# Feasibility of Detecting TCP-SYN Scanning at a Backbone router

Khushboo Shah
Dept. of Electrical Eng.
University of Southern California
Los Angeles, CA 90089
khushboo@usc.edu

Stephan Bohacek
Dept. of Electrical and Comp Eng.
University of Delware
Newark, DE
bohacek@eecis.udel.edu

Andre Broido
Cooperative Assosication for Internet Data Analysis
University of California, San Diego
San Diego, CA 92093
broido@caida.org

***Abstract*— IP address and TCP/UDP port scanning are critical components of many network attacks. Such scanning allows attackers to spread a self-propagating worm or collect detailed information about end-hosts in preparation for attacks. This paper investigates an algorithm to detect TCP-SYN scanning. Special care is taken to detect even stealthy scanners. Attackers are detected by tracking the number of half-open connections generated by each source. However, a source is only tracked until it is clear that it is not currently scanning. The computational load of this method is analyzed and it is found that the method is not computationally intensive and is suitable for online detection of scanning even on backbone links. Specifically, on a backbone link investigated it was found that the total number of sources that needs to be tracked at any one time is at most a few thousand.**

## I. Introduction

Since the early days of the Internet, network attacks have been a difficult problem [1]. As the economy, infrastructure, and society become more dependent on the Internet, network attacks pose a problem of more significance. Recently there have been major self-propagating worm attacks [2], [3], [4] that have caused significant damage in terms of labor hours, lost data, and lost service. Beyond the self-propagating worms, there are numerous other intrusions. For example, attackers may break into a system and set-up a server to distribute copyrighted material, to steal confidential information, or launch attacks on other network resources. The first step of such intrusions is to gain information about the systems to be attacked. This initial step is carried out by a technique known as "scanning", where the attacker sends a large number of packets to different IP addresses in search for vulnerable hosts. This work investigates a method to detect scanning at backbone routers.

Worms such as Blaster and Code Red [3], [4], [2] search for vulnerable hosts by sending packets at high rates to randomly selected IP addresses. If an open port was found and the port supports an application that has an exploit, then the found host is infiltrated. In order to propagate the worm quickly, an infected hosts must scan the address space at a high rate. However, this makes worm propagation very obvious and hence easy to detect and stop. A far more potent way to spread the worm is if the worm itself has a list of all vulnerable hosts. In this case, the worm does not have to perform the noticeable scanning and it can spread at very high rates. One way this can be done is for the worm designer to scan the address space in advance and maintain a database. It is possible to catalogue not only which IPs are active, but to determine the type and version of the OS and which services are offered for each IP [5], [6], [7], [8]. When an exploit becomes available, the attacker could immediately develop a worm that is sent only to the vulnerable hosts in the database. Since this highly virulent worm would spread while producing very few anomalous packets, it would be virtually undetectable.

Since the IP address space only holds merely $2^{32}$ addresses, it is possible to send a TCP-SYN to every IP address in 38 hours over a 10Mbps link. Thus, it is completely reasonable to expect that attackers will collect, catalogue, and distribute detailed maps of all end-hosts. While it is possible to scan the entire address space very fast, an attacker may want to scan more discretely and at a slow rate, where the whole address space is scanned over a period of weeks or even months. Also, the attacker may only scan a fraction of the address space until a vulnerable host is found. This vulnerable host is then infiltrated and the scanning proceeds from this host. Thus, the entire address space can be scanned, but no single host will scan the whole address space. While difficult, the goal of this work is to detect these types of *stealthy scanning*.

There has been investigations on the detection of scanning at firewalls [9], [10]. A firewall detects scanning only if the attacker scans many of the organization's addresses. If the attacker scans addresses at random, the scans will infrequently fall into the organization's address space. The firewall would require a long-term memory to detect such behavior. On the other hand, if the detection is done at a tier-1 or 2 router through which a large portion of the attackers scans pass, even a stealthy scanner can be detected. It has been shown there is a relatively small set of routers through which a large number of end-to-end connections pass [11]. Thus, by sharing information collected at these routers, it may be possible to detect even the most stealthy scanners.

The goal of this paper is to detect scanning at backbone routers. We focus on a common form of scanning known as SYN scanning where the attacker sends TCP-SYN packets

[12], [6]. The method investigated here examines each TCP flow. An alternative method is[2] to sample only a fraction of all flows. however, such methods cannot detect stealthy scanning. The detection scheme investigated here is quite thorough in the sense that the source of each TCP-SYN packet is tracked for at least a short period of time. Thus, even stealthy scanners can be detected. Nonetheless, it is found that when performing this detection on a backbone link that carried data at an average rate of 378 Mbps, the number of hosts that need to be tracked simultaneously is less than a few thousand. Maintaining and searching through a list of a few thousand addresses is within the abilities of today's processors. We make one critical assumption in this analysis: *packets with spoofed source addresses are filtered out or limited in number*.

The remainder of the paper proceeds as follows. The next three sections provide some background. After a brief discussion of the data utilized throughout this investigation, the basics of TCP connection establishment are reviewed followed by some further discussion about the scanning activity found in the data set. Section V begins the discussion of the algorithm. Section VI explores the computational efficiency of this method. Section VII provides some concluding remarks and discusses future work.

