Boosting the Scalability of Botnet Detection Using Adaptive Traffic Sampling

Junjie Zhang[†], Xiapu Luo[*], Roberto Perdisci[‡], Guofei Gu[II], Wenke Lee[†] and Nick Feamster[†]
[†]Georgia Institute of Technology, [‡]University of Georgia
[*]Hong Kong Polytechnic University, [II]Texas A&M University
{jjzhang,wenke, feamster}@cc.gatech.edu, perdisci@cs.uga.edu
csxluo@comp.polyu.edu.hk, guofei@cse.tamu.edu

ABSTRACT

Botnets pose a serious threat to the health of the Internet. Most current network-based botnet detection systems require deep packet inspection (DPI) to detect bots. Because DPI is a computational costly process, such detection systems cannot handle large volumes of traffic typical of large enterprise and ISP networks. In this paper we propose a system that aims to efficiently and effectively identify a small number of suspicious hosts that are likely bots. Their traffic can then be forwarded to DPI-based botnet detection systems for fine-grained inspection and accurate botnet detection. By using a novel adaptive packet sampling algorithm and a scalable spatial-temporal flow correlation approach, our system is able to substantially reduce the volume of network traffic that goes through DPI, thereby boosting the scalability of existing botnet detection systems. We implemented a proof-of-concept version of our system, and evaluated it using real-world legitimate and botnet-related network traces. Our experimental results are very promising and suggest that our approach can enable the deployment of botnet-detection systems in large, high-speed networks.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Network**]: Security and Protection

General Terms

Security, Algorithms

Keywords

Botnet, Adaptive Sampling, Intrusion Detection, Network Security

1. INTRODUCTION

Botnets are one of the most serious threats to Internet security. A botnet is a collection of compromised hosts (a.k.a., *bots*) that are remotely controlled by an attacker (a.k.a., *botmaster*). Botnets can be instructed to commit various malicious activities, such as launching distributed denial-of-service (DDoS) attacks, sending spam, performing click fraud, or stealing private information. To effectively control a botnet, the botmaster establishes a *command and control* (C&C) channel with the bots, through which the malicious activities can be coordinated.

A number of approaches for network-based botnet detection have been recently proposed [4, 8, 6, 5, 12, 14, 23, 17]. Almost all of these systems apply fine-grained analysis (e.g., *deep packet inspection* (DPI)) in order to detect bot-compromised machines. For example, BotHunter [6] uses a payload-based anomaly detector and a signature-based detection engine. BotSniffer [5] and Rishi [8] need to parse the content of IRC communications. TAMD [23] inspects packet payloads to compute content similarity scores. BotMiner [7] requires DPI to perform *activity-plane* (A-Plane) monitoring, such as binary downloading and remote exploit detection. Although BotMiner's *communication-plane* (C-Plane) analysis does not require DPI, it suffers from scalability issues that prevents its deployment in high-speed networks (Section 6.3). While these systems have shown promising results, because DPI is computationally expensive, they cannot be directly deployed in high-speed networks without special (usually very expensive) hardware support. Furthermore, even when special hardware support is available, most of the proposed techniques may still not be able to keep up with the traffic, due to the relatively high computational cost of their traffic analysis algorithms. Load-balancing (i.e., distributing traffic and computation to multiple processing units) may represent a possible solution. However, a deployment of these systems in load-balancing requires special design and significant changes to the existing detection algorithms.

In this paper, we propose a new packet sampling and scalable spatial-temporal flow correlation approach that aims to *efficiently* and *effectively* identify a small number of suspicious hosts that are likely bots. Their traffic can be forwarded to fine-grained botnet detectors for further analysis. This allows us to significantly reduce the amount of traffic on which fine-grained analysis such as DPI is applied. Thus, we boost the scalability of botnet detection for high-speed and high-volume networks.

Network flow analysis typically requires far fewer resources than DPI. However, collecting *precise* network flow information in high-speed networks is challenging, because we may not be able to afford to process every packet in the network.

