

Tracking Port Scanners on the IP Backbone

Avinash Sridharan
University of Southern California
Dept. of Electrical Engineering
asridhar@usc.edu

Tao Ye
Sprint
Burlingame, CA
tao.ye@sprint.com

ABSTRACT

Port scanning is the usual precursor to malicious attacks on today's Internet. Although many algorithms have been proposed for different aspects of the scan detection problem, their focused designed space is enterprise gateway level Intrusion Detection. Furthermore, we find few studies that track scanner behaviors over an extended period of time. Operating from a unique vantage point, the IP backbone, we put all the pieces together in designing and implementing a fast and accurate online port scan detection and tracking system. We introduce our flexible architecture, discuss trade-offs and design choices. Specifically, we go in depth to two design choices: the distinct counter data structure and the buffer size tuning. Our choice of a probabilistic counter is proved to have a low average error of 3.5%. The buffer size is derived using Lyapunov drift techniques on an equivalent queuing model. Our initial scanner tracking results show that the scanners' timing behavior follows a 90-10 curve. That is, 90% of scanners are active for a short time, with low scanning rates, while 10% are long term and fast scanners, with a few super-scanners lasting the entire duration of monitoring.

Categories and Subject Descriptors

C.2.3 [Computer Communication Networks]: Network Operations—*Network Monitoring, Public Networks*

General Terms

Measurement, Security, Experimentation

Keywords

Port scanning, Real time port scan detection, IP backbone monitoring

1. INTRODUCTION

Malware of many different varieties continue to spread through today's Internet at an alarming rate and volume. Port scanning is a major channel of such propagation [20]. To detect, understand, and eliminate such traffic is vital to enhancing Internet security. Although the detection has attracted considerable attention in recent

years, to our surprise, we have seen few studies on global scanner behaviors over a long time period. This is particularly true from the large scale backbone perspective.

Prior works have focused on detecting and blocking scanners at enterprise or stub networks. We have seen very few systems for the backbone network in the literature. The main existing efforts in tracking anomalies are honeypots, [3], [19] and [5] i.e. dark addresses, and the Internet Storm Center (SANS)'s [4] collection of enterprise or stub network intrusion detection system (IDS) logs. This detailed study [14] characterizes traffic received by dark addresses, termed 'background radiation', observed from many honeypots. While a very valuable resource, enterprise or stub networks can only observe a small portion of the scanning activity due to its limited IP address space. Honeypots also suffer from such limitation, furthermore, intelligent viruses and worms are known to evade large and well-known honeypots [6].

Our aim is to deploy detection and tracking devices in the upstream transit network that covers a far larger address space, complimenting existing monitors. It is much more difficult to evade transit networks, especially for scanners with a large footprint target. A transit network can also observe a higher scanning rate, hence identifying a scanner faster. Compared with stub networks, backbone networks pose several unique challenges. The traffic is much higher in speed at the core, challenging systems to scale to 10Gbps speed and above. The traffic mix is also far more diverse, requiring more efficient data structures. We address these by implementing fast streaming algorithms without sacrificing accuracy. Among the existing implementations for the backbone BeProf [18], [17] aims to profile general anomalous behavior that deviates from normal traffic behavior. However the parameter tuning of BeProf is less than intuitive and can be hard to master. Gigascope [2] is a general purpose traffic metering system. Anomaly detection systems such as earlybird [15] and Autograph [11] are signature extraction based systems, while we choose to focus on behavior pattern based systems.

An accurate detection algorithm is a critical choice for the system. TRW [10] is a state-of-the-art algorithm with fast detection time and high accuracy. However, TRW relies on knowing if a source has made a failed or a successful connection, and therefore requires stateful protocol analysis of the sources. This cannot be done reliably on the backbone network due to uni-directional link monitoring and routing asymmetry. We choose to implement TAPS [16], a connectionless port scan detection algorithm designed for the high speed backbone network, shown to have similarly high accuracy as TRW.

A primary challenge in realizing an online detection algorithm is designing efficient data structures that can efficiently store and search information on each IP address monitored. Probabilistic

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM'07, August 27–31, 2007, Kyoto, Japan.
Copyright 2007 ACM 978-1-59593-785-8/07/0008 ...\$5.00.

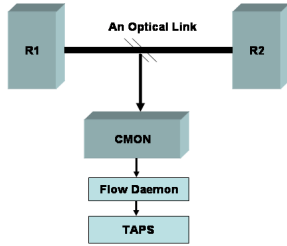


Figure 1: Traffic Monitoring Architecture

sketches are generally the data structure of choice to achieve these goals. Although there have been recent advancements in proposals of new probabilistic sketches, specifically targeting in memory counting of active flows([8] [21] and [7]), we show that using the simpler and more general Flajolet Martin distinct counters (FM counters) [9] is sufficient for the purpose of implementing TAPS on the backbone.

We build and deploy a live system to detect and track scanners in a tier 1 ISP backbone network. We present our design issues and trade-off decisions in this work. With the tracking data, we seek to answer the following: What are the long term behavior of scanners? Can we identify different classes of scanners? Can we identify outbreaks or a bot army?

