM* (2000).

[42] STANIFORD, S., ET AL. How to own the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium* (2002).

[43] STANIFORD, S., ET AL. Practical automated detection of stealthy portscans. *Journal of Computer Security 10*, 1-2 (2002).

[44] STANIFORD, S., MOORE, D., PAXSON, V., AND WEAVER, N. The top speed of flash worms. In *Proc. of ACM CCS WORM Workshop* (2004).

[45] SYMANTEC INC. Symantec ManHunt. `http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=%156&EID=0`.

[46] TSAY, R. S. Time series model specification in the presence outliers. *Journal of the American Statistical Association 81* (1986), 132141.

[47] TSAY, R. S. Outliers, level shifts, and variance changes time series. *Journal of Forecasting 7* (1988), 1–20.

[48] VENKATARAMAN, S., SONG, D., GIBBONS, P., AND BLUM, A. New streaming algorithms for superspreader detection. In *the Annual Network and Distributed System Security Symposium (NDSS)* (2005).

[49] VINOD, ET AL. Internet intrusions: Global characteristics and prevalence. In *Proc. of ACM SIGMETRICS* (2003).

[50] WANG, H., ZHANG, D., AND SHIN, K. G. Detecting SYN flooding attacks. In *Proc. of IEEE INFOCOM* (2002).

[51] WANG, H., ZHANG, D., AND SHIN, K. G. Change-point monitoring for detection of DoS attacks. *IEEE Transactions on Dependable and Secure Computing 1*, 4 (2004).

[52] WEAVER, N., ET AL. Very fast containment of scanning worms. In *USENIX Security Symposium* (2004).

[53] WEAVER, N., PAXSON, V., STANIFORD, S., AND CUNNINGHAM, R. Large scale malicious code: A research agenda. Tech. Rep. DARPA-sponsored report, 2003.

A More Details about Sketches

A.1 k-ary Sketch

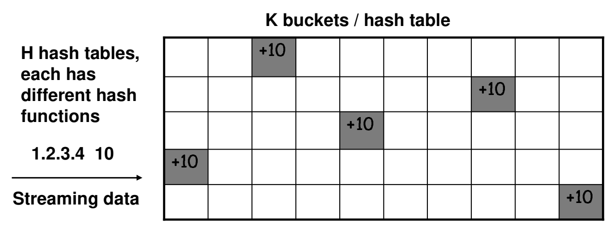

Figure 8: A sample k-ary sketch structure and its UPDATE operation for a sample input stream with an item (1.2.3.4 10). The key is 1.2.3.4 and the value is 10.

The ever-increasing link speeds and traffic volumes of the Internet make monitoring and analyzing network traffic a challenging but essential service for managing large ISPs. This service is especially important for anomaly/intrusion detection. 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 period to another. There is a significant amount of prior work on efficient and online heavy hitter detection [11, 12, 16, 27].

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 variational changes and the linear transformation changes for use with various time-series forecasting models [9, 10, 26, 46, 47]. Moreover, in today's networks, asymmetric routing, multi-homing, and load balancing are very common and many enterprises have more than one upstream or downstream links. It is crucial to do spatial aggregation over the traffic of multiple routers for anomaly/intrusion detection, which can be only enabled through the linearity of heavy *change* detection, like the k-ary sketches introduced below.

We adopt one of the most general data stream models, the Turnstile Model [30]. Let $I = \alpha_1, \alpha_2, ...,$ be an input stream that arrives sequentially, item by item. Each item $\alpha_i = (a_i, u_i)$ consists of a key $a_i \in [n] \equiv 0, 1, \ldots, n-1$, and an (possibly negative) update $u_i \in \mathbb{R}$. Associated with each key $a \in [n]$ is a time varying signal $A[a]$. The arrival of each new data item (a_i, u_i) causes the underlying signal $A[a_i]$ to be updated: $A[a_i] += u_i$. The goal of anomaly detection is to identify all those signals with anomalous behavior.

