On Scalable Attack Detection in the Network

Ramana Rao Kompella, Sumeet Singh, George Varghese
University of California, San Diego,
{ramana,susingh,varghese}@cs.ucsd.edu

***Abstract*— Current intrusion detection and prevention systems seek to detect a wide class of network intrusions (e.g., DoS attacks, worms, port scans) at network vantage points. Unfortunately, even today, many IDS systems we know of keep per-connection or per-flow state to detect malicious TCP flows. Thus it is hardly surprising that these IDS systems have not scaled to multi-gigabit speeds. By contrast, both router lookups and fair queuing have scaled to high speeds using *aggregation* via prefix lookups or Diff-Serv. Thus in this paper, we initiate research into the question as to whether one can detect attacks without keeping per-flow state. We will show that such aggregation, while making fast implementations possible, immediately causes two problems. First, aggregation can cause *behavioral aliasing* where, for example, good behaviors can aggregate to look like bad behaviors. Second, aggregated schemes are susceptible to *spoofing* by which the intruder sends attacks that have appropriate aggregate behavior. We examine a wide variety of DoS and scanning attacks and show that several categories (bandwidth based, claim-and-hold, port-scanning) can be scalably detected. In addition to existing approaches for scalable attack detection, we propose a novel data structure called partial completion filters (PCF) that can detect claim-and-hold attacks scalably in the network. We analyze PCFs both analytically and using experiments on real network traces to demonstrate how we can tune PCFs to achieve extremely low false positive and false negative probabilities.**

I. Introduction

While the very success of the Internet is due to its open model in which any computer can send to any other computer, this openness also allows attackers to send malicious messages that can cause damage to other hosts and networks, sometimes at great cost. Thus the field of network security has sprung up in an attempt to prevent or mitigate attacks against campus, enterprise, and ISP networks.

The earliest network security solutions attempted to secure Internet hosts using *anti-virus* software running at end-nodes, and *firewalls* installed at network vantage points (or, more recently, at hosts themselves). Unfortunately, end-node based approaches must be widely deployed within a network to protect against attacks. They also do very little to mitigate bandwidth attacks that may be blocked at the end-nodes but consume so much internal network bandwidth that the network is unusable. Similarly, most distributed denial-of-service (DDoS) attacks and scans routinely penetrate firewalls using essential services such as HTTP or email.

For these reasons, a number of researchers and vendors have suggested perimeter defenses that sit at the entrance to networks or subnets. Besides firewalls, a traditional approach has been to do *intrusion detection* (and sometimes mitigation) at such points. Two classical approaches to intrusion detection have been *anomaly detection* and *signature detection*. Signature detection [1] is useful to detect an important class of attacks (e.g., known worms and viruses) but is not helpful in detecting other attacks (e.g., scans, DDoS attacks) which are not characterized by a signature within *a single packet*, but by unusual behavior across *a set of packets*. In this paper, we concentrate on detecting and mitigating such attacks (that can only be detected by behavior across a set of packets) at network vantage points.

While anomaly detection also targets such attacks [2], anomaly based detection is often very general, and works by first automatically identifying a baseline for "normal" network behavior (using say wavelets [2] or change point detection [3]) and then flagging deviations from such behaviors as possible attacks. A difficulty with the most general approaches is establishing normal behaviors because most network traffic behaviors evolve in unpredictable ways.

Network based behavioral approaches: A simpler (but less general) approach to anomaly detection is to look for violations of specific behaviors, where a description of the bad behaviors has been *preprogrammed* into the detection device. In this paper, we will refer to such techniques as network based behavioral approaches. Such approaches have been widely deployed ([4], [5], [6], [7], [8], [9]).

To be effective, anomaly detection must run at a network vantage point where it sees a lot of traffic. A minimal deployment would be at the edge of a subnet; a more useful deployment would be at the entrance to a network; a potential future deployment would be within ISPs. Notice that for some attacks such as DoS, it is helpful to detect the attack as upstream in the attack path as possible, to reduce the collateral damage caused to innocent sources that share the attack path until the detection device.[1]

Running a detection device at the edge of a network requires that the detection device operate at link speeds of 1 Gbps and higher. Such positions also expose the detection system to a larger number of flows making it more difficult. But most current behavioral based approaches do maintain per-flow state. For example to detect port scans, Snort maintains a large vector per source to count all the ports and destinations each source talks to. Not only does this plugin take a large amount of space, but it also slows down the Snort code considerably when it is enabled. Bro [10] also maintains per-flow state in order to detect evasion and other attacks. Similar observations hold for many other approaches to detecting DDoS [9].

To meet the speed challenge, several vendors (e.g., NetScreen, Fortinet) and researchers are implementing detection in hardware. While signature detection is being done in

[1] While this might argue for moving detection close to sources, this does not work well either because often we have little control over rogue sources and their networks.

hardware, it has proved difficult to speed up network based behavioral approaches.

Scalable attack detection: The main question we examine in this paper is whether it is possible to scale behavioral network detection to very high speeds. To begin to grapple with this question, it is instructive to consider other network functions that have been made to be scalable. Both in the case of IP lookups and network QoS, scalability has been achieved via *aggregation* to reduce the state used by the function so as to fit into high speed memory. For example, Internet lookups in routers use prefix aggregation to store around 150,000 prefixes for the entire Internet. Similarly, DiffServ uses class aggregation to avoid per-flow state in core routers.

Aggregation helps with forwarding performance for the following reasons. First, the number of connections/flows at network vantage points can easily scale into the millions, and this does not scale with the increases in the size of high speed memory. Second, the highest speed memories (on-chip and off-chip SRAM or cache) scale far more slowly than the number of flows.

Thus we ask : *can we use aggregation to reduce the state required for behavioral attack detection?* While several types of attacks (e.g., evasion, TCP hijacking) have been proved to be impossible to detect in a scalable fashion [11], we will show in this paper that under some minimal assumptions, scan detection and DDoS detection (and perhaps several others) can indeed be detected scalably using aggregation.

The use of aggregation for scalable attack detection immediately creates two fundamental problems:

- *Behavioral aliasing*: One form of behavioral aliasing occurs when a set of well behaved connections aggregate to look like bad behavior, creating a *false positive*. For example, when detecting a port scan, if we aggregate several sources into an aggregate, while each source may talk to only a few distinct destinations, the aggregate may look like it is talking to lots of destinations. A rigorous argument [11] can be based on this intuition to show that attempts to scalably detect port scans using such a predicate do not work. A second form of behavioral aliasing occurs when the aggregate behavior of several badly behaved connections looks like good behavior – a *false negative*.
- *Spoofing:* Spoofing occurs when an intelligent attacker subverts the detection mechanisms by suitably spoofing the attack to appear benign. For example, the SYN-DOG approach to syn-flood detection [9] does a first level of aggregation by keeping state only on a per-destination basis (not sufficient, but a good start) for the difference between SYNs and FINs going to each destination. Such a scheme can always be spoofed by the attacker sending spurious FINs that do not serve to finish any active connection but only confuse the detection mechanism.

We believe that any scalable intrusion detection mechanism must deal with these two issues. Thus the contributions of this paper are as follows:

1. Framework: Our paper initiates the study of scalable attack detection schemes. We use behavioral aliasing and spoofing as a framework to analyze such techniques.

2. Technique: As a specific example, we focus on scalable DDoS and scan detection, and propose a specific new scalable technique called Partial Completion Filters (PCFs). We analyze behavioral aliasing and spoofing characteristics of PCFs in different deployment scenarios.

3. Evaluation: To evaluate the efficacy of PCFs, we use a theoretical model later validated by real traces from two different ISPs. For example, in an OC-48 traffic trace for an entire day, we were able to identify about 517 attack flows out of a total of 30.36 Million flows over the entire trace period.

The rest of this paper is organized as follows. In Section II, we describe the kind of scanning and DoS attacks, called partial completion attacks, we consider in this paper. Section III introduces a scalable mechanism, PCFs, to detect partial completion attacks and describe a theoretical analysis that shows why it is resilient to behavioral aliasing and (in some deployments) to spoofing. In Section IV, we describe experimental evaluation of PCFs for scalable monitoring and detection of partial completion attacks, scanning based attacks. Section V contains implementation details of PCFs in routers followed by related work in Section VI and conclusions in Section VII.