Our contributions are four folds. First we present a flexible architecture for scanner detection on the backbone. All three logical components are stand alone software implementations and can be deployed on different hardware boxes. Second, we realize the TAPS algorithm with an online implementation. We incorporate a simple data streaming algorithm, probabilistic counting, in our implementation to make TAPS fast and memory efficient while maintaining accuracy. Third, we deploy and evaluate the system performance on live backbone links. Fourth, we present a few intriguing results from tracking scanners over a duration of ~ 3 days. We observe an interesting disparity among scanners. Namely, 90% of scanners scan at relatively low rate but 10% scan at high rates, with a few at very high rates. 90% of scanners are also active for a short time, a few hours, while 10% lasting days. Further, we observe one incident of sudden increase in the number of detected scanners, suggesting a potential coordinated attack in the network. A future more in-depth study is planned to understand the above behaviors.

2. TRAFFIC MONITORING ARCHITECTURE

We present our architecture in Figure 1. Three logical components form our system: **CMON**, **Flow Daemon** and **TAPS**. On our network, traffic tapping of optical links is performed by CMON [13] systems. While CMON has other statistical analytical capabilities, we use it to simply monitor packet traffic and output NetFlow (V5) style flow information. It exports a five tuple flow - source IP, destination IP, source port, destination port, protocol. Endace’s commercial Ninja probe, or any router or appliance that output unsampled NetFlow (V5) with accurate timestamps can achieve the same effect. We choose CMON due to its availability, in house knowledge and cost. CMON is capable of monitoring backbone links up to link speeds of OC-192.

Multiple applications such as TAPS can potentially demand the flow data. Hence, we author **Flow Daemon** to address the main bot-

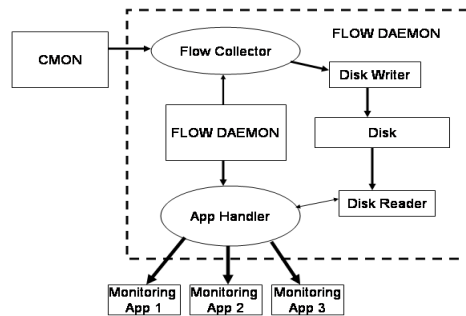


Figure 2: Flow Daemon Architecture.

tle neck of the system: the flow processing queue. This component collects flows and duplicates them to any application over a TCP/IP socket. Flow-daemon allows us to process flows asynchronously by creating a tunable buffer. This enables us to deal with large traffic variation, and ‘spikes’ or micro congestion that happen frequently in the backbone. We describe in Section 2.1 the design and implementation choices made for **Flow Daemon** to highlight the fact that the performance of the **Flow Daemon** directly impacts the performance of **TAPS**.

The final component is the real time implementation of TAPS detection and tracking module, described in Section 2.2.

2.1 Flow Daemon

Flow Daemon, shown in Figure 2, has two main components: the **Flow Collector** that listens for connections from CMON, and the **Application Handler** that listens to connections from interested applications. The **Flow Collector** and the **Application Handler** share a common buffer on the hard disk. The **Flow Collector** uses a component called the **Disk Writer** to write flows to the disk and the **Application Handler** uses the **Disk Reader** to read flows from the disk.

We make a design choice of storing the flows on disk and not in volatile memory, for capacity reasons. The **Disk Writer** implements a simple scheme using circular buffers as the storage mechanism. The **Disk Writer** divides flows that it receives from the **Flow Collector** into time intervals of 1 minute and writes them as single files to the disk. The files function as a circular buffer because **Disk Writer** overwrites the first file when a file number limit is exceeded. We use a centralized locking mechanism with the POSIX **pthread**’s atomic read and write operations to enforce synchronization between the **Disk Writer** and **Disk Reader** on the files of the circular buffer.

This design requires us to answer an important question on parameter tuning: What should be the size of the circular buffer? Or, how many minutes of data do we buffer? We need a finite bound on the buffer size to ensure that the system can perform well even under heavy loads.

To answer the above question, we model the circular buffer as a single queue that can service data only after Q bytes worth of data has been stored. In order to make our analysis tractable let us assume that the queue is slotted and works on discrete time boundaries. Each time slot the queue receives $A(t)$ arrivals and has $U(t)$ backlogs in the queue. The service rate that is available to the queue

is $\tilde{\mu}(t)$. Where $\tilde{\mu}(t)$ is defined as follows:

$$\tilde{\mu}(t) = \begin{cases} \mu(t), & U(t) \geq Q \\ 0, & U(t) < Q \end{cases}$$

That is the queue starts processing packets at a service rate $\mu(t)$ only if it has a backlog $\geq Q$. If we assume that $\mu(t) \leq \mu_{max}$ where $\mu_{max} \leq Q$, then the queueing dynamics can be given by the following equation:

$$U(t+1) = U(t) - \tilde{\mu}(t) + A(t)$$

The service rate of the queue would thus be given by:

$$\tilde{\mu}(t) = \begin{cases} \mu(t), & U(t) \geq Q \\ 0, & U(t) < Q \end{cases}$$

Where $U(t)$ is the queue backlog at time t and $\mu(t)$ is the service rate available to the queue at time t .

By providing bounds for the back log of the queueing system we will be able to bound the buffer size of the flow daemon. A bound on the expected backlog of the queue can be found by calculating the Lyapunov drift [12] and applying the Lyapunov drift theorem [12]. The bound on the expected back log queue will be given by :

$$\bar{U} \leq \frac{\mathbb{E}\{\mu^2\} + \mathbb{E}\{A^2\} + 2Q\bar{\mu}}{2(\bar{\mu} - \lambda)} \quad (1)$$

Where $\lambda = EA(t)$ is the expected arrival rate for the arrival process, $\bar{\mu}$ is the expected service rate of the service process, $\mathbb{E}\{\mu^2\}$ is the mean square of the service process and $\mathbb{E}\{A^2\}$ is the mean square of the arrival process. The proof of the bound in equation 1 can be found in Appendix A.