The model above is very general and one can instantiate it in many ways with specific definitions of the keys and updates. In the context of network anomaly detection, the key can be defined using one or more fields in packet headers, such as source and destination IP addresses. The updates can be the size of a packet, the total bytes or packets in a flow.

As shown in Figure 8, a k-ary sketch S consists of a $H \times K$ table of registers: $T_S[i][j]$ ($i \in [H], j \in [K]$). Each row $T_S[i][\cdot]$ ($i \in [H]$) is associated with a hash function from $[n]$ to $[K]$: h_i. We can view it as an array of hash tables. The K-ary sketch has three basic functions: UPDATE to update a sketch, ESTIMATE to reconstruct v_a for a given key a, and COMBINE to compute the linear combination of multiple sketches. The forml specification of these functions is as follows.

- UPDATE(S, a, u): For $\forall i \in [H], T_s[i][h_i(a)] += u$.
- ESTIMATE(S, a): Let $sum(S) = \sum_{j\in[K]} T_s[0][j]$ be the sum of all values in the sketch, which only needs to be computed once before any ESTIMATE(S, a) is called. Return an estimate of $v_a = median_{i\in[H]}\{v_a^{h_i}\}$, where $v_a^{h_i} = \frac{T[i][h_i(a)]-sum(S)/K}{1-1/K}$. Each $v_a^{h_i}$ is an unbiased estimator of v_a with variance inversely proportional to (K-1), please look at [26] for further description.
- COMBINE(c_1, S_1, ..., c_l, S_l): The linearity of the sketch data structure allows us to linearly combine multiple sketches $S = \sum_{k=1}^{l} c_k.S_k$ by combining every entry in the table: $T_s[i][j] = \sum_{k=1}^{l} c_k.T_{S_k}[i][j]$.

A.2 Reversible Sketch

However, original k-ary sketches do not preserve the key (*e.g.*, source IP address) of the flows. Hence, even if anomalies are detected, it is difficult to infer the culprit flows, making it a big practical hurdle for online deployment.

To address this challenge, we propose efficient *reversible hashing* algorithms to infer the keys of culprit flows from sketches without storing any explicit key information. Few extra memory or memory accesses are needed for recording the streaming data. Meanwhile, the heavy key detection daemon runs in the background with space complexity and computational time sublinear to the key space size. Here we will introduce some concept and simple cases of heavy key inference, for the more detailed algorithm please refer to our technique report [39].

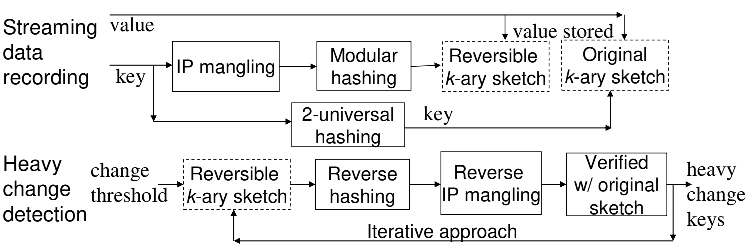

Figure 9: Architecture of the reversible k-ary sketch based heavy key detection system.

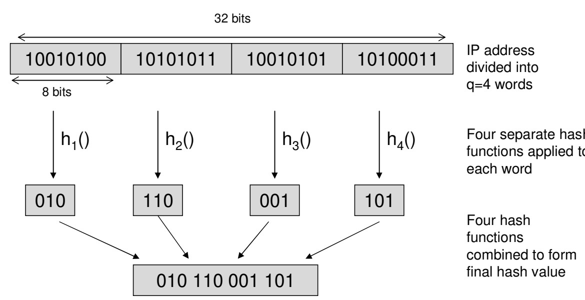

Figure 10: Modular hashing uses q hash functions to hash each word of the key, which are then combined for the final hash.

The framework of reversible sketch based heavy key detection has two parts as in Fig. 9: streaming data recording and heavy key detection.