II. Attack Classification

We restrict our focus in this paper to analyzing the following three different types of attacks.

Partial completion attacks: Such attacks are also known as claim-and-hold attacks. The basic theme in these attacks is to grab a precious resource and not release it thereby denying service to legitimate clients. The classic example of a partial completion attack is syn-flooding. In syn-flooding, the attacker initiates several connections to the server by sending TCP SYN packets with spoofed IP addresses and never terminates any of these connections. In a variant called Naptha [12], the attacker initiates a connection and finishes the initial three way handshake, but does not do any further activity forcing the connection to time out. In both these attacks, we can see that the precious resource namely, connection memory, is claimed but never released.

Attacks that do scanning: Host scanning represents an important component of several attacks including most worm epidemics [4], [13], [14][2]. Thus several recent worms such as Code Red-II [16], Nimda [17] etc., propagated by scanning other vulnerable hosts in the Internet. A second example is probing for backdoors installed on various machines either installed during worm infection or by other means such as viruses. Such activity also exhibits scanning behavior. Finally, horizontal (multiple hosts and same port) and vertical (one machine, multiple ports) port scans are often performed by attackers as preliminary reconnaissance to identify a large number of vulnerable hosts in the Internet. Henceforth, we refer to these myriad activities as just *scanning*.

Bandwidth attacks: Finally, the third kind of attacks we discuss in this paper are what are commonly called *bandwidth attacks*. In such attacks, an attacker or a set of compromised slaves (zombies), continuously pound a victim with a large

[2]Note that there exist worms that spread through other means and hence do not exhibit scanning; for example, MyDoom spread through email and a Kazaa vector [15].

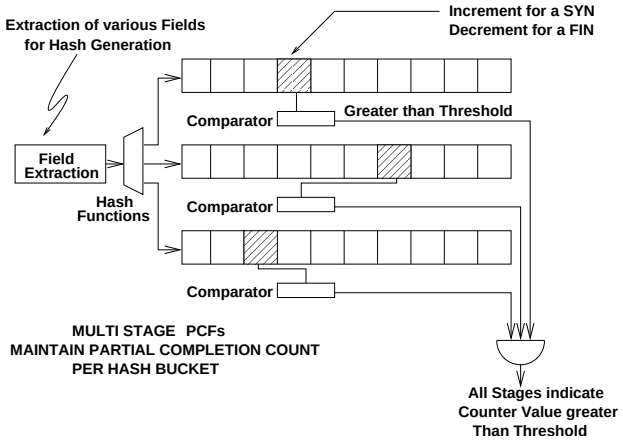


Fig. 1. Partial completion filters

number of packets, crippling normal services. In other such attacks, the attacker can take advantage of other machines to amplify the magnitude of traffic directed towards a particular destination. Smurf [18], fraggle, and reflector attacks [19] fall into this category. The common theme in all such attacks is huge traffic volume.

III. Detection of TCP scans and partial completion attacks

It is fairly immediate to see that bandwidth attacks have scalable solutions using existing techniques such as MULTOPS [20], sketches [21], [22], multistage filters [23] and tools such as Autofocus [24]. On the other hand, these techniques do not work well to detect TCP flood attacks since the traffic volume of SYNs (especially early in the attack tree) may not be large enough compared to the volume of benign traffic. Sampling [25], [23] works best for bandwidth attacks because an attack with a large traffic footprint is more likely to be sampled. However, it is not at all clear how sampling can be used to detect partial completion and scan-based attacks which have much smaller traffic footprints. Given the importance of these low traffic attacks, in this section we introduce a new data structure — *partial completion filters (PCFs)* that can detect both scanning attacks and partial completion attacks even when they correspond to small traffic volumes.

A. Partial completion filters: algorithm

Partial completion filters identify flows with high imbalance between two types of control packets that are usually balanced. For example, benign TCP connections consist of equal number of SYN and FIN packets – PCFs can be used to detect SYN-flooding that involves transmitting only SYN packets (and hence high imbalance between SYNs and FINs).

PCF data structure consists of parallel stages each containing a set of counters. Packets are hashed based on the header fields using multiple independent hash functions (Figure 1) and counters indexed by these hash functions are incremented/decremented for the two types of control packets. If all the counters indexed by the hashes of a packet are above a particular threshold (exhibiting high imbalance), the flow is output. At the end of a measurement interval, these counters are all reset.

The expected value of these counters is zero if there are equal number of SYN/FIN packets in a given flow. However the standard deviation is $O(\sqrt{N})$ after N packets. Thus a benign bucket may have fairly large positive counters (causing false positives) while a bucket containing an attack may be be pulled down to zero (causing a false negative). The tricky part is to show that *both* false negatives and false positives stay within control for reasonable parameter values.

One might hastily conclude that PCFs are the same as multistage filters first proposed in [23] to detect heavy-hitter flows in the network. This is not true for the following three reasons:

1. Non-monotonicity: In multistage filters (and in fact in all Bloom filter [26] variants), the counters are only incremented and never allowed to be negative.

2. False negatives: Bloom filters and multistage filters have only one-sided errors; there are no false negatives. Unfortunately, since PCFs allow counters to decrease, they can cause false negatives.

3. Different analysis: The analysis of PCFs using the Central Limit theorem (Section III-B) is very different from the simple counting argument for multistage filters.

Also, the design of a PCF reflects a delicate balance between false positive and false negative rates. For instance, using more than 3 stages is almost always a bad idea for PCF while it is always a good idea for multistage filters.

B. Behavioral aliasing in PCFs

In this section, we provide a theoretical analysis that allows us to predict the behavior of as well as tune PCFs in real network settings. The analysis is in three parts. In Part 1, we will use the Central Limit Theorem and tail bounds on Gaussian distributions to bound the false negative and false positive probabilities, which in turn determines the operating range of PCFs. In Part 2, we identify how to use PCFs to detect flows that greater than a given threshold. Finally, in Part 3, we analyze the false positives and false negatives in the presence of other bad flows.

Before we proceed, note that if a flow begins and ends in a given measurement interval, then the contribution of that flow to the counters would be 0. However, due to the presence of intervals, there can be benign but malformed connections. First, a connection may be long-lived, in which case it contributes its SYN to one measurement interval, and its FIN to another measurement interval. Second, a connection may retransmit its FIN. However, as a first-order approximation, we can assume that a connection is equally likely to retransmit its SYN. In practice, TCP has a built-in asymmetry that makes SYN retransmissions happen slightly more often than that of FIN retransmissions. After using our first-order model (with equal retransmission probabilities), we show how this small bias can easily be corrected for in Section IV-A. Third, route churn may cause the SYN to be seen but not the FIN, but in that case, during another interval due to another route churn, it might see the FIN but not the SYN. On average, a set of measurement intervals should be able to smooth out this noise. In [27], the authors have experimentally verified that the routes are stable on the

scale of a few minutes. So, we believe that the noise generated due to route churn is not really significant. Nevertheless, our analytical model captures this effect as well. In all these cases, we can simply assume that in a given measurement interval, the probability of a SYN or a FIN is 0.5.

Part 1, Estimating noise range of PCFs: Without loss of generality, since the counters are chosen at random, let us consider one particular counter and a particular measurement interval. Let X_i represent the random variable that represents the value added to the counter in the ith trial. X_i is 1 with probability 0.5 and is -1 with same probability. The expectation of X_i say μ is 0, and standard deviation σ is 1. After n trials, we are interested in what the final value of $X = \sum_{i=0}^{n} X_i$ is. This represents the current counter value assuming that the counter started at zero. In the case of SYN-FIN correlation, this number represents the number of excess SYNs or FINs that have arrived for this bucket.

The exact distribution for counter values follows a binomial for which it is difficult to estimate tail probabilities in closed form. Fortunately, for sufficiently large values of n[3], the Central Limit Theorem assures us that the binomial distribution can be approximated using a Normal distribution. Thus we can use cookbook Normal tables to obtain confidence bounds on the probability that the counter value is above a particular threshold. If $X = \sum_{i=0}^{n} X_i$,

$$Pr\left[a \leq \frac{X - n \cdot \mu}{\sigma \cdot \sqrt{n}} \leq b\right] = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-z^2/2} dz$$

As if we choose, $a = -3, b = 3$, also, $\mu = 0$, $\sigma = 1$, then

$$Pr\left[|X| \leq 3\sqrt{n}\right] = 0.9987$$

Note that these bounds can be found in statistical tables for Z-ratios (e.g., [28]). Therefore, counters of buckets containing only unattacked destinations should approximately lie within 3 times the standard deviation times $\sqrt{n}$. For example, in a measurement interval suppose there are 10 million SYN/FIN packets. Also, let the number of buckets be 3000. The expected number of trials per bin would be 10 million divided by 3000 which is approximately 3300 i.e, $n = 3300$.