In order to calculate the required buffer size we need to know the expected arrival rate λ , the service rate $\bar{\mu}$, the mean square of the arrival rate $\mathbb{E}\{A^2\}$ and the mean square of the service rate $\mathbb{E}\{\bar{\mu}^2\}$. To get an estimate of the required buffer size we performed an experiment on an BB-west-1 (Table 2) link that we plan to operate on. In our experiment we over provisioned our flow daemon to store 24 hours worth of data in its cyclic buffer before it starts over writing the data stored in the very first minute of the experiment. We allowed the setup CMON+Flow Daemon+TAPS to run on this link for slightly less than 24 hours. In this run we logged the number of flows that were being processed by the flow collector every second. The service rates from the logs give us a means of calculating λ and $\mathbb{E}\{A^2\}$ for the arrival process at the flow collector. To estimate $\bar{\mu}$ and $\mathbb{E}\{\bar{\mu}^2\}$, we use the fact that the bound on the back log queue will increase as the difference between the arrival rate λ and the service rate $\bar{\mu}$ reduces. Thus we can approximate a loose upper bound, equivalent to over provisioning the system, by assuming $\bar{\mu} = 1.1 \times \lambda$ and $\mathbb{E}\{\bar{\mu}^2\} = (1.1)^2 \times \mathbb{E}\{A^2\}$. Also an approximation to Q can be made by setting $Q = \lambda \times 60$. Using the above approximations the bound on the buffer size can be expressed in terms of λ and $\mathbb{E}\{A^2\}$ alone:

$$\bar{U} \leq \frac{2.2\mathbb{E}\{A^2\} + 132\lambda^2}{0.2\lambda} \quad (2)$$

For our specific experiment, by taking averages over 2 hours of data we get $\lambda = 11697$ flows per second, i.e. ~ 12000 flows per second, and $\mathbb{E}\{A^2\} = 144240416$, giving $\bar{\mu} = 12866.7$, $\mathbb{E}\{\bar{\mu}^2\} = 174530902$ and $Q = 701820$ flows per minute. Plugging the above values in equation 2 and dividing it by Q , the number of flows stored per minute, we get the number of time bins we need to store data is equal to $10.91 \sim 11$ time bins. In order to cater for bursts of traffic (which might occur in case of worm attacks) we keep ~ 5

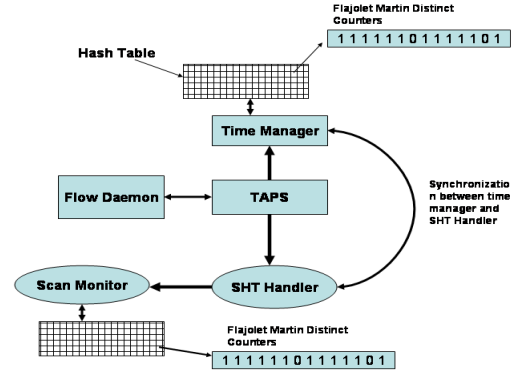


Figure 3: Taps Architecture

times the bound we have on the buffer size. Thus, instead of storing 11 minutes of data, we store 60 minutes worth of data.

2.2 TAPS: Time based Access Pattern Sequential hypothesis testing

The TAPS scanner detection and tracking module is the core of our system. We summarize the TAPS [16] algorithm and focus on the online implementation in this section. TAPS, a time-based access pattern sequential hypothesis testing algorithm, performs sequential hypothesis tests on a per source basis on flows to determine whether a source is malicious or benign.

2.2.1 Algorithm and Implementation

The intuition behind this algorithm is that a scanning host's access pattern demonstrates a high value for the ratio of $\frac{No.of\ destination\ IP}{No.of\ ports}$, in a given period of time. We denote this period of time as a time bin. A scanner is thus expected to go to a large number of destination IP's over a small range of ports, exhibiting this behavior over multiple time bins. The distribution of this ratio shows a clear demarcation in the statistical behavior of the scanner set and benign hosts. This ratio is then used to perform a test for the hypothesis of whether a host is *BENIGN* or a *SCANNER*, across multiple time bins. A detailed description of TAPS is presented in [16] along with analytical bounds on its performance and an empirical evaluation of the algorithm on back bone traces. TAPS depends solely on counting the destination IPs and ports of a source, without relying on connection state information. Therefore, it can be used for detecting both TCP and UDP scans. Moreover, since backbone links are all uni-directional in nature, connection state inference is difficult. TAPS's connectionless design make it a suitable algorithm for backbone scan detection. Finally, it is shown to be fast, reaching a decision within 6 time bins on average, while yielding low false positives and high detection rates. The parameters that govern the TAPS algorithm are :

k : The threshold for $\frac{IP}{Port}$ ratio.

H_1 : The hypothesis that a source is a SCANNER.

H_0 : The hypothesis that a source is BENIGN.

θ_1 : Probability that $\frac{IP}{Port} < k$, given that the source is a SCANNER.

θ_2 : Probability that $\frac{IP}{Port} < k$, given that the source is BENIGN.

m :The number of hash functions used in the FM counters.