In order to solve this problem, packet sampling techniques are commonly employed to reduce the number of packets to be processed. For example, *uniform sampling* and its variant *periodic sampling* are among the most popular packet sampling techniques, and they allow a network operator to reconstruct approximate network flow information. However, their limitation is that they are able to reconstruct relatively precise information about large flows (i.e., flows that carry a high number of packets), such as media streaming flows, but may poorly approximate or miss outright information about small and medium flows. In order to address this issue, some new sampling algorithms have been recently proposed. For example, FlexSample [2] is a programmable framework where a network operator can set conditions to increase the sampling rates packets from specific traffic subpopulations (e.g., packets in small and medium flows). Unfortunately, because different botnet implementations may introduce strong diversity in the properties (e.g., flow size) of their C&C communication flows, it is challenging to set conditions that allow FlexSample to sample packets targetted for a wide range of botnet C&Cs. For example, flows of HTTP-based C&Cs are usually small (i.e., short lived) while those related to IRC-based C&Cs are intrinsically larger. In order to address this problem, we introduce a new *adaptive* sampling technique. Our sampling technique is *botnet-aware* since it is driven by intrinsic characteristics of botnets such as *group similarity*, where the group similarity reflects the fact that bots belonging to the same botnet share similar C&C communication patterns. We also propose a new scalable spatial-temporal correlation approach to identify hosts that share *persistently similar* communications. That is, we aim to identify hosts in a network that persistently share similar communication patterns for a relatively long (not necessarily continuous) period of time. Our spatial-temporal flow correlation analysis is motivated by the following observation. Because of their (illegal) economy-driven nature, botnets are used by the botmasters for as long as possible to maximize profits (e.g., several months, or until the botnet is dismantled by law enforcement), so their C&C communications will be active for a relatively long period of time.

This paper makes the following contributions:

1. We propose a network traffic analysis approach for botnet detection in high-speed and high-volume networks. The objective of our analysis is to efficiently and effectively narrow down suspicious hosts that are likely bots. The network traffic generated by these suspicious hosts can then be forwarded to fine-grained botnet detectors for further analysis.

2. We introduce an *adaptive* sampling technique based on *group similarity*, an intrinsic characteristic of botnets, to sample packets that are likely related to C&C communications with high probability.

3. We propose a new scalable spatial-temporal correlation analysis to identify hosts in a network that share *persistently similar* communication patterns, which is one of the main characteristics of botnets.

4. We implemented a proof-of-concept version of our system, and evaluated it using *real-world* legitimate and botnet-related network traces. Our experimental results show that the proposed approach is scalable and can effectively detect bots with few false positives, which can be further reduced by fine-grained botnet detection systems.

2. RELATED WORK

Researchers have proposed many approaches to detect botnets. Some of the approaches [14, 17, 12, 8, 4] are designed for detecting botnets with IRC-based C&Cs. Recently, researchers proposed an approach to differentiate P2P bots from P2P file sharing applications [18]. These approaches detect botnets with either IRC- or P2P-based C&Cs, while our system can detect both. Some other detection approaches are driven by specific attack information (i.e., spam). Ramachandran et al. [15] used DNSBL to identify bots for spamming, while Zhao et al. used Hotmail logs in BotGraph [22]. Hu et al. [20] proposed RB-Seeker to detect redirection botnets based on spam and network flow information. Compared to these approaches, our system mainly use packet header and network flow information, which indicates a wider deployment. Some detection algorithms uses correlation approaches. BotHunter [6] associates IDS events to a pre-defined bot infection dialog model for detection. BotSniffer [5] leverages the homogeneity of messages and activities to identify botnet C&Cs. Yen et al. [23] proposed TAMD to detect bots by aggregating traffic which shares the same external destination, similar payloads and OS platforms. BotMiner [7] is a protocol- and structure-independent botnet detection system using clustering techniques. These systems depend on DPI-based components, which limit their usage in high-speed networks. In our system, we design botnet-aware packet sampling algorithm and scalable spatial-temporal flow correlation approach for efficient and effective botnet detection, which aims at the deployment in high-speed networks.

Various sampling algorithms have been proposed to reduce the amount of data the network devices have to process in high speed networks and infer the traffic statistics based on the sampled packets. Most of them focus on sampling large flows and improving their estimation accuracy [21]. Recently researchers proposed approaches to focus on sampling packets in small flows. Kumar et al. [1] and Hu et al. [9] proposed algorithms to sample packets in small flows. However, their overall sampling rate depends on the Zipfian nature [19] of Internet and thus they cannot achieve a pre-defined target sampling rate. Ramachandran et al. [2] designed FlexSample, which can sample packets based on pre-defined conditions. FlexSample can be configured to capture packets in small/medium flows while keeping a target sampling rate. However, characteristics of network flows for botnet C&Cs exhibit great diversity among different botnets and thus it is very challenging to propose good conditions to describe all the flows of botnet C&Cs. Therefore, these existing sampling algorithms maybe ineffective to sample packets for botnet C&Cs. In contrast to the above sampling algorithms, our algorithm is driven by the intrinsic characteristics of botnet C&Cs, and thus our sampling algorithm captures more botnet packets related flows given a certain sampling rate.