From the above probability calculation, the probability that the counter value is less than $3 \cdot \sqrt{3300} = 172$ is rather large, and in fact 0.9987. So, even if all the connections were benign, the counters can be between -172 and 172 with very high probability. This determines the operating range of PCFs. Note that we can build more sensitive PCFs by choosing a larger number of buckets so that the noise range becomes smaller. In fact the noise range of PCF is inversely proportional to $\sqrt{n}$, where n is the number of buckets in PCF.

The probability that a benign flow maps to any counter greater than the calculated noise of 3σ is 0.0013. In order to reduce this even further, we add k parallel stages to the PCF. The probability that in k stages the counter value is greater than 3500 (as in the previous example) becomes 0.0013^k. For $k = 3$, the probability is $2.197\text{x}10^{-09}$ which is extremely small. This bounds the noise range of the filter to 3σ with very high probability. Another reason to add stages is to reduce the probability of false positives that occur when an unattacked destination hashes into a bucket containing an attacked destination (discussed in part 3).

Therefore, if the number of stages increases, the probability that a benign flow maps to all counters greater than 3σ drops *exponentially*. However, false negatives, i.e, the probability that a flow greater than 3σ goes undetected increases *linearly*. This is because if the flow maps to at least one counter for which the negative noise cancels out its contribution, it appears benign. The more stages we use, the more likely it is for the flow to map to one such counter.

Due to an exponential decrease in the false positive probability and linear increase in the false negative probability, a small number of stages can drastically decrease the false positive probability without causing the false negative probability go too high. For example, we show later (in Section IV-A) that choosing 3-4 stages often reduces false positive probability while keeping false negative probability low enough.

Dynamic setting of threshold T: As we have seen, the noise range of the filter is dependent on the number of packets in the given measurement interval. How then can we dynamically adapt this threshold ? There are many ways we can approximate the exact count of packets in a given interval. One simple way is to use the count of number of packets in the previous interval as a means to estimate the threshold. If the traffic does not exhibit drastic change in different intervals (if the intervals are suitably small), then this estimate of the threshold remains correct. One other way to choose this threshold is based on past history of this particular link. However, if we explicitly set the threshold to a reasonably large value, as we show in the next part of the analysis, there would be no need to dynamically adjust the threshold.

Part 2, Using PCFs to detect flows greater than a threshold T: Earlier, we have obtained a theoretical bound on the noise that PCF counters are susceptible to, even if all the flows were benign. The noise arising from other benign flows is additive with that of a malicious flow that hashes to these buckets. Hence, a flow of size s can hash to buckets that lie between $s - 3\sigma$, and $s + 3\sigma$ with high probability. Hence, if we choose the PCF threshold to be T, false negatives (flows of size greater than T are undetected) increases. So, we should choose the PCF threshold to be at least $T - 3\sigma$. However, this increases the false positives since now, a flow of size $T - 6\sigma$ can, due to the positive noise, appear as a malicious flow with high probability.

From this analysis, we can see that if we are interested in flows greater than size T, choosing a threshold value of $T - 3\sigma$ allows us to guarantee two properties. First, PCF identifies all flows with greater than T with high probability. Second, if PCF identifies a flow, the flow is at least of size $T - 6\sigma$ with high probability.

Part 3, Estimating false positives and false negatives in the presence of attacks: So far, we have analyzed those cases where we found the false positive and false negative probability in the presence of one malicious flow. In the presence of more number of attack flows, a subset of the buckets appear large in-

[3]And for reasonably large measurement intervals on real traffic, n is indeed large enough

creasing the number of false positives. We now estimate false positives in the presence of a number of attack flows.

For the analysis, let us assume that there are b attack flows. It is easy to prove that the expected number of buckets to which these b attack flows hash to is $n(1-(1-1/n)^b)$, where n is the number of buckets. We call such buckets as bad buckets. If $b \ll n$, this expected number of bad buckets is approximately b. Any flow, even if it is benign, that hashes to one of these b buckets in all the three stages is deemed an *attack flow*. The probability that a flow hashes to one of these bad buckets in one stage is given by b/n. For k stages, the probability is $\left(\frac{b}{n}\right)^k$. The expected false positives now is given by $\left(\frac{b}{n}\right)^k \cdot f$ where f is the total number of benign flows, namely the bad flows subtracted from total flows.

For example, suppose the total number of flows were, say, 250,000 out of which 500 of them were genuinely bad flows. Let us assume that the total number of buckets is 1000 per stage and there are three stages. The expected number of buckets into which these 500 bad flows hash to is approximately 394. Therefore the expected number of false positives is $(394/1000)^3 \times 249,500 = 15,265$. If there were only 100 bad flows, then the number of false positives is only about 250. As we can see from this analysis, the number of false positives increases super-linearly with the number of bad flows. In cases where the number of bad flows is only around 20, the number of false positives decreases to close to one.

A similar analysis applies to false negatives. In the presence of a reasonable number of flows with larger negative SYN-FIN differences, some of the buckets become unreasonably small. Any flow that maps to these buckets would not be detected. However, this case is less often, since SYN packets usually dominate FIN packets for any flow.

The bottom line from the analysis is that one can tune PCFs appropriately to obtain reasonably high probabilities of detecting a partial completion attack or a TCP scan, while making sure there are not too many false positives. Note that despite the small chance of missing an attack, if several routers in the attack path are using this scheme, then it is more likely that one of them will detect the attack.

C. Applying PCFs to detect partial completion and scanning attacks

Notation: We use PCF(A, B, C) to denote a PCF that increments (decrements) on a TCP packet with flags A (B), and uses C as the field(s) used to hash the packet.

1. Partial completion detection: For the detection device, the key abstract behavior that signals a SYN flood to a destination is the presence of a *destination* that receives a large number of SYNs from various sources.[4] Thus a PCF(SYN, FIN, <DIP,DP>) can be used to scalably detect a TCP SYN flood attack by hashing based on *destination* IP address, port pairs.

[4]Note that unlike a number of other approaches like backscatter [6] our approaches work regardless of whether the attacker employs address spoofing. Thus we can detect DDoS attacks that use a large army of zombies (increasingly common today) that often use true IP addresses. We can also detect reflector attacks, reflector servers use true IP address to the victim since it sends a response packet to the original request packet.

In the network, if we assume that the detection mechanism cannot see both directions of the traffic, the attacker can easily spoof the PCF by transmitting an additional FIN packet along with the SYN packet. We show how to make SYN flood detection spoof resilient using *reverse path* deployments in next section (Section III-E), though this will require an inversion: the trick is to hash source addresses (as opposed to destinations) in the reverse path to identify victims. This is because, without collusion from inside the network, the attacker cannot force the victim to send FIN packets.

2. TCP scanning detection: At a network vantage point, during TCP scanning activity such as portscan, a detection device can observe a large number of SYN packets to a particular port but with no corresponding FIN packets that correspond to legal tearing down of the connection.

Therefore, for the detection device the key abstract behavior that signals a TCP scan is the presence of a *source* that sends a large number of SYNs to various destinations and destination ports without sending a corresponding FIN. Thus, a PCF(SYN, FIN, <SIP>) can be used to scalably detect a TCP scan by hashing based on source IP addresses and zeroing in on such sources that have a large SYN-FIN imbalance.

In the network, if we assume that the detection mechanism cannot see both directions of the traffic as we have assumed, then PCF methods are easily subject to spoofing. One approach is to ignore spoofing because most attackers employ tools such as NMAP [29] that do not spoof today; however, we will show how to make detection spoof-resilient using bidirectional deployments (in scenarios where we can see both directions of traffic) in Section III-E. Next, we apply PCFs for scalable monitoring of partial completion and scanning attacks in the network.