For a more detailed description of these parameters we refer the reader to [16].

TAPS performs the sequential hypothesis test on every source at the end of every time bin (It is essential to note that these time bins

are different than the ones used in the circular buffer for **Flow Daemon**. The size of the time bins and the $\frac{IP}{port}$ ratio threshold k are user configurable parameters that are inputs to TAPS. The TAPS implementation consists of three components, shown in Figure 3. The first component, called the **Time Manager**, keeps track of the information required by TAPS to perform the analysis. This includes the start and end times of flows, the sources seen in a given time bin, and the per source data namely the number of distinct IP and port that a source attempted to connect to. The second component, the **SHT Handler** (Sequential Hypothesis Test Handler), uses the information collected by the **Time Manager** and performs the sequential hypothesis test at the end of each time bin to tag sources as scanners or benign hosts. The third component **Scan Monitor** provides the tracking functionality for every source that was tagged by the **SHT Handler** as a *SCANNER*. For each time bin it maintains the following statistics for each scanner IP: (i). The number of distinct IP and ports accessed by the scanner, (ii). The number of time bins that the scanner has been active, (iii). The first time bin at which the source was tagged as a scanner.

2.2.2 Data Structure: Flajolet Martin Distinct Counters

Although the theoretical complexity of TAPS is low, the associated data structure plays a significant role in the performance of the algorithm. At the end of every time bin, for every source IP under test TAPS requires a count of the number of distinct IP and distinct ports accessed. A naive approach would be to maintain a heap of all the addresses seen so far. However, the time to update a heap is $O(n \log n)$ and the memory required is $O(n)$, where n can be quite large. Instead, maintaining probabilistic sketches [8] [21] [9] with compact data structures rather than exact counts can solve this problem. The key is to minimize the errors and its impact on false positives and negatives of scan detection.

A classic solution to the counting problem is the Flajolet Martin distinct counters [9] (FM counters). FM counters maintain a probabilistic sketch of the incoming data and give an estimate of the number of unique elements present in the data set. The estimate could be made arbitrarily close to the actual count by increasing the number of hash functions used in the sketch. Estan et al [8] subsequently proposed more sophisticated multiresolution bitmap and trigger bitmap algorithms [8] that aim at a very small memory implementation (SRAM). Since our system implementation is entirely in software on common PC hardware, we choose to use the older but simpler FM counters for this design. Moreover, our empirical analysis shows that although using a small number of hash functions can decrease the accuracy of the raw count of IP or port, its effect on the accuracy of the ratio of $\frac{IP}{Port}$ is actually quite low. We proceed to describe the FM counters and our evaluation for determining the possible number of hash functions required for our implementation.

FM counters consist of m hash functions and m 32 bit vectors (also called the bitmap). The values generated by each hash function are used to set the bits in the corresponding vector. A bit k in the i^{th} vector is set for an element x if $min_bit(hash_i(x)) = k$. Here min_bit is a function that gives the least significant bit of $hash_i(x)$ that is 1. The estimate of the number of distinct elements in a multi-set M is equal to $\frac{2^R}{\phi}$, where $\phi = 0.77351$. R is the position of the leftmost zero in the bitmap generated after hashing all elements in the multi-set. The only property the hash functions need to exhibit is that they can map any element x from the multi-set M to a value between $0 \dots 2^L$, uniformly. Since the multi-set M in question is the 32 bit Internet address space, it is relatively straight forward to design hash functions with the above

Bin Size	m	mean error(%)
1sec	256	0.03%
1sec	32	5%
1sec	8	15%
1min	256	0.009%
1min	32	5%
1min	8	15%

Table 1: Performance of FM counters in terms of % mean error for absolute values of distinct IP and distinct Port counters.

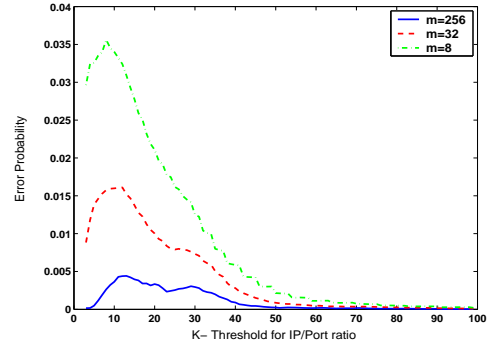


Figure 4: Performance of FM counters with different number of hash functions in terms of the probability of error incorrectly estimating the comparison of $\frac{IP}{Port}$ ratio with different values of k .

property.

2.2.3 Evaluating the Flajolet Martin Counters

FM counters are remarkably accurate when a sufficiently large number of hash functions are used. For example, with $m = 256$ hash functions we could get estimates of the count with close to 5% standard error [9]. However, TAPS is not interested in accurate counts of distinct IP and ports; rather we are interested in a reliable estimate of whether $\frac{IP}{Port} < \text{or } \geq$ to a threshold k . The exact value of the $\frac{IP}{Port}$ ratio is therefore not necessary. We show that in measuring the inequality, even small values of m give us accurate estimates. Smaller m in turn improve on the time and space complexity of our implementation during online detection. However, accurate absolute counts are important in the tracking module for tagged scanners, there we use a large m value.