3. SYSTEM OVERVIEW

As shown in Figure 1, our botnet detection framework has three components: **Flow-Capture**, **Flow-Correlation**, and **Fine-Grained Detector**.

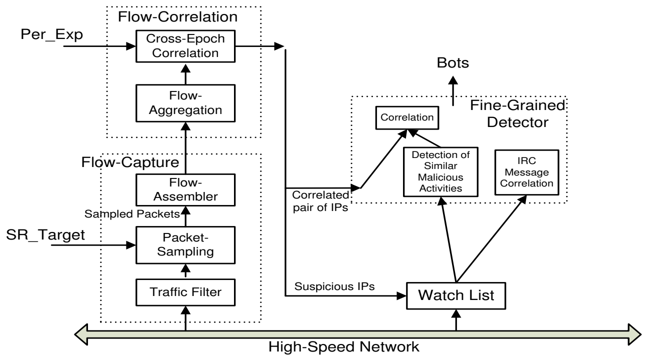

Figure 1: Architectural Overview

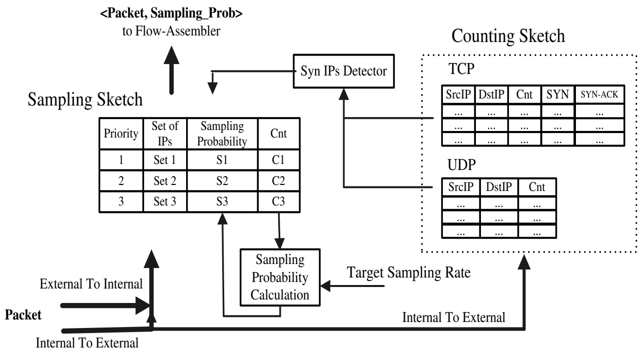

Figure 2: Packet Sampling Architecture

The Flow-Capture module aims to monitor the traffic at the edge of high-speed networks to gather network flow information, based on the sampled packets. The Flow-Capture module is divided in two components: **Packet-Sampling** and **Flow-Assembler**. Packet-Sampling is a botnet-aware sampling algorithm. Given an overall target sampling probability (SR_{Target}), it samples packets likely related to botnet C&C communications and delivers them to Flow-Assembler, along with their corresponding *instant* sampling probabilities (Section 4). The Flow-Assembler reconstructs flow information, and assembles the sampled packets into **raw flow**s (defined in Section 4.2).

The Flow-Correlation module groups flows output by Flow-Assembler into C-flows (defined in Section 5.1). A C-flow is an abstraction introduced in BotMiner [7] to represent the C&C communication patterns of potential bots. Each C-flow represents a view of the communication patterns from a monitored host to a remote service over a certain epoch (e.g, 12 hours). Flow-Correlation applies a scalable clustering algorithm over the C-flows to identify hosts that exhibit similar communication patterns towards machines outside the monitored network. This step is similar to the C-Plane analysis performed by BotMiner [7], but there are two fundamental differences. First, we use a significantly more efficient flow clustering process (see Section 5.2), compared to BotMiner, which can handle large traffic volumes typical of high-speed networks. Second, unlike BotMiner, our Flow-Correlation module performs *cross-epoch* correlation to identify hosts that show persistently similar communication pattens, a telltale sign of botnets. Any pair of hosts that exhibit persistently similar communication patterns will then be labeled as suspicious hosts (potential bots) and delivered to the Fine-Grained Detector for further in-depth analysis. The Fine-Grained Detector can then focus on monitoring the packets related to only the suspicious IPs provided by our Flow-Correlation module, thus reducing the overall cost of the botnet detection process.