D. Applying PCFs for Attack Monitoring

PCFs can be applied to characterize attack flows in an online fashion, in contrast to current approaches that rely on passive traces. Note however that, for the ease of validation, we used real traces to evaluate PCFs in Section IV. Using PCFs, we can identify attack flows (sources and destinations using different PCFs or can be combined into one by hashing each packet twice), count the estimated size and duration of attacks in a scalable fashion. We will show in Section IV-B.2 our experiences with PCFs in scalable characterization of attack flows.

As we have seen earlier, PCF(SYN, FIN, <DIP,DP>) based on destination IP address, port pairs can detect the destinations under attack in a scalable fashion. We call this the *forward* path of the attack since we infer DoS activity based on the attack packets going towards a victim.

PCF(SYN, FIN, <SIP,SP>), based on source IP address, port pair can be effective in monitoring based on the *reverse path* of the attack. This follows the fact that a victim under attack generates several SYN-ACK packets but no corresponding FIN packets. This is because the connection typically does not get established, and even when it does, it does not terminate. Together, the forward and reverse path PCFs can aid in scalable monitoring and characterization of partial completion based DoS activity with an ISP network domain. The forward

path PCF however is spoofable, but the reverse path PCF is *spoof-resistant* as we discuss later in Section III-E.

Network telescopes [6] based on "backscatter analysis" are currently employed to infer world-wide DoS activity in a scalable fashion. Network telescopes however, cannot detect certain types of attacks such as those that do not employ random spoofed source IP addresses. For example, reflector attacks [19] where a large number of attackers send packets to listening servers with source IP address spoofed with that of a victim thus generating a flood of responses from these listening servers directed to victim, can not be detected using a network telescope. PCFs on the other hand can easily detect such attacks.

A quick comparison between telescopes and reverse path PCFs is given below, while actual trace driven comparisons between the two schemes are presented later in Section IV-B.3.

- *Reliance on source address spoofing:* Network telescopes can only detect attacks that employ spoofed source addresses. PCFs, on the other hand, do not depend on this feature. Thus PCFs are capable of detecting reflector based attacks, and other attacks using "zombies" that do not employ source address spoofing.
- *Reliance on TCP:* Network telescopes can observe source address spoofed attacks using a wide variety of protocols including TCP, UDP, ICMP as opposed to PCFs that are designed only for TCP.
- *Reserved IP space:* Network telescopes have an inherent trade-off between detection time and the size of the unused private address space they watch: the larger the space, the faster the detection. PCFs on the other hand, do not require any reserved IP space.
- *Geographical scope:* Network telescopes receive the aggregate traffic sent from any source under attack that uses address spoofing towards the private IP space reserved for it. Hence, network telescopes have much wider geographical scope in contrast to PCFs that have only local scope.

As we can see from the comparison, PCFs can be a viable and scalable complementary solution for general TCP-imbalance detection that passes through the detection device. It can be widely deployed without any issues. On the other hand, telescopes are invaluable for the global detection of DoS attacks (based on fake source addresses) at a *few* monitoring points. A combination of network telescopes and PCFs can scalably detect partial completion attacks in the network.

E. Deployment and spoofability of PCFs

In this section, we apply PCFs to partial completion attacks and scanning attacks and discuss their spoofability properties.

Partial completion detection: Since spoofing is dependent on the particular instance of deployment, we first discuss briefly various network deployment options of interest, and discuss the spoof resistance of each option. Figure 2 shows these deployment options.

- *Near sources.* A PCF(SYN, FIN, <SIP>) can easily identify those sources that are generating many SYN packets but no FIN packets. This deployment scenario could potentially be used to monitor sources of attacks. However this is subject to spoofing using a variety of mechanisms. Sample spoofing mechanisms include the sending of fake FIN packets, the use of carefully crafted TTLs for FIN messages that only crosses the monitor but never reaches the victim. Attempting to defend against the dazzling variety of spoofing options is a virtual impossibility; we choose instead to finesse this issue by espousing reverse path PCFs because it is harder for the attacker to control the reverse path.
- *Outgoing/incoming edge of an ISP or customer network*: A PCF(SYN,FIN, <DIP,DP>) could be deployed at the edge of an an ISP network or a large customer network to detect attacks to destinations that originate or are directed towards their domain. This constitutes the *forward path* of the attack (which is controlled by the attacker). The attacker can again spoof using either FINs before SYNs, or using FINs with TTLs that expire early. Note that this particular spoofability arises from uni-directionality of traffic. If we assume a bi-directional model, then PCF (incoming SYN, outgoing FIN, <DIP,DP>) is *not spoofable* since it is hard to forge a packet in the opposite direction.
- *Outgoing edge of an ISP or customer network*: PCF (SYN, FIN, <SIP,SP>) can detect destinations in the domain under attack. This follows the fact that a victim under attack generates several SYN-ACK packets but no corresponding FIN packets (since connection is never really established, even if it did doesn't terminate). This is *spoof-resistant* [5] since the attacker cannot force the victim to generate a FIN packet without legally tearing down the connection. We call this the *reverse path* of attack. Later in Section IV-B.3 we focus on this particular deployment of PCF to scalably detect backscatter in the network
- *Near destinations:* This case is similar to that of the incoming edge of an ISP or customer network and hence spoofable in uni-directional but not spoofable in bi-directional detection model.

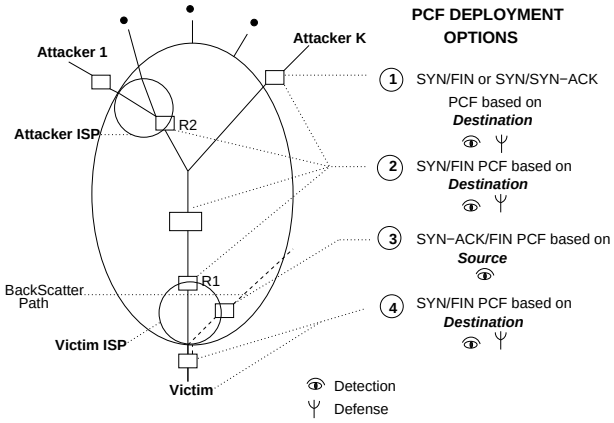


Fig. 2. Deployment options.

[5]More accurately, to spoof this scheme the attacker will need to have machines under its control *in the victim domain* that can send fake FIN packets. If the attacker already has such power, it is unclear why the attacker needs to stop at simple DoS attacks of victims.

Fundamentally, *any approach which observes only the forward direction (active SYN-FIN(ACK)) of TCP traffic (as is often the case in the network) to a particular destination is susceptible to spoofing.* This is because an attacker can always manufacture packets in the forward path (some of which can die before they reach the victim via low TTLs etc.) that look exactly like valid TCP connections.

Scanning detection: PCF(SYN,FIN,<SIP>) can detect attacks that involve scanning hosts for vulnerabilities. However, a clever scanning approach can easily spoof the PCF (or any other full state approach) that only sees one direction of the traffic. On the other hand, by assuming a bidirectional model (where PCF sees both directions of traffic, usually true at the edge), we can scalably detect hosts that are scanning for other hosts.

In the bi-directional model, the PCF increments for the SYN packet going out and a FIN packet coming in to detect scanners inside the network. In order to detect a scanner outside the network, PCF correlates between in-bound SYN packet and out-bound FIN packet. Note that this deployment of PCF is not susceptible to spoofing since, the originator of the scan cannot force a non-existent host to generate a FIN packet in the opposite direction. Similarly, PCF(SYN, FIN, <SIP, DP>) can detect horizontal scans for specific ports in a scalable fashion too and can be made *spoof-resistant*.

IV. Measurement on real traces

We evaluated PCFs on a set of real traces that belongs to two ISPs, ISP-A and B, we obtained from CAIDA [30]. The traces are named ISP-A Dir-0, ISP-A Dir-1, ISP-B Dir-0, ISP-B Dir-1 corresponding to two different directions of traffic. All these links have OC-48 capacity.

The main aim of these experiments is two-fold. First, we want to validate the theoretical analysis of PCFs when applied in real settings; any deviations from analysis can be used to tune and modify PCFs. Second, we want to gather experience using PCFs to scalably detect attacks (in the categories we restricted ourselves to earlier) on real network traces.

We use the following terminology throughout the evaluation section. First, a "flow" is any unique tuple over a subset of packet contents. For example, if the aggregation is over fields SIP and SP, all packets that bear (say) <192.168.10.1, 80> as the source IP, port pairs (perhaps with different destination IP addresses or ports) is termed a flow. Note that the word flow in the usual sense is often referred to as the 5-tuple; for the lack of a better word, we used this somewhat loaded term in a different way.