## II. Data description

The data used in this work was collected in a network of a US tier-1 Internet Service Provider. The trace was captured by Linux-based monitor with Dag 4.11 network card from the University of Waikato [13] and Endace[14]. It contains 44 bytes of each packet, enough to include the IP and TCP/UDP headers.

The trace analyzed in this paper (D09S in the notation of [15]) was taken on May 7, 2003, 10:00am to 12:00 noon at the Seattle, WA to San Jose CA, SONET OC-48 (2.5 Gbps) link of CAIDA's Backbone 2. This link was 15% utilized with an average bitrate of 378 Mbps. TCP contributes 93.5% packets and 97.8% of all bytes observed in that measurement. Out of those, 42.3 M packets have SYN flag set (7.2% of all TCP packets).

## III. TCP connection establishment

In order to initialize a connection, TCP uses a three-way hand-shake [16].[1] The first step of the connection establishment is when the client sends a TCP packet with the SYN flag set (a TCP-SYN packet). Upon receiving this packet, the server responds with a TCP packet with both the SYN and ACK flags set (a TCP-SYN/ACK packet). When the client receives the TCP-SYN/ACK, it responds with a TCP packet with the ACK flag set (a TCP-ACK packet) or with a TCP data packet that also has the ACK flag set. When the server receives the ACK from the client, the connection establishment phase of TCP is complete.

If the client does not receive a SYN/ACK from the server, it will resend the SYN typically after waiting for 3 seconds. If a SYN/ACK still does not arrive, the client will send another SYN after 6 seconds. This doubling in time continues for a total of 4 or 6 attempts (the exact number of attempts depends on the implementation). In most implementations, the maximum time between SYNs is 64 seconds [16]. Similarly, if the server does not receive an ACK within the time-out period, it will resent the SYN/ACK in the same pattern as the SYNs are sent.

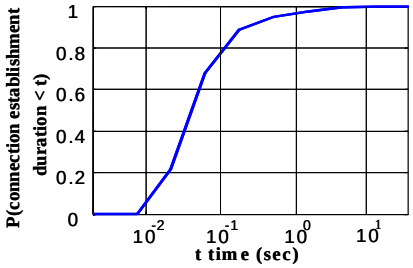


Fig. 1. Distribution of the duration of TCP connection establishment.

At an intermediate router, the TCP connection is assumed to be complete when an ACK packet arrives from the client and is destined for the server. At this point, the router can assume that the destination has responded to the SYN and hence exists[2]. For detecting SYN scanning, the duration of the connection establishment plays a critical role. Figure 1 shows the cumulative distribution of the connection establishment duration. For the link that we observe, it was found that the mean duration of the connection establishment is close to 0.3 seconds. Note that this duration is not simply a function of the round-trip time, but also a function of how long it takes for a server to respond to a SYN.

To give an idea of TCP packets' statistics, we list in Table I the percentage of most frequent types (those with over 1% packets on at least one link direction). An in-depth description of these and other datasets can be found in [15]. The most popular types are SYN, ACK, FIN, SYN-ACK, PUSH, RST (reset) and FIN-PUSH. The last two rows contain sum of the percentages and the overall percentage of TCP packets in observed packets and bytes. Comparison shows that TCP accounts for 85% or more packets and 95% bytes, and that almost every TCP packet has one of the listed types. Among those, ACK and PUSH are most frequent and voluminous. Opening and closing packets (SYN, SYN-ACK, FIN) make up 2-4% of all packets in our data; RST packets vary between 1-2%. These observations are confirmed by our analysis of other traces. The percentage of SYN packets for D09S analyzed here is close to 3%.

## IV. TCP-SYN scanning

In this section we examine the behavior of scanners. First we introduce some definitions. We separate TCP flows in two categories. The first is a normal TCP flow or a *non-pure*

[1]For brevity, we call the opening end *client* and the responding end *server*.

[2]Future work will investigate the situation where the source mimics a normal connection by sending unrequested ACK packets.

TABLE I

TCP FLAGS FIELD. D09, BB2, MAY 2003.

| | Northbound (dir.1) | | Southbound (dir.0) | |
|---|---|---|---|---|
| Flags | % pkt | % byte | % pkt | % byte |
| SYN | 2.57 | 0.20 | 2.91 | 0.27 |
| RST | 1.22 | 0.08 | 1.16 | 0.09 |
| ACK | 56.49 | 67.12 | 58.74 | 71.84 |
| FIN | 2.63 | 0.21 | 3.48 | 0.28 |
| SYN-ACK | 2.06 | 0.16 | 3.86 | 0.34 |
| PUSH | 24.10 | 27.40 | 21.56 | 23.35 |
| FIN-PUSH | 0.47 | 0.44 | 1.23 | 1.55 |
| Sum | 89.53 | 95.61 | 92.94 | 97.71 |
| TCP | 90.08 | 97.10 | 93.47 | 97.76 |

*SYN flow* as described in section III. The second is a flow that only contains TCP-SYN packets. We call such flows *pure SYN flows*. Note that in the case of half open TCP connections and pure SYN flows, the SYN has not been responded to. We call say that such SYNs are *unanswered*. We call the number of *unique* destinations that a sources has sent unanswered SYNs to be the degree of the source. We denote the degree of source $\sigma$ to be $d^{\sigma}$.