Our empirical evaluation is set up as the follows; We implement both data structures in TAPS, a heap and an FM counter, to verify FM's accuracy. We run TAPS offline on a 1 hour trace collected on the Sprint IP backbone. At every time bin TAPS maintained the number of distinct IP and distinct ports for every source IP seen, in both data structures. We then compare the actual count in the heap and an estimated count from the FM counters. The above evaluation is done on the same trace for different values of $m = 8$, $m = 32$ and $m = 256$.

Table 1 presents the mean percentage error observed between the actual count and the estimated count. The mean error increases with the decrease in the number of hash functions used. It is relatively high for the $m = 8$ case ($\sim 15\%$). Thus if we required an exact count of the IP and ports accessed we would need to use $m = 256$ or $m = 32$ hash functions.

Figure 4 presents an interesting observation. It shows the probability of error in estimating the correct decision for the inequality $\frac{IP}{Port} < k$ or $\frac{IP}{Port} \geq k$. For $m = 256$ the probability of error

Link name	Description	Utilization	Utilization
BB-west-1	peering, incoming	72kpps	320Mbps
BB-west-2	peering, outgoing	115kpps	560Mbps

Table 2: Description of backbone links that our initial deployment monitors

is quite small across all values of k . The probability of error increases for small values of k as the number of hash functions m are reduced. However even for $m = 8$ the probability of error is 0.03 for small values of k ($k < 10$). For large values of k the probability of error tends to zero independent of m . The figure 4 verifies our intuition that even by using FM counters with small m ($m = 8$) the accuracy will not significantly degrade when compared to the performance of TAPS with $m = 256$. We will quantify this argument in section 3.2.

3. SYSTEM EVALUATION

In this section we detail the system deployment and performance evaluation.

3.1 Deployment and Performance

We deploy the entire system to monitor two backbone OC-48 links, described in 2. These links represent a well utilized peering point of Sprint’s backbone network with another tier 1 ISP, in the west coast of United States.

We use a CMON [13] passive monitoring node on a PC equipped with DAG 4.3 optical packet capture card [1] for each link. CMON’s flow classifier sets an active timeout period of 1 minute, and exports flows through sockets to the Detector PC. On the Detector PC Flow-daemon and TAPS run as separate processes. These PCs have a typical 2U rack server configuration, with dual processors of 2.5Ghz, 4GB of memory and 350GByte of hard disk space. They run Linux with a 2.6 kernel with a Fedora Core 5 distribution.

A common mode of failure is packet or flow dropping, indicating the system cannot process as fast as traffic arrives. Since the flow daemon buffer size was set according to the bounds found in section 2.1, for our set of experiments we were able to stabilize the system. By observing the logs of CMON, flow daemon and TAPS, we validated this design choice by ensuring that no flows were dropped during the ~ 3 day run of our experiments.

As any scanner classifier, TAPS can produce false positives or negatives. The question is how severe is this inaccuracy and if it is acceptable by operators. In [16] we quantify the low false positive and high detection rate of TAPS, given a data set with ground truth. Here we point out TAPS’s known weakness in distinguishing a NAT firewall or proxy from a true scanner, as these false positives are notoriously difficult to avoid and as yet we do not have a good solution. However, complimentary deep packet inspection (DPI) techniques can be applied to our scanner subset to further verify if an IP is truly malicious. Since DPI is known to be much more resource intensive, TAPS can feed a DPI process only highly suspected flows rather than all flows. In the rest of the paper, we simply focus on the scanners as defined by TAPS and do not revisit this issue.

3.2 Number of hash functions and accuracy

In order to evaluate the affect on accuracy we ran the system with three different versions of TAPS, with different number of hash functions ($m = 8, m = 32, m = 256$). We denote these as TAPS_256, TAPS_32 and TAPS_8. All three versions were started simultaneously, receiving the same flows from **Flow Daemon** for

m	missed	mean	variance	standard deviation
32	18 \ 93	2.3226	2.224	1.49
8	15 \ 93	3.3187	6.0638	2.46

Table 3: The mean, variance and standard deviation of the $\frac{IP}{Port}$ ratio for the scanners missed by TAPS_8 and TAPS_32 and caught by TAPS_256.

12 hours. The parameter setting for all three versions of the TAPS was $k = 3, \theta_1 = 0.2, \theta_0 = 0.8, \eta_1 = 99$ and $\eta_0 = 0.01$. As shown in [16] the above parameters effectively define the average scanning rate that TAPS can detect. Average scanning rate is defined as:

$$scanning\ rate = \frac{E[\frac{IP}{Port}|H_1]}{E[N|H_1]}$$

Where $E[IP|H_1]$ is the expected number of IP’s accessed by a source given that it is a scanner and $E[N|H_1]$ is the expected number of time bins required by TAPS to come to a decision that the source in question was a scanner. Thus the scanning rate quantifies the average $\frac{IP}{Port}$ ratio exhibited by the scanner. A lower bound on the scanning rate in terms of the above parameters is presented in [16] as follows;

$$scanning\ rate = \frac{1}{E[N|H_1]} \times \left(\frac{k + \frac{\theta_1}{\theta_0}}{1 - \frac{\theta_1}{\theta_0}} \right) \left(\frac{\ln(\eta_1)}{\ln(\frac{1-\theta_1}{1-\theta_0})} - 1 \right) \quad (3)$$

Thus with a threshold of $k = 3$ the average scanning rate that can be detected by TAPS would be a $scanning\ rate = 1.77 \sim 2$. Thus based on these parameter settings, on average, if a source visits two IP’s on a single port within a given time bin it would be tagged as a SCANNER.