A flow is said to be *correctly identified* if both the full-state approach and PCFs identify that the flow has a value greater than the threshold. A flow on the other hand is a *false positive* if PCF indicates that the flow has a value greater than the threshold but using the full-state approach we find that the flow does not have a value greater than the threshold. Similarly, a *false negative* refers to those flows that have not been identified using PCFs but were detected using a full-state approach.

A. Part I: Validation and tuning of the model

Recall our assumption that the probability of a SYN and FIN is equal to 0.5. How accurate is it really ? There are a set of issues that warrant adjustment to the model. In this section, we briefly discuss these issues and discuss how to correctly adjust the behavior of PCFs for real traffic.

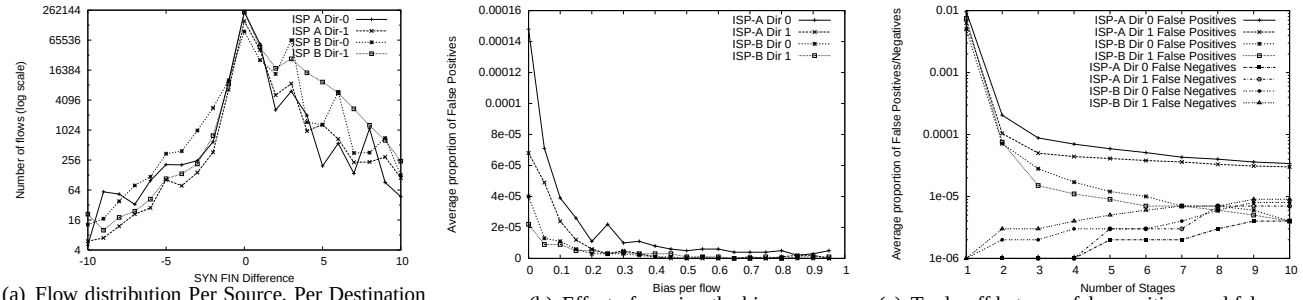
In the first part of our results, we validate and tune the model to possible deviations from these assumptions. These potential idiosyncrasies in real traffic determines the operating range and usefulness of PCFs. Unless explicitly stated otherwise, we used a measurement interval of one minute, and used a total of 5000 buckets for our evaluations. Since attack detection time is directly dependent on the measurement interval, we choose a minute for the measurement interval. The number of buckets is usually dependent on the memory constraints on the router. To implement PCFs with 5000 buckets and 3 stages, we need about 480 Kbits of memory, that can be easily accommodated in today's routers.

Q1. How does the asymmetry between SYNs and FINs affect the results? In the theoretical model, we assumed that the expected value of each counter is 0 since the packet can be a SYN or a FIN with equal probability. However, in reality, this is not true. The expected value of the counters is close to zero but not quite zero. This is primarily due to the fact that there are potentially a larger number of SYN retransmissions than that of FIN retransmissions in a connection. This is plausibly because the round-trip delay has not been calculated when a SYN is sent (as opposed to a fairly accurate value when the FIN is sent); thus estimates that are too low will lead to more unnecessary SYN retransmissions.

Also, note that some connections are torn down by RSTs as opposed to FIN packets. One way to take into account this bias is to decrement for the RSTs along with FINs. We decided however to capture all these effects into a single parameter called "bias". This positive bias accounts for all irregularities that arise from SYN-FIN differences and can be appropriately set. The problem with this positive bias is that, even a reasonable number of these flows with the positive bias can now aggregate to look like a bad bucket. Hence, in actual practice, the mean μ has to be set to a positive value. The analysis however remains the same, except that the threshold for detection has to be tuned based on the positive bias.

Figure 3(a) shows the number of <DIP,DP> pairs (any other kind yields similar results) for different SYN-FIN difference values ranging from -10 to 10. We chose 10 as our cut-off point since beyond this value, it is reasonable to expect that they are outliers. From Figure 3(a), we can observe that the area under the right half of graph ($x = 0$ to $x = 10$) is higher than that on the left side ($x = -10$ to $x = 0$). The amount of positive bias that needs to applied to each counter per flow is equal to the weighted mean of the SYN-FIN differences.

Suppose all flows (e.g <DIP,DP> pairs) on an average have a positive bias of x (average SYN-FIN difference for all flows = x). Then we have to modify the threshold to take into account this positive bias. In our counter model, suppose f were the total number of flows, and b be the total number of buckets. Then, the expected number of flows that map to a particular bucket is going to be $\frac{f}{b}$. But since, these flows have a genuine



(a) Flow distribution Per Source, Per Destination of SYN-FIN Counters.

(b) Effect of varying the bias

(c) Trade-off between false positives and false negatives with different number of stages

Fig. 3. PCF validation and tuning. We first characterize benign behavior to tune the model. We later, show how to select number of stages based on the trade-off between false positives and false negatives.

positive bias, the expected value of each counter is $\mu = \frac{f}{b} \cdot x$.

Therefore, instead of an expected value of 0 used in the Normal approximation, if we use the new expected value (the standard deviation remains same), the probabilities would still remain the same. In other words,

$$Pr\left[a \leq \frac{X - \frac{f}{b} \cdot x}{\sigma \cdot \sqrt{n}} \leq b\right] = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-z^2/2} dz$$

Now again, for $a = 3,\ b = -3,\ \mu = 1, \sigma = 1$,

$$Pr\left[|X| \leq \frac{f}{b}x + 3\sqrt{n}\right] = 0.9987$$

Our new threshold value has to be modified to include this positive bias, resulting in a new term equal to $\frac{f}{b}x + 6\sqrt{n}$. Recall that f is the number of flows, and n is the total number of events per bucket (SYN-FIN packets in the case of partial completion attacks).

The theory above suggests that the model can be adjusted to account for this bias we observe in real life. In the next experiment, we evaluate the significance of this bias. Figure 3(b) shows the relationship between the bias and the false positive ratio. For the traces we used in this figure, we can see that a reasonable bias of 0.5 reduces the number of false positives dramatically, although we calculated empirically that the actual bias for the ISP A and B traces was less than 1. Choosing a high value of the bias increases the threshold but the operating range of the filter becomes smaller.

Summary: We see that for some traces more than the others, bias effects the number of false positives significantly. Hence, this factor must be appropriately set (we assume henceforth a threshold of 0.5) to a reasonable value. For the set of traces we considered, a bias value of 0.5 worked well in all cases.

Q2. How many stages are necessary in PCF ? In this experiment, we evaluate empirically the number of stages required in the PCF to sustain a reasonable false positive rate. Figure 3(c) shows the variation of false positives and negatives with increase in the number of stages from 1 to 10. Again, the y-axis represents the average proportion of false positives/negatives to the total number of flows. From the figure as well as our theoretical analysis, 3 stages represents a good trade-off to reduce the false positives while keeping the false negatives to a minimum.

Beyond 3 stages, we can see that there is really little gain in the false positive rate. In addition, false negatives begin to increase. In fact, from the figure, 3 stages represent the "knee" of the false positives curve with diminishing returns from adding more stages. Henceforth, we operate with 3 stages for the rest of the paper for our evaluations.

Q4. Why is the false positive rate still high? A curious reader can immediately notice that in the earlier plots, even for the theoretical threshold, the false positive rate is much higher that what we have predicted using our analysis in Section III-B, part 1. This is due to the fact that, in the presence of a large number of flows that have a high imbalance in the counters (bad flows), the number of false positives increases as we have analyzed before in Section III-B, part 3. In fact, we found that the number of flows that were anomalous was close to 100. Using our analysis in Section III-B, the estimated number of false positives in proportion to the total number of flows is about $(100/5000)^3$ which is close to 10^{-5}. We can see that the false positive proportion is close to this value in the plots.

One way to reduce the false positives in this case is to add an extra threshold (beyond the theoretical threshold value including that calculated taking into account bias). This can potentially restrict the number of attacks of interest to a low value (only the heavier attacks will be identified – in fact maybe desirable) but it also reduces the number of flows that will monitored. We can even adaptively vary the threshold based on the amount of flow memory available. In [23], the authors suggest a heuristic algorithm that adjusts the threshold based on the amount of flow memory in the context of heavy hitter detection using multistage filters. We can use a similar algorithm so that we can operate within the memory constraints of the router.