Since k-ary sketches are not reversible, to allow reversibility, we modify the update procedure of the k-ary sketch with *modular hashing*. To maintain the accuracy of the sketch with this type of hashing, we also need to perform *IP-mangling*.

A.2.1 Modular Hashing and IP Mangling for Streaming Data Recording

Modular hashing is illustrated in Figure 10. Instead of hashing the entire key in $[n]$ directly to a bucket in $[m]$, we partition the key into q words, each word of size $\frac{1}{q}\log n$ bits. Each word is then hashed separately with different hash functions which map from space $[n^{\frac{1}{q}}]$ to $[m^{\frac{1}{q}}]$. For example, in Figure 10, a 32-bit IP address is partitioned into $q = 4$ words, each of 8 bits. Four independent hash functions are chosen which map from space $[2^8]$ to $[2^3]$. The results of each of the hash functions are then concatenated to form the final hash. In our example, the final hash value would consist of 12 bits, deriving each of its 3 bits from the separate hash functions h_1, h_2, h_3, h_4.

However, an important issue with modular hashing is the quality of the hashing scheme. Consider network traffic streams, which exhibit strong spatial localities in the IP addresses, *i.e.*, many simultaneous flows only vary in the last few bits of their source/destination IP addresses, and share the same prefixes. With the basic modular hashing, the collision probability of such addresses is significantly increased. For example, consider a set of IP addresses $129.105.56.*$ that share the first 3 octets. Our modular hashing always maps the first 3 octets to the same hash values. Thus, all distinct IP addresses with these octets will be uniformly mapped to 2^3 buckets, resulting in a lot of collisions. This observation is further confirmed when we apply our modular hashing scheme with the network traces. To overcome this problem, we introduce the technique of *IP mangling*.

In *IP mangling* we attempt to artificially randomize the input data in an attempt to destroy any correlation or spatial locality in the input data. The objective is to obtain a completely random set of keys, and this process should be still reversible.

The general framework for the technique is to use a bijective function from key space $[n]$ to $[n]$. For an input data set consisting of a set of distinct keys $\{x_i\}$, we map each x_i to $f(x_i)$. We then use our algorithm to compute the set of proposed heavy change keys $C = \{y_1, y_2, \dots, y_c\}$ on the input set $\{f(x_i)\}$. We then use f^{-1} to output $\{f^{-1}(y_1), f^{-1}(y_2), \dots, f^{-1}(y_c)\}$, the set of proposed heavy change keys under the original set of input keys. We propose the IP mangling functions based on simple arithmetic operations on a Galois Extension Field [19] $\mathbb{GF}(2^\ell)$, where $\ell = \log_2 n$.

A.2.2 Single Heavy Change Detection and Key Inference

Once we have updated the sketch for each item in the data stream we want to obtain the set of suspect heavy keys from the sketch, *i.e.*, the set of keys that hash to heavy buckets in each of the H hash tables. One possible way is to check all keys in the space $[n]$ which obviously is not scalable. The second basic approach is to perform bucket intersections. Suppose for each hash table i there is exactly one heavy bucket. Denote the set of keys that hash to the heavy bucket in table i as A_i. We can determine the set of suspect keys by computing the set $\bigcap_{i \in [H]} A_i$. The problem is that each set A_i is of expected size $\frac{n}{m}$, and is thus quite large. However, if we are using modular hashing, we can implicitly represent each set A_i by the cross product of q *modular* reverse mapping sets $A_{i,1} \times A_{i,2} \times \cdots A_{i,q}$ determined by the corresponding modular hash functions $h_{i,w}$. The pairwise intersection of any two reverse mapping sets is then $A_i \bigcap A_j = A_{i,1} \bigcap A_{j,1} \times A_{i,2} \bigcap A_{j,2} \times \cdots \times A_{i,q} \bigcap A_{j,q}$. We can thus determine the desired H-wise intersection by dealing with only the smaller *modular* reverse mapping sets of size $(\frac{n}{m})^{\frac{1}{q}}$. This is the basic intuition for why modular hashing might improve the efficiency of performing reverse hashing and constitutes the approach used in [39].