In the 12 hour duration TAPS_256 caught the maximum number of scanners (93), followed by TAPS_8 (87), followed by TAPS_32 (80), all within 20% of each other. From our empirical evaluation of the FM counters in section 2.2.3 we can assume that a hash function size of $m = 256$ is accurate. Hence, we assume the set of scanners generated by TAPS_256 to be the ground truth for comparison with TAPS_32 and TAPS_8. For our comparison we are interested in understanding the scanners that were caught by TAPS_256 and missed by others. TAPS_32 missed 20 scanners, and TAPS_8 missed 17. On closer observation, we could see that two of the missed scanners lasted for a duration of less then 6 seconds. Note the logs for each scanner are generated once they have been tagged by the **SHT Handler**. We have therefore neglected these two scanners while analyzing the behavior of scanners missed by TAPS_32 and TAPS_8.

Table 3 shows that the mean of the $\frac{IP}{Port}$ of the scanners missed by TAPS_8 and TAPS_32 are close to the threshold $k = 3$, implying that these are relatively low rate scanners. Further, the mean, variance and the standard deviation of the scanners missed by TAPS_32 are smaller than those missed by TAPS_8. This is an expected behavior and can be inferred from figure 4. The table implies that the scanners missed by using a smaller hash function value ($m = 8/m = 32$) are low rate scanners. Therefore the choice of using a smaller value for m should depend on the necessity of detecting low rate scanners. From the perspective of the backbone, our belief is that detecting high rate scanners is more crucial in stopping worm spreads and the task of accurate detection of low rate scanners can be pushed towards the edges. This motivates the argument for using hash functions of small values $m = 8$ at the backbone.

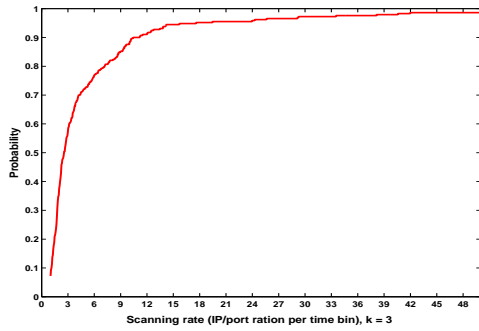


Figure 5: CDF of the scanning rate observed on an OC-48 link BB-WEST-2

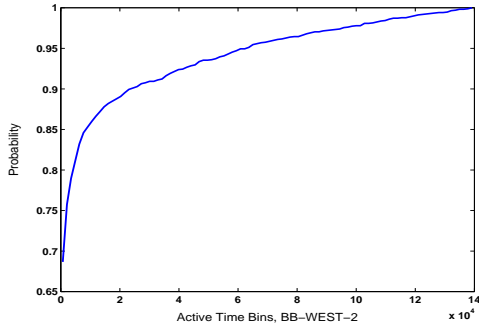


Figure 6: CDF of the number of active time bins of the scanners for the BB-WEST-2 link.

4. SCANNER TRACKING RESULTS

We discuss our initial results from the **Scan Monitor**. Once a source is tagged as a scanner, the tracking component records the $\frac{IP}{port}$ ratio, the number of distinct IP, the number of distinct port every time bin that the scanner was observed. Since we require an accurate estimate of the distinct ports and IP's accessed by the scanner we use the FM counters in the **Scan Monitor** with m set to 256. The two links, BB-West-1 and BB-West-2, yields similar results. However due to space concerns we only present data from BB-West-2 here.

The first question we ask ourselves is: What is a typical scanning rate on the backbone? Interestingly, we observe that scanners scan at very different rates. From one hour of data, in Fig 5, we can see that 80% of the identified scanners scan at a relatively low 9 IP/port per second, while 5% scan at a much high rate of 48 IP/port per second. There are also close to 2% scanners that scan at over 500 IP/port per second. This is not shown in Fig 5 due to the limited range of x-axis on the graph.

Second, we are interested in the length of the active time of the scanners. Fig 6 shows the distribution of active time on the BB-west-2 link. 90% of the identified scanners are active for less than 5.5 hours, while a few super-scanners are active for ~ 3 days that we collected data for. Many factors affect the active time we observe. It is possible that most scanners move to other areas of the Internet after a period of scanning in Sprint's network. Yet the same distribution exists for the entire footprint covered by a scanner. We also observe that 87% of the identified scanners only cover less than 100K IP addresses, 5% cover between 100K to 200K IPs, while a few super-scanners cover 3.5 million IP addresses in three days.

It is interesting to note that most scanners are transient, relatively low rate and cover a small number of IP addresses, while a few

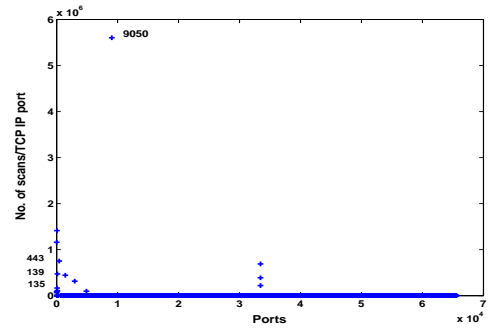


Figure 7: Port profile, Active Time Bin $\geq 5.5hrs$ for BB-WEST-2 link.