Summary: The false positive rate is higher in the presence of a large number of bad flows. A way to decrease the false positive rate to a reasonable value is to increase the threshold beyond our current theoretical value. This remains a compromise between the available memory resources, expected number of attacks and so on.

Q5. How big should the measurement interval be ? The next important issue is how to choose the measurement interval. A plausible first guess could be to choose a very small measurement interval. In any scheme, a small measurement interval, while facilitating fast detection, lacks a clear signature to infer an attack. A large measurement interval on the other

hand increases the time of detection. Usually the choice of the measurement interval is dependent on the scenario of application. We choose *one minute* as our measurement interval for all our experiments. However, we hasten to add that this can be varied depending on the situation. Note that this implies that we require multiple copies if we want to simultaneously accommodate multiple applications with different requirements. However, typically PCFs need to operate with the smallest interval among requirements across all applications. Another option is to divide the set of buckets into multiple classes, with each class of buckets operating at different measurement intervals. For the purposes of this paper, however, we operate with one measurement interval and one threshold.

B. Part II : Experience with PCFs:

In this section, we present our experience (attack identification and characterization) with PCFs over real traces. We also compare our scheme with the Network telescopes approach for scalable monitoring, followed by detection of scanning activity in our traces.

1) Experience with PCF on ISP-A OC-48 link: Through this experiment, we discuss briefly our experience with PCFs over large time periods (1 day). The main aim of this experiment is to discuss how PCFs operate over large time periods along with a general characterization of SYN floods observed in such periods. We set the PCF threshold to 150 (theoretical threshold is 60) , which allowed us to identify attacks of size greater than 2.5 SYNs per second. Recall once again that we use the term "flow" to represent any unique tuple over a subset of packet contents. We only had access to ISP-A's Direction 0 trace for an entire day. The total number of unique destination IP addresses in the full trace was about 5.16 Million over the entire day. The total number of unique <DIP, DPort> pairs was over 30.36 Million. Similarly, the number of unique sources, <SIP, SPort> pairs were found to be approximately 1.9 Million and 30.9 Million respectively. In all, we have observed a total of 517 attack flows that were detected by PCF in the forward path. We had 6 false positives and 0 false negatives reflecting the efficacy of PCFs. About 40% of the attacks found lasted for less than a minute and 85% less than 10 minutes. A similar observation has been made by the authors in [31].

2) Attack characterization using PCFs on other traces: We now present the characterization of partial completion attacks that we detected using PCFs on both directions of ISP-A and B one hour traces. Table I shows the number of attacks identified for each trace and the corresponding false positives and false negatives using both forward path and reverse path PCFs. As before, we have used a PCF threshold of 150 to identify these attacks.

For brevity, we only show the characterization of the attack flows obtained using the reverse path PCFs. Figure 4 consists of CDF plots that show the relative distribution of the attack flows PCFs identified in both directions. From the Figure, we observe that in ISP-A traces, 30% of the attacks identified lasted through the complete duration of the trace (60 minutes). We have observed similar behavior in the forward path (not shown here) as well. However, for the ISP-B traces, we observed that 85% of the attacks last for less than 10 seconds. This points out that the attack duration varies depending on the particular location of the monitor. In Figure 4, we also characterize the rate and total size of the attack. About 20% of attacks discovered in ISP-A trace had more than 1200 SYNs per minute, roughly 20 SYNs per second constituting much heavier attacks. Similar to the attack duration, we observed that the characteristics of the total attack volume varied significantly across traces.

3) Comparison of DoS detection with state-of-the-art network telescopes: As promised earlier in Section III-D, we conducted an experiment that provides empirical comparison between PCFs and backscatter analysis using network telescopes. We used a trace, BACKSCATTER obtained on Jan 22nd, 2004 at 2pm from the CAIDA network telescope. Simultaneously, we obtained a trace from ISP A Direction 0 and Direction 1. The main intuition in comparison with the backscatter approach is that PCFs should be able to detect sources that are generating a lot of SYN-ACK packets in response to a SYN-flood and hence should be aggregated based on sources.

On ISP-A direction 0 and 1, PCFs detected about 19 and 33 flows in the reverse path as shown in Table I. At the same time, the telescopes detected about 776,990 different flows. However, only three destinations were detected by both PCFs and telescopes, that too in only one direction of the traffic. We believe this occurred because the scope of any one router especially if it is a transit router (as in our case) is much smaller than that of the total backscatter. Unfortunately, we cannot verify this hypothesis with the traces available to us. For one of the flows common to both backscatter and PCFs, PCF received a huge number of SYN packets while backscatter detected only one packet. We conjecture that this attack did not employ enough spoofing hence was not detected by the backscatter telescope.

We also found that a large percentage of attacks that we observed in the router trace were not observed by the telescope. This could be due to either of two reasons. One is that the attacks were mostly reflector attacks which did not employ spoofing as a method to bombard the victim. In these cases, the telescope cannot detect any backscatter. The other reason could be the presence of DoS attacks which are single or multiple source attacks that do not employ spoofing. We have manually verified the existence of both in our trace.

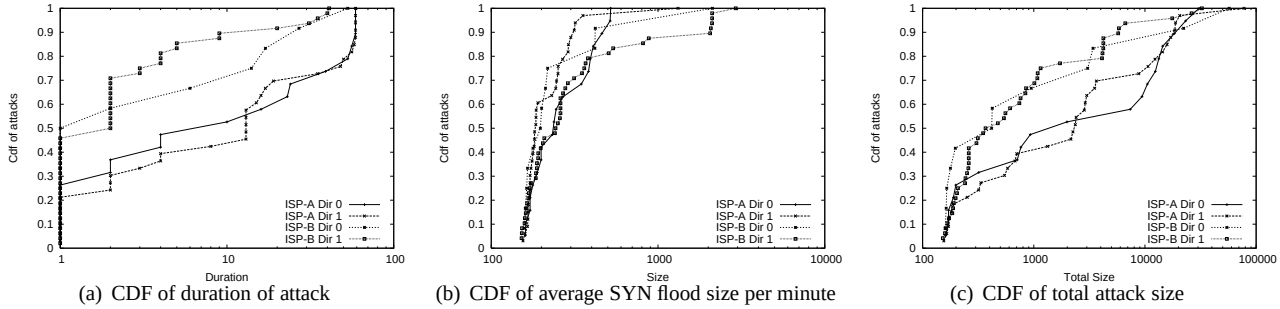
Summary: Comparison of the reverse path PCF reveals that both telescopes and PCFs can identify attacks the other can not. The telescope observes worldwide DoS activity that primarily employs packet spoofing. Any attacks such as reflector attack, or a DDoS attack with a set of zombies with no spoofing never reach the telescope. On the other hand, reverse path PCF is oblivious to address spoofing but suffers from a much smaller scope. The intersection set is rather small, making PCFs *a complementary solution* to backscatter for scalable attack monitoring.

4) Scanning Detection: In this section, we use PCFs to scalably detect scanning in the network. We conducted two sets of experiments to validate our findings – one based on aggregation using source IP Addresses, and the other using source IP Address, destination Port combinations as discussed in Section II. In order to validate whether the sources we have found are indeed scanners, we counted the number of unique destinations found. Any source which generated SYNs but no FINs for more

Trace Code	Attacks identified		False Positives		False Negatives	
	Fw. Path	Rev. Path	Fw. Path	Rev. Path	Fw. Path	Rev. Path
ISP-A Dir-0	68	19	2	0	1	0
ISP-A Dir-1	39	33	1	1	0	0
ISP-B Dir-0	34	12	0	0	0	0
ISP-B Dir-1	41	47	0	0	0	1

TABLE I

SUMMARY OF FLOWS WE IDENTIFIED USING FORWARD AND REVERSE PATH PCFS ON VARIOUS TRACES USING A THRESHOLD OF 150.



(a) CDF of duration of attack (b) CDF of average SYN flood size per minute (c) CDF of total attack size

Fig. 4. CDF of the duration, average attack size, total attack size of attacks detected using PCFs on different traces in the *reverse path*.