The design and implementation of the Flow-Capture and Flow-Correlation modules and the detection framework are the main contributions of this work. Existing DPI-based botnet detectors can be plugged within our framework with little or no modification to constitute the Fine-Grained Detector module. We developed a Fine-Grained Detector derived from BotMiner [7] and BotSniffer [5], and we plugged it into our botnet detection framework. In particular, we used two components: i) an implementation of the malicious activities detector derived from BotMiner's A-Plane monitor, which can identify groups of similar malicious activities based on the attack features (e.g., the scanned port, the exploits or binary content), and ii) BotSniffer's IRC-based botnet detection module. Similar to the Cross-Plane correlation in BotMiner, the correlation component correlates communication patterns and activity patterns to detect bots. Any pair of IPs that share persistently similar communication patterns (generated by Flow-Correlation) and similar malicious activities (generated by the malicious activities detector) are labeled as bots by the correlation component. And any host identified by the BotSniffer's IRC-based botnet detection module will be labeled as bot.

4. FLOW CAPTURE

The Flow-Capture performs packet sampling and reassembles raw flows using a novel *botnet-aware* adaptive sampling algorithm, which we call **B-Sampling**. Our B-Sampling algorithm leverages the *intrinsic* characteristic of bots, namely *group similarity*, to guide the sampling procedure. Given a pre-defined target sampling rate, B-Sampling adaptively tunes the instant sampling probabilities for different categories of IPs. For example, priority will be given to packets related to IPs that share similar communication patterns, while keeping the overall sampling rate close to the overall target sampling rate SR_{Target}. The target sampling rate is usually suggested by the process capacity of the monitor device and the traffic speed of the monitored network. For example, the monitor device with capacity of Cap_{device}bps and the network with the speed of $Cap_{network}$bps indicate $SR_{Target} = \frac{Cap_{device}}{Cap_{network}}$.

4.1 Packet Sampling

As described in Figure 2, **Packet-Sampling** has four components: **Counting-Sketch**, **Sampling-Sketch**, **Synchronized IPs Detector** (**SID**), and **Priority-based Sampling Probability Calculation** (**PSPC**). Counting-Sketch tracks the number of packets sent from a SrcIP to a DstIP. After each time interval of T, Packet-Sampling transfers the Counting-Sketch to the SID, and then resets the Counting-Sketch to 0 for next interval. The end of each time interval also triggers the SID and PSPC to identify IP addresses with synchronized behaviors and recalculate the instant sampling probability for each category of IPs. Sampling-Sketch gets the instant sampling probability for a packet and decides whether this packet is going to be sampled.

4.1.1 Counting/Sampling Sketch

The Counting-Sketch is a table indexed by $Hash(SrcIP||DstIP)$ for TCP and UDP packets, where each entry in the table is defined as a **track-flow**. Each entry contains a pair of IPs ($SrcIP$ and $DstIP$) and a counter cnt, which represents the number of packets for this pair of IPs. For TCP packets, the entry keeps $SYN/SYNACK$ flag. On arrival of a packet, the SrcIP and DstIP will be recorded and the counter in the corresponding entry will be increased by 1. The Counting-Sketch only handles the packets from internal networks to external networks. Such design can simplify the system implementation by just monitoring the separated physical line for outgoing traffic. Moreover, it reduces the time and memory consumption to access the table. Counting-Sketch is reset to be 0 after the time interval T (currently 15 minutes).

Each entry in Sampling-Sketch records a category/set of IPs, a counter of packets related to these IPs, a sampling probability and a priority. On arrival of a packet, Sampling-Sketch checks the category of this packet based on its SrcIP and DstIP. It then finds the instant sampling probability (p_i) for the corresponding category and samples this packet with probability p_i. The sampled packets, together with their sampling probabilities, are sent to Flow-Assembler.

4.1.2 Synchronized IPs Detector

The SID identifies two kinds of hosts with synchronized behaviors: i) **syn-servers**: the hosts in external networks whose clients have similar network behaviors; ii) **syn-clients**: the hosts in internal networks that share similar network behaviors to multiple destination hosts.

The detection of syn-servers is motivated by the network behavior of C&C servers for centralized-based botnets, where their clients (bots) are synchronized and thus share similar network behaviors. For the legitimate servers, especially the popular ones, their clients' behaviors usually diverse from each other due to various usage patterns of different users. The detection of syn-clients is motivated by the network behaviors of P2P-based C&Cs. P2P-based bots usually actively query their peers to maintain the overlay P2P network for botnet C&Cs. Such behaviors will cause many similar connections to multiple peer bots.