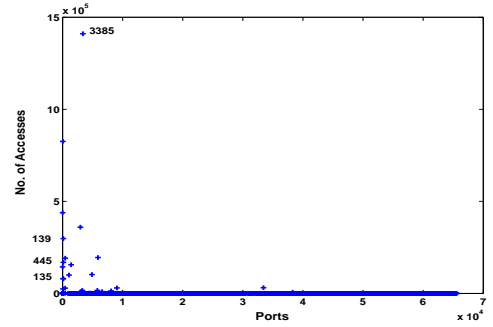


Figure 8: Port profile, Active Time Bin $< 5.5hrs$ for BB-WEST-2 link.

super-scanners scan for long term, at a high rate and cover a large footprint area. We therefore proceed to divide the scanners into two categories: persistent scanners and transient scanners, and show other differences of their behaviors. We set the division point at 5.55 hour of active time, i.e., the 90-10 divide point. Fig 7 shows the ports scanned by the scanner active for more than 5.55 hours and Fig 8 for scanners active for less than 5.55 hours. Firstly, some ports, such as 135 and 139, are common in both sets. Although port number alone cannot determine the worm, we suspect Blaster is behind these scans. The most scanned port in the two sets tells a different story. Port 9050 in long active set and port 3385 in shorter active set are the most scanned ports, respectively. This suggests that the most aggressive malware in the two categories are different.

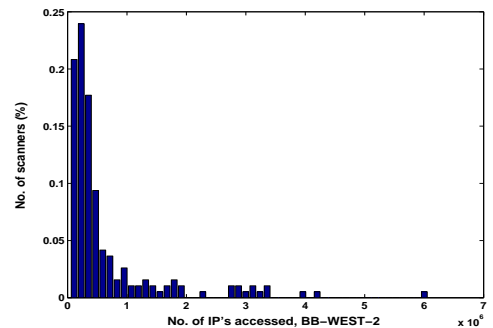


Figure 9: PDF- Distinct destination IP Accessed, Active Time Bin $\geq 5.5hrs$ for BB-WEST-2 link.

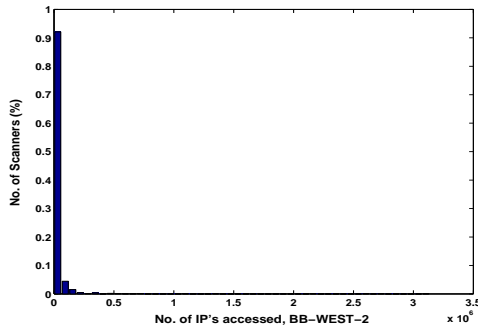


Figure 10: PDF - Distinct destination IP Accessed, Active Time Bin $< 5.5hrs$ for BB-WEST-2 link.

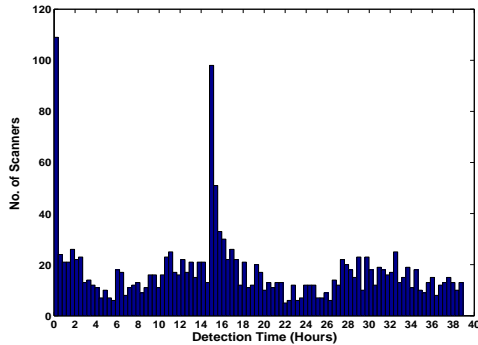


Figure 11: The number of scanners that were detected every hour since the beginning of the trace. At 15 hours from the start of the trace there seem to be surge in the worm scanning activity.

In figures 9 and 10 we present the histogram of the No. of IP's accessed for two sets of scanners. While 92% of the identified short lived scanners accessed fewer than 10000 IP's, there are 3 high rate scanners that accessed more then 100000 IP's. In the long lived scanner set, there also exists disparity in the scanning rate, with a few scanners access above 4000000 IP's while most access around 1000000 IP's.

Finally, we identify sudden outbreaks of scanners. In figure 11 we plot the number of scanners detected every hour for the duration of the trace. Interestingly, there are two spikes at the beginning of the trace and at 15 hours. The spike at the beginning of the trace can be explained by the presence of long lived scanners that were detected at the start of the experiment. The spike at hour 15 shows a sudden surge in scanners entering this link which could be correlated with the rise of a BOT army. The figure thus demonstrates our system's ability to detect malicious events such as the instantiation of large scale worm outbreaks.

5. DISCUSSION

We present the implementation and evaluation of an online port scan detection and tracking system at the IP backbone. One of the primary challenges of implementing the online port scan detection algorithm, TAPS, was to choose efficient data structures to ensure speed and efficiency. Our choice of the Flajolet Martin probabilistic counters [9] presents an efficient solution to this challenge. Our choice of the Flajolet Martin counters over the more advanced bit map algorithms [8], proposed for network monitoring off late, was further strengthened by our experimental evaluation, which showed

that the fraction, $\frac{IP}{Port}$, are more robust to the inaccuracies in the Flajolet Martin counters as compared to the individual counts of destination IP's and ports.

Another facet to designing the online system was to come up with bounds on buffer sizes for the **Flow Daemon**. We accomplished this task by applying queueing theoretic analysis to present bounds on the buffer size which ensured that the system remains stable. Our evaluation suggests that the online implementation of TAPS performs within the theoretical limits of the parameter settings of TAPS.