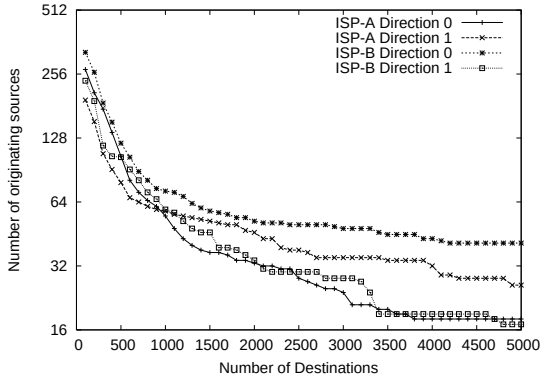


Fig. 5. Port scan detection using PCFs. Figure plots sources identified by PCF that <SIP> alone. The y-axis represents the number of scan sources identified by PCFs that have more than x destinations (for varying x).

than a particular number of destinations, we posit as being a true port scan for our comparison. The question at hand is whether PCFs can scalably detect such scanners. Note that PCFs based on <SIP>, fundamentally observe sources which are generating too many failed connections generating SYNs but no FINs. This counting of destinations is only a step we can use to ascertain ourselves that these sources are indeed scanners.

This approach combines two traditional approaches of scan detection – counting events in a given time interval [32], [1], and observing failed connections [33], [10]. PCFs identifies sources that generate many SYNs but no corresponding FINs (corresponds to failed connections) in a given time interval. Note that in situations where there are a limited number of end-hosts such as in an enterprise, heavy weight schemes such as [34] might be practical. PCFs allow scalable and efficient detection of scans in situations where scalability is important (hence heavy weight approaches are impractical).

In order to establish that the sources identified by PCFs were really portscans, we plot the number of identified sources with increasing number of destinations in Figure 5. In other words, on the x-axis we vary the number of destinations in steps of 100, and we plot on the y-axis the number of sources identified by PCF that have sent SYN packets (but no corresponding FIN) to more than x destinations. From these Figures, we can observe that a large number of the sources that have been identified by PCFs have failed connections to more than 500 destinations – portscans.

For brevity, we only showed the results using a PCF on <SIP>. Results obtained using <SIP,DP> for horizontal portscans were similar.

V. IMPLEMENTATION

We now discuss an architecture for implementing PCFs in high end routers. PCFs can be implemented on the line card as a dedicated ASIC that obtains the relevant fields of the packet such as source IP address, source port, destination IP address, destination port, protocol, TCP flags and so on. This process does not require too many modifications, since routers already obtain destination IP address from the packet header. PCF then uses these fields to simultaneously update the counters in different stages and identifies those packets that map to buckets whose counter value is greater than the threshold. The mechanism then involves PCFs triggering a rule in the access control list (ACL) in the forwarding path that will allow the capture of the packets belonging to that particular flow to obtain further forensics.

Upon detection of an attack, the attack stream can then be diverted through a special port in the switch by configuring the ACL appropriately. This stream can now be examined further using other techniques such as hop-count based filtering or active probing using RST-cookies [35] etc., in the network to validate the authenticity of the packets. There are specially

available vendor boxes such as that by Arbor Networks [8] or Riverhead Networks [36] that can further characterize the attack stream. Basically, PCFs can be used as efficient trigger for full-flow forensics without having to resort to expensive mechanisms in the fast forwarding path of the router.

Hash Function : We used the Carter-Wegman H3 [37] hash function for PCF Evaluation. H3 hash function can be easily implemented in hardware as it consists primarily of XOR gates proportional to the number of output bits. To avoid synchronization of flows that get mapped to the same set of buckets in each and every memory interval, we can vary the hash function every few measurement intervals. Any long enough malicious flow might get lucky a few times (by mapping to some other bucket that aggregates with this flow to make it look benign). But, if the hash function is varied every now and then, this synchronization can be easily mended.

VI. Related work

The general notion of scalable attack detection has been addressed independent of our work, recently by Yaar et.al in [38]. However, their work requires routers to implement marking along with some header changes to support marking of packets. In [39], the authors also propose a Path Identification scheme for efficient identification of the sources of DDoS attacks. Their work builds on other traceback related schemes (see [39], [38] for further references on Traceback).

In [9] and [40], the authors propose simple stateless SYN-FIN and SYN-SYN/ACK counters in the last hop IDS solutions to detect the presence of SYN flooding presence. While their approach does a preliminary level of aggregation, it does not aggregate across destinations (or sources). They also do not consider the issue of spoofing or suggest solutions to this problem.

For SYN flooding defense, there are several end-host solutions that have been proposed before. These include SYN-cookies [41] and SYN-cache [42] that have been widely deployed. "Backscatter analysis" was proposed in [6]. This technique uses the reverse path attack properties of SYN-floods to infer DoS activity around the world. Since it is relevant to our work, we provided a comparison earlier in the paper.

Vendor based solutions such as SynKill [43], Netscreen [44] or free open source IDS tools such as Bro [10], Snort [1] can be used to detect SYN floods, port scans, but they (as far we can ascertain) employ per-flow state. A clever hop-count based method to detect spoofing in general has been proposed in [45]. Here, the main intuition is that spoofed packets typically have a wrong Time-to-Live (TTL) value and hence should be identified by a previously detected TTL value for a particular IP address. This technique is fairly resilient to spoofing. MULTOPS [20] is a data-structure maintained by each network device that detects bandwidth attacks by the significant, disproportional imbalance between packet rates going to and coming from the victim or attacker.

Most scan detection techniques [32], [1] in the literature are based on detecting N events in T seconds. Another approach [33], [10] relies on failed connections as a better indicator of a scan. Leckie et.al [46] use probabilistic approaches to estimate the degree to which a given local IP address is unusual. SPICE [47] is an offline analysis algorithm to detect stealthy scans (scans which are of low rate) and cannot be performed scalably in the network. A recent paper by Jung et.al [34] apply threshold based random walks for fast portscan detection. The need to track for each remote host the different local hosts to which it has connected to makes the scheme unscalable.

VII. Conclusions

It appears to be widely perceived that detecting intrusions scalably within the network is a bad idea. Unfortunately, that causes security devices to choose between performance (which requires low memory) and completeness (which appears to require per-flow state). This paper is a gentle first step towards suggesting that this tradeoff may not be as Draconian as is commonly thought. While the general problem is still very hard (and indeed for attacks such as evasion attacks, we believe that aggregated solutions cannot work without causing unacceptably high false positives), our paper shows some progress for bandwidth-based and partial completion DoS attacks, and scan-based attacks including worms.

This is fortunate because market researchers [48] have already begun to warn that increases in total ownership costs for end node and edge solutions require the network to play a proportionately larger share in detecting and combating network intrusions. This paper explores this possibility in the specific context of DoS attacks and scan attacks. While we have not harped on this point, doing DoS detection in the network also finesses the need for traceback and/or manual intervention, and allows enterprise networks and ISPs to automatically filter out attacks before they enter (or leave) their networks.

More fundamental than the specific techniques discussed in this paper is the general question of scalable behavior-based detection of attacks within the network. We believe this question is interesting because many other network functions (forwarding, classification, QoS) have already received considerable attention in the research and product literature, and solutions that scale to 40 Gbps already exist. As security functions become more prevalent in the edge first and then the core, it is natural to expect the same attention to be paid to scalable security solutions.

More than just introducing the question and suggesting a specific mechanism for some problems, our paper shows that the issues of *behavioral aliasing* and *spoofing* are key questions that must be addressed in any scalable solution, even if the only response is to simply ignore the problem. For example, it may be reasonable to ignore spoofing until the bar is raised. These two provide a simple lens to view existing and future work in attack detection, and can perhaps suggest new solutions to an even broader class of attacks.

References

[1] Martin Roesch, "Snort," `http://www.snort.org`.

[2] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron, "A signal analysis of network traffic anomalies," in *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Nov. 2002.

[3] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proceedings of the conference on Internet measurement conference*. 2003, pp. 234–247, ACM Press.

[4] Stuart J. Staniford, "Containment of scanning worms in enterprise networks," in *Journal of Computer Security*, Nov. 2003.

[5] Forescout Technologies, ," http://www.forescout.com.

[6] David Moore, Geoffery Voelker, and Stefan Savage, "Inferring internet denial of service activity," in *USENIX Security Symposium*, 2001.

[7] Mazu Networks, ," http://www.mazu.com.