The hallmark of a scanning source is that it sends pure SYN flows to a large number of destinations. The data examined here was found to contain many scanners (e.g., 251 sources sent more than 200 pure SYN flows). Figure 2 gives an idea of the number and the behavior of the scanners. Note that the smallest number of non-pure SYN flows shown here is 0. That is indicated by $0$ on y-axis.

Sources that appear in the lower right of the plot are the ones that are mostly likely to be scanning as they send pure SYN flows to a huge number of destinations, but few or no normal TCP connections. Indeed, if the source sends more than 1000 pure SYN flows within two hours, the source is likely scanning. A mode careful analysis by examining the domain name of the addresses, ports, size and frequency of file transmission, etc confirms this conclusion. However, it is not clear whether the sources that send a moderate number of pure SYN flows and non pure SYN flows are scanning. We examined some of those sources by looking at their. We found that some of those sources that merely sent 50 pure SYN flows were likely scanners. Figure 2 shows that it is not trivial to draw a line between scanning sources and normal source. This difficulty remains even when further information such as the non-pure SYN degree is considered. For this reason, the approach here will detect sources that scan a relatively small number of destinations. It is expected that offline methods would perform evaluation of the sources detected by the method presented here.

There are some types of activity that appear to be scanning but are filtered out and not examined here. Specifically, some mail servers will try to open a connection on port 113. However, many firewalls block port 113. Hence, these mail servers will send a large number of pure SYN flows. Similarly, port 25 is often blocked and, as a result, mail servers will send many pure SYN flows to port 25. Also, to search for peers to share files, Gnutella and Kazaa will scan the address space. Thus, along with ports 113 and 25,

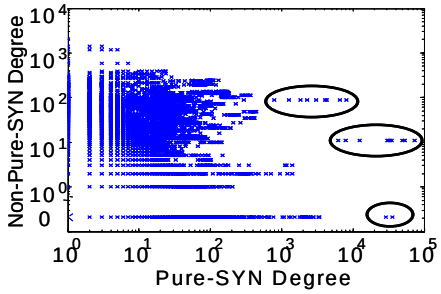


Fig. 2. Pure-SYN degree versus Non-Pure SYN Degree. Clearly, the sources falling inside the circles are scanners. But there are many other scanning sources as well.

flows on port 6346, 6347, and 1214 are filtered out of the data set.

## V. TRACKING SOURCES

An attacker can scan the address space while utilizing a low packet rate. This is especially the case for stealthy scanners that send TCP-SYNs at a slow but steady rate. In order to detect such scanners, it is necessary to collect information about the hosts that send TCP-SYNs. The algorithm used to detect TCP-SYN scanners is as follows.

Assume that a SYN arrives at time $t_0$ and no other SYNs have arrived from this source before this point. We denote $d^{\sigma}(t)$ to be the degree observed $t$ seconds since the source is being tracked. Thus, $d^{\sigma}(t)$ is the number of unique destinations that source $\sigma$ sent unanswered SYNs to during the time interval $(t_0, t_0 + t)$. In the case of a normal TCP connection establishment, $d^{\sigma}(t) = 1$ for $0 \leq t < T$ and $d^{\sigma}(t) = 0$ for $t \geq T$, where an ACK arrived from the source at time $t_0 + T$.

In order for a router to determine $d^{\sigma}$, it must observe and process packets from this source, that is the router must *track* the source. When a packet arrives, it is first determined if the packet is a TCP packet. If so, it is determined if the source is already being tracked. Note that this requires searching through the list of sources that are being tracked. In order to minimize the computational load, it is necessary to minimize the size of this list.

Suppose that the just arrived packet is a SYN packet. If the source is not being tracked, then a new data structure is allocated and the source and destination of the SYN is recorded along with the arrival time of the SYN. If the source is being tracked, then the source's data structure is examined to see if a SYN destined to this destination has been observed. If so, then the time that this SYN arrived is updated. On the other hand, if the data structure does not contain an entry for this destination, a new entry is made which contains the address of the destination.

Now if the packet is a TCP-ACK or a TCP data packet, then the list of tracked sources is also searched for this source. If the source is found to not be tracked, then there is no change to the list of tracked sources. On the other hand, if this source is being tracked and there is an entry

for this destination address, the entry is deleted because the TCP-ACK or data packet indicates that the connection is now completely open. If there are no more entries for this source, the data structure for this source is deleted and the source is no longer tracked until another SYN arrives form this source.

Note that while a connection remains in its half open state, the source is tracked. In the case that a source has sent SYNs that are never answered, i.e., pure SYN flows, then this source is tracked for a longer period of time. As mentioned, it is critical to keep the list of tracked source small. Therefore, the objective is to not track sources of pure SYN flows for too long.