A.2.3 Multiple Heavy Change Detection and Key Inference

Extending the intuitions for the multiple heavy bucket cases is not trivial, actually, much complex. The algorithm for the single bucket cases cannot be directly extended. Since it is quite complicated, we ignore the algorithm here, for more information about multiple heavy buckets algorithm please refer to our technique report [39].

B Accuracy Analysis of Two Dimensional Sketch

B.1 Misclassification Caused by the Randomness of Normal Traffic

We use *Generalized Increment-Decrement Counter Model* [25] to obtain the bound on the false positives and negatives due to the randomness of normal traffic background. 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 called a *measurement interval*, T, the counter should accumulate all the packets hashed to it to get the outcome value N. For the normal traffic, the expectation of N, $E(N)$, should be equal to zero. Since the time between a SYN and the corresponding SYN/ACK is close to the Round Trip Time (RTT), which is normally very short (most likely it will be less than 500ms), for the normal traffic if $T >> RTT$ then the N should be close to zero. The SYN and its corresponding SYN/ACK should fall into the same measurement interval T. There are several reasons which could make N non-zero:

- When the SYN packet happens at the end of the *measurement interval*, its corresponding SYN/ACK could be in the next interval.
- Sometimes, the SYN or SYN/ACK may be resent.
- Occasionally, the response time from a busy server could be quite long (longer than one interval).
- Route churn may cause SYN or SYN/ACK to be missing.

Observe that in each of these malformed but benign cases, the anomaly is equally likely to add an extra SYN as it is to add an extra SYN/ACK to an interval. Therefore all these cases can be modeled as contributing, in the general case, either a SYN or SYN/ACK with probability 0.5. Without loss of generality, let us randomly pick up a counter. Let X_i represent the random variable that represents the ith random event that happens in the interval T. For X_i, $P\{X_i = 1\} = 0.5$ and $P\{X_i = -1\} = 0.5$. Therefore, the expectation of X_i, μ_i, is 0 and the standard deviation, σ_i, is 1. For all n random events, the final value of the counter will be $X = \sum_{i=0}^{n} X_i$. For a sufficiently large value of n, the Central Limit Theorem assures that the distribution of X will approach a Gaussian distribution. And for X we have:

$$\mu = \mathrm{E}(X) = 0 \qquad \sigma^2 = \mathrm{VAR}(X) = n \qquad Pr\left[a \le \frac{X-\mu}{\sigma} \le b\right] = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-z^2/2} \mathrm{d}z \tag{2}$$

Such random events would only influence a very small portion of the SYN and SYN/ACK packets. And normally all of SYN and SYN/ACK packets only account for 1/3 or less of the total packets on network. So the ratio ϵ of random events to the total packets is normally very small (less than 5%). Given network speed S and measurement interval T, the total random events within T will be ϵST, so the random events n for each bucket of a two dimensional hash matrix will be $n = \frac{\epsilon ST}{K_x K_y}$, K_x and K_y are the number of columns and rows of the two dimensional hash matrix, respectively.

As we discussed before, a SYN flooding attack should only send SYNs to few ports (*e.g.*1-3), and a vertical scan should evenly distribute the SYN to many ports. In practice, we use this method to differentiate these two attacks: for each two dimensional hash matrix, the $\{$`SIP,DIP`$\}$ can select the column of buckets. Then for that column, we first obtain the sum of the top p buckets. If it is larger then ϕ ($\phi < 1$) times the threshold of $D = min(\text{vertical scan, SYN flooding})$, then we regard it as SYN flooding; otherwise it is a vertical scan.