[8] Arbor Networks, ," http://www.arbornetworks.com.

[9] H. Wang, D. Zhang, and K. Shin, "Detecting syn flooding attacks," in *IEEE INFOCOM*, 2002.

[10] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks, 31(23-24), pp. 2435-2463*, Dec. 1999.

[11] Kirill Levchenko, Ramamohan Paturi, and George Varghese, "On the difficulty of scalably detecting network attacks," in *Proceedings of the ACM Conference on Computer and Communications Security, Washington, D.C.*, Oct. 2004.

[12] Robert Keyes, "The Naptha DoS Vulnerabilities," http://razor.bindview.com/publish/advisories/adv_NAPTHA.html.

[13] Nicholas Weaver, Vern Paxson, Stuart Staniford, and R. Cunningham, "A taxonomy of computer worms," in *Proceedings of the ACM Workshop of Rapid Malcode (WORM)*, 2003.

[14] Stuart Staniford, Vern Paxson, and Nicholas Weaver, "How to 0wn the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*, Aug. 2002.

[15] "MyDoom.B Virus," http://www.us-cert.gov/cas/alerts/SA04-028A.html.

[16] "CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL," http://www.cert.org/advisories/CA-2001-19.html.

[17] "CERT Advisory CA-2001-26 Nimda Worm," http://www.cert.org/advisories/CA-2001-26.html.

[18] "CERT. Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks," http://www.cert.org/advisories/CA-1998-01.html.

[19] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," in *Computer Communication Review 31(3)*, July 2001.

[20] Thomer M. Gill and Massimiliano Poletto, "MULTOPS: a data-structure for bandwidth attack detection," in *USENIX Security Symposium*, 2001.

[21] M. Datar and S. Muthukrishnan, "Estimating rarity and similarity over data stream windows," Technical report, 2001-21,, DIMACS, Nov. 2001.

[22] A.C Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M.J Strauss, "Quicksand : Quick summary and analysis of network data," Technical report, 2001-43, DIMACS, Nov. 2001.

[23] Cristian Estan and George Varghese, "New directions in traffic measurement and accounting," in *ACM SIGCOMM*, Aug. 2002.

[24] Cristian Estan and George Varghese, "Autofocus : A tool for automatic traffic analysis," in *Proceedings of ACM SIGCOMM*, 2003.

[25] "Cisco NetFlow," http://www.cisco.com/warp/public/732/Tech/netflow.

[26] Burton H. Bloom, "Space/time tradeoffs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[27] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the constancy of internet path properties," in *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[28] Richard J. Larsen and Morris L. Marx, *An Introduction to Mathematical Statistics and Its Applications*, Prentice Hall, Upper Saddle River, NJ 07458, 2001.

[29] Fyodor, "NMap," http://www.insecure.org/nmap.

[30] Cooperative Association for Internet Data Analysis, "http://www.caida.org," .

[31] Alefiya Hussain, John Heidemann, and Christos Papadopoulos, "A framework for classifying denial of service attacks," in *ACM SIGCOMM*, Aug. 2003.

[32] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J.Wood, and D.Wolber, "A network security monitor," in *Proc. IEEE Symposium on Research in Security and Privacy*, 1990, pp. 296–304.

[33] S. Robertson, E.V. Siegel, M. Miller, and S.J Stolfo, "Surveillance detection in high bandwidth environments," in *Proceedings of the 2003 DARPA DISCEX III Conference*, Apr. 2003, pp. 229–238.

[34] Jaeyeon Jung, Vern Paxson, Arthur Berger, and Hari Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proceedings of IEEE Symposium on Security and Privacy*, 2004.

[35] E. Shenk, "Another new thought on dealing with syn flooding," http://www.wcug.wwu.edu/lists/netdev/199609/msg00171.html, Sept. 1996.

[36] Riverhead Networks, ," http://www.riverhead.com.

[37] Larry Carter and Mark N. Wegman, "Universal classes of hash functions," *J. Comput. Syst. Sci.*, vol. 18, no. 2, pp. 143–154, 1979.

[38] Abraham Yaar, Adrian Perrig, and Dawn Song, "Siff: A stateless internet flow filter to mitigate ddos flooding attacks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[39] Abraham Yaar, Adrian Perrig, and Dawn Song, "Pi: A path identification mechansim to defend against ddos attacks," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[40] H. Wang, D. Zhang, and K. Shin, "Syn-dog: Sniffing syn flooding sources," in *IEEE ICDCS*, Vienna, Austria, 2002.

[41] Dan J. Bernstein, "SYN cookies," http://cr.yp.to/syncookies.html, 1997.

[42] J. Lemon, "Resisting syn flooding dos attacks with a syn cache," in *Proceedings of USENIX BSDCon'2002*, Feb. 2002.

[43] C.L Schuba, I.V Krsul, M.G Kuhn, E.H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proceedings of IEEE Symposium on Security and Privacy*, May 1997.

[44] "Netscreen Technologies," http://www.netscreen.com.

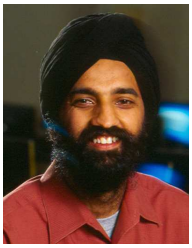
[45] Cheng Jin, Haining Wang, and Kang G. Shin, "Hop-count filtering: An effective defense against spoofed ddos traffic," in *ACM Conference on Computer and Communications Security (CCS)*, Oct. 2003.

[46] C. Leckie and R. Kotagiri, "A probabilistic approach to detecting network scans," in *Proceedings of the Eight IEEE Network Operations and Management Symposium*, Apr. 2002.

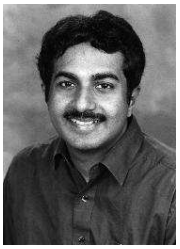
[47] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," in *In Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000.

[48] J. Pescatore, M. Easley, and R. Stiennon, "Network security platforms will transform security markets," http://www.techrepublic.com/article.jhtml?id=r00220021223jdt01.htm&src=bc, Dec. 2002.

Ramana Rao Kompella (S'03, ACM S'03) is a fourth year Ph.D student at University of California, San Diego. His main research interests include fault-management in IP networks, scalable algorithms and architectures for high speed switches and routers, and scheduling in wireless networks. During his Ph.D, he worked at ATT Labs – Research for two summers (2004 and 2005) to devise patent pending mechanisms to localize IP and MPLS layer failures in the network using novel spatial correlation approaches. Prior to joining UCSD, he worked as an ASIC Design and Verification Engineer at Chelsio Communications and as Design Engineer at SwitchOn Networks (acquired by PMC Sierra Inc in 2001) where he helped pioneer algorithms for packet classification and routing lookups. Together with several colleagues, he has six patents (two awarded and four pending), more than 15 research publications in the areas of packet classification, fault localization and wireless networks. He received his B.Tech degree from Indian Institute of Technology (IIT), Bombay in 1999 and M.S degree from Stanford University in 2001.



Sumeet Singh is a Technical Leader at Cisco Systems. His research interests include network security, switching and routing. Previously he was Co-founder and Chief Scientist at NetSift, Inc. and pursuing his PhD in computer science at the University of California, San Diego where he helped to pioneer one of the fastest approaches to packet classificaiton (HyperCuts) and one of the first approaches to automated signature extraction (the EarlyBird system).



George Varghese (M '99/ACM F '99) worked at DEC for several years designing DECNET protocols and products (bridge architecture, Gigaswitch) before obtaining his Ph.D in 1992 from MIT. He worked from 1993-1999 at Washington University. He joined UCSD in 1999, where he currently is a professor of computer science. He won the ONR Young Investigator Award in 1996, and was elected to be a Fellow of the Association for Computing Machinery (ACM) in 2002. Together with colleagues, he has 14 patents awarded in the general field of Network Algorithmics. Several of the algorithms he has helped develop have found their way into commercial systems including Linux (timing wheels), the Cisco GSR (DRR), and Microsoft Windows (IP lookups). He also helped design the lookup engine

for Procket's 40 Gbps forwarding engine. He has written a book on building fast router and endnode implementations called "Network Algorithmics", which was published in December 2004 by Morgan-Kaufman. In May 2004, he co-founded NetSift Inc., where he was the President and CTO. NetSift is now part of Cisco Systems. From Aug 2005 to Aug 2006, he has been working at Cisco Systems to help equip future routers and switches to detect traffic patterns to facilitate traffic measurement and real-time intrusion detection.