In order to reduce the computational load, it is necessary to limit the number of sources that are being tracked. Thus, it is critical to stop tracking non-scanning sources as soon as possible while ensuring that these sources are indeed not scanning. Since the detection of stealthy scanners is the objective of this work, the goal of not missing scanners takes precedence over concerns of computational load. However, as will be shown in the next section, the impact of detecting even stealthy scanners does not add a significant load to a detector that can detect aggressive scanning.

We will assume that it is not possible for a source to be scanning if the degree of the source that sent pure SYN flows grows at a rate that is less than $R$ for some $R$. Therefore, if a source does not send a SYN in the $1/R$ seconds after a the first SYN was sent of a pure SYN flow, then this source does not need to be tracked until it sends another SYN. In general, source $\sigma$ cannot be currently scanning if $t > d^{\sigma}(t)/R$, where $t$ is the time that has elapsed since this source has been tracked and $d^{\sigma}$ is defined in Section V.

As discussed in Section V, since SYNs may be answered, an unanswered SYN might or might not lead to a pure SYN flow. Thus, the value of $d^{\sigma}(t)$ might decrease over time. If $d^{\sigma}(t) > H(t)$, it does not necessarily mean that the source is scanning. We define $H(T)$ to be the threshold such that if a source sends more than $H(T)$ SYNs within $T$ seconds, then the source is declared to be a scanner. Specifically, we define

$$H(t) = \begin{cases} V \text{ for } t < T \\ V\frac{t}{T} \text{ for } t \geq T \end{cases} . \quad (1)$$

To accommodate this, we define $d_{\tau}^{\sigma}(t)$ as the number of distinct destinations that source $\sigma$ has sent unanswered SYNs to in the time interval $[t_o, t_o + t - \tau]$, where $t_o$ is the time when the source started to be tracked. Thus, $d_0^{\sigma}(t) = d^{\sigma}(t)$ and $d_{64}^{\sigma}(t)$ is the number of pure SYN flows detected from time $t_o$ to time $t_o + t - 64$. Note that SYNs that have not been answered within 64 seconds can be counted as a pure SYN flows since most implementations of TCP give up if a connection is not established within 64 seconds [16]. Thus, we can say

$$\text{source } \sigma \text{ is scanning if } d_{64}^{\sigma}(t) > H(t-64).$$

However, from Figure 1, we see that a large majority of connections are established within 1 second. Thus, a faster, but slightly less accurate detector is to use

$$\text{source } \sigma \text{ is scanning if } d_1^{\sigma}(t) > H(t-1). \quad (2)$$

While there is a possibility of more sophisticated approaches, in this paper we present the results based on (2).

In summary the algorithm is

1) When a SYN arrives, determine if this SYN is for an ongoing pure SYN flow, or a new pure SYN flow.
2) If this SYN is for a new pure SYN flow and the source of this SYN is $\sigma$ where $\sigma$ is not being tracked, then set $d^{\sigma}(0) = 1$ and begin to track $\sigma$. If $\sigma$ is being tracked, then increment $d^{\sigma}(t)$.
3) At periodic moments check if $t > d^{\sigma}(t)/R$. If so, the stop tracking $\sigma$, otherwise continue to track $\sigma$.
4) When a SYN arrives for a tracked flow, check if $d_1^{\sigma}(t) > H(t-1)$. If so, then declare $\sigma$ to be a likely scanner and pass this address and the collected data to the off-line verification by more sophisticated algorithms (e.g. those recognizing signatures of specific scanning tools.)

## VI. Computation load

When tracking sources, the source address of each TCP packet must be compared with the sources of the addresses that are being tracked. Searching through this list of tracked sources is the principal computational load for this method. In this section we examine the length of this list. We will see that the list of tracked sources is surprisingly small indicating that such detection could be performed even on backbone routers.

Recall that searching through an ordered list of length $N$ can be accomplished in $O(\log(N))$ comparisons. In our experiments, a straightforward implementation of a binary tree was utilized. However, if a suitable hashing function is found, then a hash table would likely further reduced the computation. Furthermore, high speed network processors are becoming available. For example, Intel's network processor IXP2800 can process packets and provide 23 billion operations per second. For the link examined, this algorithm would require memory accesses roughly every 100ns, well within the reach of today's processors such as IXP2800.

The list of tracked sources is made up of three types of sources: sources that are establishing normal TCP connections, sources that are not scanning, but send pure SYN flows, and sources that are scanning or likely scanning. We examine these parts of the list in the following sections.

Three approaches to examining the computational load are taken. In Section VI-E, the detection algorithm is tested with the data described in Section II. A second approach is to directly analyze the computational load. A third approach is to develop models of the network traffic and determine the computational load based on the models. These last two approaches are closely related and are presented in the following subsections. Beyond the conclusion that such

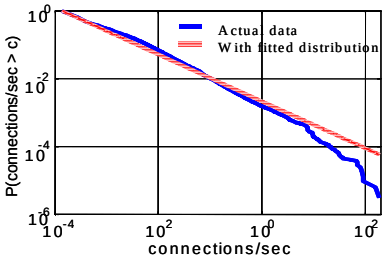


Fig. 3. Complementary distribution of the rate that TCP connections are established by a source. The solid blue curve shows the empirical distribution from observations, while the dashed red curve shows a fitted Pareto distribution.

detection appears to be computationally feasible, are the results that the model of the network traffic are simple and fit well. Similarly, we will see that models for the computational load are simple and fit well.