A possible weakness of TAPS in generating false positives for legal applications that might fall under the purview of our definition of scanners needs to be highlighted. Specifically applications such as NAT, firewall or proxy which might end up emulating the behavior of a scanner. We could curtail these false positives to some extent by increasing the threshold that TAPS detects. However our belief is that, to segregate these false positives completely we would require to go beyond pure statistical analysis and rely on deep packet inspection techniques to achieve fool proof results. This also highlights the main applicability of a tool like TAPS. Although DPI techniques are more reliable, their resource intensive nature makes them counter productive in high speed, high volume environments. It therefore seems logical to make these two techniques to work in conjunction with each other. TAPS, being a fast detection algorithm, can generate a set of IP's that have a high probability of being malicious. Signature based tools can then be applied to data being generated by these sources to get better estimates on the nature of their activity. Also both these tools need not be deployed on the backbone itself. Since TAPS is designed to work on the back bone, one configuration that seems viable is TAPS generating a set of IP's that seem malicious and exporting this data to the edgess where signature based tools could perform a more thorough analysis on flows emanating from these sources.

Our scanner profiling results presents a different facet to the applicability of TAPS. That of monitoring and classifying the sources that exhibit the scanning behavior. We're in the process of collecting long term traces of scanners in order to profile their behavior for future studies. With statistical classification of the behavior of these scanners we hope to build a strong theoretical foundation that would help to improve existing detection algorithms and at the same time help us better understand the spread of these malicious entities.

6. REFERENCES

- [1] Endace. <http://www.endace.com>.
- [2] Gigascope. <http://www.research.att.com>.
- [3] Honeynets. <http://www.honeynets.org>.
- [4] Internet storm center. <http://www.sans.org>.
- [5] Leurrecom honeypot project.
- [6] Mark Allman, Paul Barford, Balachander Krishnamurthy, and Jia Wang. Tracking the role of adversaries in measuring unwanted traffic. San Jose, CA, USA, 2006.
- [7] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. 2003.
- [8] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *USENIX/ACM Internet Measurement Conference*, 2003.
- [9] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. volume 31, pages 182–209, 1985.
- [10] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast portscan detection using sequential

hypothesis testing. In *IEEE Symposium on Security and Privacy*, May 2004.

- [11] Hyang-Ah Kim and Brad Karp. Autograph: Tward automated, distributed worm signature detection. In *13th USENIX Security Symposium*, 2004.
- [12] L. Tassiulas L. Georgiadis, M.J. Neely. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1(1), 2006.
- [13] Preetham Mysore. Cmon: "always-on" monitoring platform for high-speed links. Master Thesis, Rutgers University.
- [14] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *USENIX/ACM Internet Measurement Conference*, 2004.
- [15] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, 2004.
- [16] Avinash Sridharan, Tao Ye, and Supratik Bhattacharria. Connectionless port scan detection on the backbone. In *Malware workshop, held in conjunction with IPCCC*, Pheonix, AZ, April 2006.
- [17] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharria. Profiling internet backbone traffic: Behavior models and applications. In *ACM Sigcomm 2005*, Philadelphia, PA, August 2005.
- [18] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharria. Reducing unwanted traffic in a backbone network. In *USENIX Workshop on Steps to Reduce Unwanted Traffic in the Internet (SRUTI)*, Boston, MA, July 2005.
- [19] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In *the Symposium on Recent Advances in Intrusion Detection*, 2004.
- [20] Vinod Yegneswaran, Paul Barford, and Johannes Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proceedings of ACM SIGMETRICS*, 2003.
- [21] Qi Zhao, Abhishek Kumar, and Jun Xu. Joint streaming and sampling techniques for accurate identification of super sources/destination. In *USENIX/ACM Internet Measurement Conference*, 2005.

theorem [12]:

$$\bar{U} \leq \frac{\mathbb{E}\{\mu^2\} + \mathbb{E}\{A^2\} + 2Q\bar{\mu}}{2(\bar{\mu} - \lambda)}$$

APPENDIX

A. BOUNDING THE EXPECTED QUEUE BACK-LOG

The queueing dynamics are given by:

$$U(t+1) = U(t) - \tilde{\mu}(t) + A(t)$$

Using the above queueing dynamics we can get the inequality:

$$\begin{aligned} U(t+1)^2 &\leq U(t)^2 + \tilde{\mu}(t)^2 + A(t)^2 - 2U(t)[\tilde{\mu}(t) - A(t)] \\ &\leq U(t)^2 + \mu(t)^2 + A(t)^2 - 2U(t)[\tilde{\mu}(t) - A(t)] \\ &= U(t)^2 + \mu(t)^2 + A(t)^2 - 2U(t)[\mu(t) - A(t)] + 2U(t)[\mu(t) - \tilde{\mu}(t)] \\ &\leq U(t)^2 + \mu(t)^2 + A(t)^2 - 2U(t)[\mu(t) - A(t)] + 2Q\mu(t) \end{aligned}$$

The final inequality follows since $\tilde{\mu}(t) \leq \mu(t)$ for all t , thus $2U(t)[\mu(t) - \tilde{\mu}(t)] \leq 2Q\mu(t)$ for all t . Taking conditional expectations of the above and assuming the random variables to be i.i.d the conditional Lyapunov drift is given by:

$$\delta(U(t)) \leq \mathbb{E}\mu^2 + \mathbb{E}A^2 - 2U(t)[\bar{\mu} - \lambda] + 2Q\bar{\mu}$$

Under rate stable conditions ($\lambda < \bar{\mu}$), applying the Lyapunov drift