Because we only use the two-dimensional sketch to test the attacks which have already been detected in the reversible sketches, the false positive and negative to normal traffic will be as low as the false positive and negative of the reversible sketch (actually, even lower). We can always ignore it. So we only need to calculate the ratio that the vertical scans are incorrectly classified as SYN flooding attacks, r_1 and the ratio that the SYN flooding attacks are incorrectly classified as vertical scans, r_2.

Since Equation 2 shows the normal traffic in every bucket can be approximated as a Gaussian random variable, and as we know the sum of p Gaussian random variable still is Gaussian, so the sum of p buckets is still a Gaussian random variable Y, $E(Y) = \mu_Y = 0$ and $E[(Y - \mu_Y)^2] = \sigma_Y^2 = np$. Then we will calculate the possibility that a vertical scan is actually in that column but due to the randomness, the sum of the top p packet can larger than ϕD. If one vertical scan whose total SYN packets are above the threshold D exists, and it is misclassified as a SYN flooding attack, then $\frac{pD}{K_y}$ is the portion of the SYN packets on the top p buckets caused by the vertical scan, the possibility of it misclassified as a SYN flooding is $P_1 = Pr\left[\frac{X_p}{\sqrt{np}} \geq (\phi - \frac{p}{K_y})D\right]$.

And the possibility of a SYN flooding misclassified as a vertical scan is $P_2 = Pr\left[\frac{X_p}{\sqrt{np}} \leq -(1 - \phi)D\right]$. This is because excluding the SYN flooding traffic which are larger than D, the sum of the top p buckets is a Gaussian random variable Y with $\mu_Y = 0$ and $\sigma_Y^2 = np$.

In a K-ary two-dimensional sketch we have H independent two-dimensional hash matrices, thus $r_1 = P_1^H$ and $r_2 = P_2^H$. If we choose $K_x = 2^{12}$, $K_y = 2^6$, $\phi = 0.8$, $\phi' = 0.8$,$D = 100$, $p = 5$, $H = 3$, and $\epsilon = 5\%$, for 2.5Gb/s network, we assume the average packet size is 400 bytes [17], measurement interval is 5 minutes , we can obtain that $n = 37.5$, $r_1 = 2.178 \times 10^{-17}$, and $r_2 = 2.879 \times 10^{-6}$. Therefore, we only need a few stages of Hash matrices obtain very good results. This result above actually only consider the sum of top p bucket is larger than ϕD. If we require the both conditions mentioned in Section 6.2 to meet, the result could be even better.

B.2 Misclassification Caused by Collision

However, up to now one thing we still have not considered is that given that the number of buckets in the 2D sketches is much less than the key space size of $\{$`SIP, DIP`$\}$, (*i.e.*, 2^{64}), different $\{$`SIP, DIP`$\}$ pairs may collide into the same column of a hash matrix. Then, we will misclassify some vertical scans as SYN flooding attacks and vise versa. Now we will calculate the chance of collision by probability model. Suppose the `SIP` and `DIP` of both vertical scans and SYN flooding attacks are randomly distributed. In one measurement interval T, if there are m_v vertical scans and m_s SYN flooding attacks, most likely the m_v vertical scans are distributed in m_v columns[2]. Then, the possibility of one random SYN flooding attack collided in the column of vertical scans is $p = m_v/K_x$. Totally we have m_s SYN flooding attacks, so it follow binomial distribution $prob(l) = \binom{m_s}{l} p^l (1-p)^{(m_s - l)}$. Here l is how many collisions could happen. Since $m_v << K_x$ and $m_s << K_x$, the possibility of one collision of a two-dimensional hash matrix will be $\frac{m_v m_s}{K_x}$. And since we have H independent two-dimentional hash matrices, thus the possibility of one collision will be $(\frac{m_v m_s}{K_x})^H$. if the $m_v = 5$, $m_s = 5$, $K_x = 2^{12}$ and $H = 3$, the possibility of one collision will be 2.274×10^{-7}.

[2]There are some chances of two vertical scans can collide to one column, but the chances are very little, and it can only reduce the chances of collision between vertical scans and SYN flooding attacks