### *A. Computational load of detecting normal connection establishments*

One might expect that at any particular moment there are a huge number of half open connections. Indeed, the data set investigated here had 38.8 million TCP connections in two hours. Since the source of the half open connections must be tracked, these sources can potentially cause very large lists. However, as will be shown, this is not the case. There are two principal reasons that this list is short. First, the duration of the connection establishment typically lasts around 300 ms. Thus, the number of simultaneously ongoing connection establishment is limited. And the second, there are a few sources that generate a large fraction of the connections. Indeed, some of these hosts establish connections so frequently that multiple establishments from the same source are tracked at the same time. Figure 3 shows the complementary cumulative distribution of the rate at which each source establishes TCP connections. Specifically, it shows the fraction of sources that made connections faster than $c$, where $c$ is the independent variable on the X-axis. We see that there are some sources that made connections with a huge number of destinations. For example, 1 in 10000 sources made connections with 100 or more destinations. These sources contribute to make large fraction of connections.

In order to understand how long the list of tracked sources will be, we model the list as a M/G/$\infty$ queuing system. The entry in the list of tracked sources can hold information about a large number of ongoing connection establishments, i.e. is a infinite number of servers. The rate that customers arrive into this system is the rate at which the source attempts to establish connections. The service time in the system is the time it takes to establish a TCP connection. Hence, when all queues in the system are empty, there is no entry in the list for the source. Thus, the fraction of time that this source is being tracked is the same as this fraction of time that this infinite queue system is not empty. To get an estimate of this number, we assume that the rate of connection arrivals is a Poisson process. In this case, the results of M/G/$\infty$ queueing system yield that the fraction of time that the source is not tracked is

$$P\left(\begin{array}{c|c}\text{source } \sigma \text{ not} & \text{average time between connection} \\ \text{being tracked} & \text{establishments for source } \sigma\end{array}\right) = \exp\left(-\frac{1/\left(\begin{array}{c}\text{average time between connection} \\ \text{establishments for source } \sigma\end{array}\right)}{1/\left(\begin{array}{c}\text{average duration of a TCP} \\ \text{connection establishment}\end{array}\right)}\right).$$

We denote the average duration of connection establishment as $\bar{C}$ and, as discussed in Section III, $\bar{C}$ was found to be 296 ms.

Now suppose that over a time interval of length $T$, we find that a source has established $Q$ connections, yielding an average time between connections of $Q/T$. Also, during this same time period, there were $N_{Q,T}$ such sources observed. Such sources yield a list with average length of $\left(1-e^{-\frac{Q\bar{C}}{T}}\right)N_{Q,T}$. The total list size is

$$\text{Normal connection list length } = \sum_{Q}\left(1-e^{-\frac{Q\bar{C}}{T}}\right)N_{Q,T} \tag{3}$$

For the link examined, this value came to be 759.33.

In the limit as $T\to\infty$, the above can be written as

$$\text{Normal connection list length } = \int_{\beta}^{\infty}\left(1-e^{-\bar{C}v}\right)\bar{N}\phi\left(v\right)dv, \tag{4}$$

where $v$ is the connection establishment rate, $\bar{N}$ is the total number of source addresses that pass through the router and $\phi\left(v\right)$ is the density[3] of nodes with connection establishment rate $v$.

Figure 3 shows a fitted Pareto density for $\phi$ with exponent $\alpha = 0.69$ and a minimum value of $\beta = 1.35\times10^{-4}$:

$$\phi_{\alpha}\left(v\right) = \frac{\alpha\beta^{a}}{v^{a+1}} \text{ for } v\geq\beta.$$

Defining

$$G\left(\alpha\right) := \int_{\beta}^{\infty}\left(1-e^{-\bar{C}v}\right)\phi_{\alpha}\left(v\right)dv$$

we get

$$\text{Normal connection list length } = \bar{N}G\left(\alpha\right).$$

Figure 4 shows the value of $G$ for different $\alpha$. For the parameters found from the Backbone 2 link, $\bar{N} = 322562$, we get normal connection list length $= 811$. This value if slightly larger than the value given by (3).

While it seems plausible that $\phi$ typically has a Pareto distribution, more work is required to verify if the parameters

[3]The distribution found in (4) indicates that the mean connection establishment rate is infinite. However, examining the Figure 3, it can be seen that for the high connection establishment rate the distribution is not well modeled by Pareto distribution and likely has a finite mean. However, this is irrelevant for this analysis as any source with high connection establishment rate will be tracked continuously according to the algorithm.

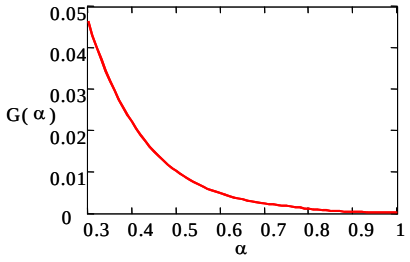


Fig. 4. Fraction of total sources that are simultaneously tracked. Here only flows that established connections are considered.

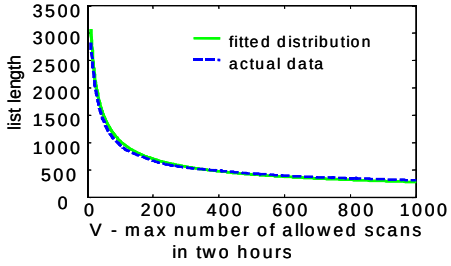


Fig. 5. List length of non-scanning pure SYN flows as a function of maximum number of allowable scans in two hours.

are consistent over other links. For example, Figure 4 shows that $G(\alpha)$ grows quickly as $\alpha$ decreases. On the other hand, the computation is dependent on the *log* of the length of the list. Thus, the computation might not be greatly affected by small variations in the parameters. While it is possible that $\alpha$ and $\beta$ remain fairly constant for different links, $\bar{N}$ depends on the data rate carried by the link. To see how this might be true, note that $\alpha$ and $\beta$ and the distribution of the files accessed define the end-users demand. Thus, assuming that the users' demand remain fairly constant, a variation in the data rate carried over the link can only be accomplished by changing $\bar{N}$. However, more experimentation is required to verify this claim.

The analysis presented (both the direct approach via (3) or model-based approach via (4)) are conservative because they assume that each source establishes connections according to a Poisson process. This assumption produces widely spaced connection establishments. In reality, a source may establish connections in a more bursty fashion. As a result, some connection establishments might be more closely spaced so that they overlap more than predicted by the Poisson model. Since the total time in which a source is tracked is reduced when connection establishments overlap, bursty connection establishments by a source would reduce the length of the list.

### *B. Computational load due to non-scanning pure SYN sources*

While scanning sources will send a moderate or large number of pure SYN flows, normal, non-malicious, end-hosts will also occasionally send pure SYN flows. The source of these pure SYN flows must be tracked to ensure that they are not scanning. In the case of normal TCP connection establishments, it was critical to realize that a small number of sources engaged in establishing a large number of connections. This allows the tracking of a single source to track many open SYN simultaneously. However, when tracking non-scanning sources, there is no similar benefit. Furthermore, while a normal connection establishment requires a source to be tracked for roughly 300ms, the source of a pure SYN flow must be tracked for far longer. Nonetheless, the number of sources that send such pure SYN flows is small enough that the number of such sources that is tracked at any one time is small. In this section the computational load associated with these non-scanning sources of pure SYN flows is analyzed. Specifically, we find the number of such sources that are simultaneously tracked. As in the previous section, we take two approaches. First we apply some simple analysis to estimate the number. Second, we model the distributions of the traffic and estimate the number based on the model. While more work is required, it is hoped that the model parameters derived from the data examined here is similar to the model parameters for other backbone links, modulo the data rate carried by the link.

In this section the assumption that packets with spoofed address have been filtered out is critical. If this is not the case, then a single source can send a large number of SYNs each with different source addresses as is the case in a DoS SYN attack. Such an attack would certainly cause the list of tracked sources to grow very large. However, as mentioned, there has been extensive work on filtering packets with spoofed source addressed. For example, the Cisco's IOS software Release 11.1(17)CC has such capabilities.

To analyze the computational load due to non-scanning pure SYN sources, we assume that the threshold for scanning is of the form (1). Thus, a source is not scanning if the total number of pure SYN flows sent within $T$ seconds is less than $V$. As mentioned, for each pure SYN flow that a source sends, the sources is tracked for a period of $1/(V/T)$. However, if a source is declared to be scanning, then it is continuously tracked. Thus, if $d^{\alpha}(T)$ is the number of pure SYN flows from source $\sigma$ and $d^{\sigma}(T) < V$, then the total time that this source will be tracked $d^{\sigma}(T)/(V/T)$. The fraction of time that this source is being tracked during the $T$ interval is $d^{\sigma}(T)/V$. Thus, the source $\sigma$ adds $d^{\sigma}(T)/V$ to the average number of sources being simultaneously tracked. The average number of sources simultaneously being tracked is

$$\sum_{d^{\sigma}(T) \leq V} d^{\sigma}(T) \frac{1}{V}. \tag{5}$$

Figure 5 shows this value for different values of $V$.

On the other hand, the average rate that a source sends pure SYN flows is $d^{\sigma}(T)/T$. We denote this rate as $r^{\sigma}$. With the threshold of the form (1), sources that send pure SYN flows at an average rate that is less than $V/T$ are not scanning. Let $f(r)$ be the fraction of sources that send pure

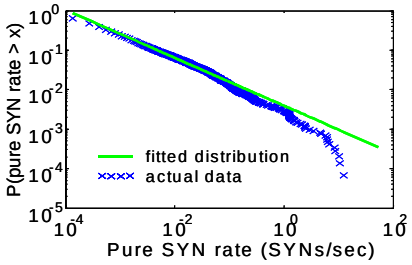


Fig. 6. Complementary distribution of the rate of a source sending pure SYNs.

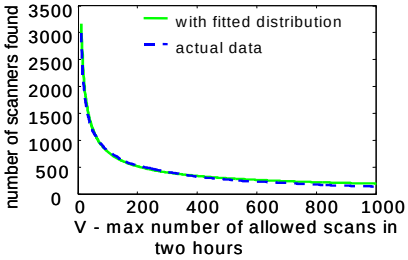


Fig. 7. The number of scanning sources as a function of the maximum number of allowable scans in two hours.

SYN flows at rate $r$. Since each pure SYN flow leads to a source being tracked for $T/V$ seconds, the sources that send pure SYN flows at rate $r$ add $\tilde{N}f(r)rT/V$ to the list length, where $\tilde{N}$ is the total number of sources. Thus, the number of non-scanning sources of pure SYN flows that are simultaneously tracked is

$$\tilde{N}\sum_{r<\frac{V}{T}} f(r)\, r\frac{T}{V},$$

where $\tilde{N}$ is the total number of sources. Figure 6 shows the complementary distribution of $r$. This distribution is well approximated by a Pareto distribution with $\alpha = 0.6$ and $\beta = 10^{-4}$ and is shown by the solid straight line in the Figure 6. Assuming this continuous distribution, the average length of the list of non-scanning sources that are tracked because of pure SYN flows is

$$\tilde{N}\frac{1}{R}\int_{r<R} f(r)\, r dr, \tag{6}$$

where a source is declared to be scanning if it sends pure SYN flows at an average rate of $R$ over the time interval $T$ (i.e., $R = V/T$). Figure 5 shows this value (6) for different values of $R$. We see that using the model with a continuous distribution via (6) yields the same result as the direct approach given by (5).

In the data examined here we found that the number of sources that send pure SYN flows is 14736 in the data studied here, or 4.5% of the total sources that sent any TCP SYN. Further data analysis is required to determine if this percentage is similar to the percentage found on other backbone links. Similarly, further investigation is required to determine if the distribution of the model parameter, $\alpha$ and $\beta$, are similar to those found on other links.

Like the analysis in the previous section, the analysis here is conservative. Here it is assumed that when a source sends pure SYN flows, these flows are widely spaced. However, this is often not the case as a non-scanning source may be attempting to connect to a disconnected host and hence send many SYNs in a short period of time. In this case, these SYNs are tracked together, while the analysis here assumes that each flow is tracked individually.

### C. Computational load due to scanning sources

The objective of this work is to detect and track sources that are scanning, i.e., sources that send more than $H(t)$ pure SYN flows during a time period of length $t$. There is no way to limit the impact of tracking these sources as the objective is to track them. Once a source is declared to be scanning, all packets that come from it should be collected for later investigation. Nonetheless, when a packet arrives, it must be determined if the source of the packet is known to be a scanner, is currently being tracked, or not being tracked. This classification requires searching through the list of addresses and the list must include the sources that are known scanners.

The analysis of computational load of tracking scanning sources is similar to load of non-scanning sources. Specifically, the number of scanning sources is

$$\sum_{d^{\sigma}(T)>V} 1. \tag{7}$$

Employing $f$, the distribution of the rate that sources send pure SYN flows, the number of scanning sources is

$$\tilde{N}\int_{r>R} f(r)\, dr. \tag{8}$$

Figure 7 shows the number of detected scanners as a function of different thresholds $V$ in (1). Again, the direct approach and model-based approach agree.

### D. Total computational load

The total computational load is the sum of the computational load of the three parts, tracking TCP connection establishments, tracking non-scanning sources, and tracking scanning sources. Left-hand plot of Figure 8 shows the total average number of simultaneously tracked sources, both from direct analysis and using traffic models. Right-hand plot of Figure 8 also shows the same number but shown in log-scale. Since searching through a list of length $N$ takes $O(\log(N))$ operations, the right-hand figure is an indication of the computational load due for the detection method. Note the computational load varies quite slowly.

In the case of the link examined, this detection method would require between 10 and 12 memory accesses per TCP packet. Since this link had a packet arrival rate of 83K packets per second (SYN packet arrival rate is under 4,000/sec) this detection method would require an memory

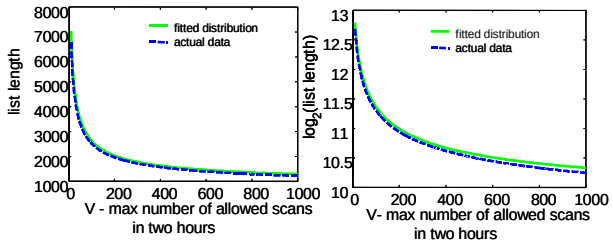


Fig. 8. Total Computational Burden. The left-hand figure shows the computational burden in terms of the length of the list of source that are simultaneously tracked. The right-hand figure is the same as the left-hand figure, except that the log of the length of the list is taken.

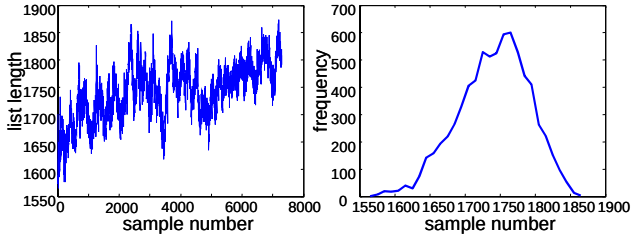


Fig. 9. The left-hand side shows a time series of the number of simultaneously tracked sources of non-pure SYN flows. The right-hand side shows the histogram of this time series. Here the threshold is 50 pure SYN flows in two hours.

access roughly once every 100 ns; this is well with the reach of today's microprocessors.

### *E. Detection test*

The algorithm was tested on the data collect from the backbone link described in Section II. The right-hand plot in Figure 9 shows the number of sources tracked while the left-hand plot shows the histogram of the number of tracked sources. Here the source is declared to be a scanner if it sends more than 50 pure SYN flows in two hours. It can be seen that the variance of the list size appears to be small. The algorithm was further tested for different thresholds. Figure 10 shows the mean list length (left side) and number of scanners (right side) as a function of different thresholds $V$. It can be seen that the actual list length in these figures is smaller than that given by the analysis in the previous sections. In light of the discussion at the end of sections VI-A and VI-B, we expect this to be the case.

## VII. Conclusion

The method investigated here tracks every source that sends an unanswered TCP-SYN. As a result, it is able to detect even stealthy scanners. On the other hand, since most hosts do not send an unanswered SYN very frequently, tracking such sources is computational feasible. The analysis indicates that the computational complexity of the method is not sensitive to variations in the traffic characteristics. Thus, we expect that detecting TCP-SYN scanning at the backbone is feasible. Of course, a key assumption is that packets with spoofed source addresses

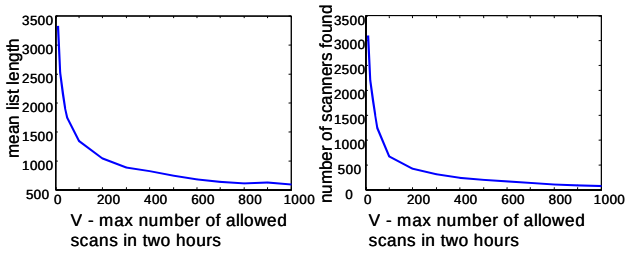


Fig. 10. The left-hand side shows the mean list length versus threshold $V$. The right-hand side shows the number of scanners found versus threshold $V$.

are filtered. Nonetheless, this analysis shows that some stateful filtering at even backbone links is feasible.

This detection scheme does not detect all types of scanning. However, for most other types of scanning, detection can be carried out perhaps by employing a scheme similar to the one presented here. For example, there are other scanning techniques that do not use SYNs. Some of these are easily detectable since they send TCP packets with the flags set in a nonsensical fashion (e.g., both SYN and FIN set). Detecting these types of scanning is the focus of ongoing effort.

## VIII. Acknowledgement

We would like to thank K. Claffy and Colleen Shannon at CAIDA for their valuable comments. We would also like to thank Ken Keys and Dan Anderson at CAIDA for their help and support for data processing and collection.

## References

[1] E. Stafford, "The Internet Worm Incident," *2nd European Software Engineering Conference*, 1989.
[2] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The spread of the sapphire/slammer worm," 2003.
[3] CERT Advisory CA-2001-23, "Continued threat of the "Code Red" worm," 2002.
[4] CERT Advisory CA-2003-20, "W32/Blaster worm," 2003.
[5] O. Arkin, "Network scanning techniques," *http://www.sys-security.com*, 1999.
[6] Nmap, "http://ww.insecure.org/nmap/index.html," 1999.
[7] M. Vivo, E. Carrasco, G. Isern, and G.Vivo, "A review of port scanning techniques," *ACM SIGCOMM Computer Communication Review*, vol. 29, Issue 2, pp. 41–48, 1999.
[8] SHADOW Indications Technical Analysis, Coordinated Attacks and Probes. *http://www.nswc.navy.mil/ISSEC/CID/*, 1999.
[9] V. Yegneswaran, P. Barford, and J. Ullrich, "Internet intrusions: Global characteristics and prevalence," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, Issue 1, pp. 138–147, 2003.
[10] S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *J. of Computer Security*, 2002.
[11] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," *Sigcomm*, 2001.
[12] C. B. Lee, C. Roedel, and E. Silenok, "Detection and characterization of port scan attacks,"
[13] Waikato Applied Network Dynamics group, "The dag project: http://dag.cs.waikato.ac.nz,"
[14] E. M. Systems, "www.endace.com,"
[15] A.Broido, Y.Hyun, K.Claffy, and R.Gao, "Their share: Diversity and disparity in IP traffic," *Passive and Active Measurement*, 2004.
[16] W. R. Stevens, *TCP/IP Illustrated (Vol. 1): The Protocols*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc, 1993.