To detect syn-servers and syn-clients, we introduce "homo-server" and "similar-client".

1. Homo-server: We aggregate entries in Counting-Sketch based on each DstIP. For each DstIP that has at least two SrcIPs, we calculate the variance of the track-flow sizes. We sort the variances and get the medium value v_{medium}. For one DstIP, if its variance $v_i < v_{medium}$, we mark it as a homo-server. Otherwise, we take the server as non-homo-server if it has at least two SrcIPs.

2. Similar-clients: We keep an array of bins (denoted as B in Algorithm 1) and a pre-defined size R (currently $R = 10$). Each bin b_i is represented by its center that is the average size of track-flows in this bin. For a flow with size L, if $|L - b_i.center| \leq R$, we insert this track-flow into b_i and then update the $b_i.center$. Otherwise, we build a new bin and insert this flow into it. In each bin, if we find a pair of SrcIPs and each of them has more than C (currently $C = 10$) flows (e.g., connecting to C different DstIPs), we take this pair of SrcIPs as similar-clients.

On identifying the syn-clients, we currently discard the TCP and UDP track-flows with size smaller than 10 to avoid potential false positives generated by popular network services like DNS or by the scanning-like behaviors. On identifying the homo-servers, we ignore the TCP track-flows with size of 1 or with only $SYNACK$ flag. A TCP track-flow with size of 1 indicates an unsuccessful connection. The flag of $SYNACK$ indicates a TCP connection initiated from external networks, which is unlikely a connection for botnet C&Cs. Bots usually initiate connections to external C&C servers for two reasons. First, the widely deployed firewall/NAT devices block the connections initiated from external networks. For example, researchers have shown that more than 40% `storm` bots are behind a firewall or NAT [11]. Second, the dynamic IPs make it very hard for C&C servers to initiate connections to bots with dynamical IPs accurately.

For each time interval T, we identify the homo-servers and similar-clients. We accumulate evidence over multiple intervals to decide whether a host is syn-server or syn-client. We keep each syn-server and syn-client in the Sampling Sketch for T_{rec} (currently $T_{rec} = E/2$, where E is one epoch of 12 hours) from its last update. The algorithm is described in Algorithm 1. $TH_{syn-server/client}$ is the threshold of the score to identify syn-server/client. TH_{down} is the lower bound of the score. $step_{up/down}$ is the step to increase/decrease the score. We set $TH_{syn-server/client} = 4$, $TH_{down} = -10$, $step_{up} = 1$, and $step_{down} = 0.2$. $Record$ represents one data structure for IP and time stamp. Arr is an array of scores indexed by the hosts and t_{cur} is the time stamp derived from current packet. If one record in Arr is not updated from its last update for T_{Arr} (currently $T_{Arr} = E/2$), we can eliminate it from Arr.

4.1.3 Sampling Probability Calculation

We dynamically calculate the sampling probability for each category of IPs to fulfill two targets: i) to get as many packets as possible that are related to syn-clients or syn-servers; ii) to keep the actual sampling rate close to the target sampling rate.

To keep the actual sampling rate close to the pre-defined sampling rate, a scheme for allocating instant sampling probabilities for different categories has been proposed by Ramachandran et al. [2]. However, this scheme requires pre-configured budgets for different categories. Inappropriate allocated budgets may affect the packet sampling process. For example, the inadequate budget for synchronized IPs will cause the lost of packets related to botnet, while the over-allocated budget for synchronized IPs would be a waste of the resources. To fully utilize the resources to capture packets, we design a sampling algorithm named **Priority-based Sampling Probability Calculation** algorithm. The principle for this algorithm is as follows: under a pre-defined sampling rate, we use the available resources (budget) to capture as many packets in the first priority category as possible. The remaining available resource will be used to capture as many packets as possible in the next level priority category. Such process will continue until there is no further category or no available resource. Algorithm 2 shows this approach. P_t is the pre-defined target sampling rate. $\{f_1, f_2, \ldots, f_n\}$ is the fraction of packets in each category where $priority_1 > priority_2 \cdots > priority_n$. $\{p_1, p_2, \ldots, p_n\}$

Algorithm 1: Identify Synchronized Hosts

```
Input: Counting Sketch, $Set_d$, $t_{cur}$
Output: $Set_d$: Records for syn-clients/servers.
begin
    foreach Record $R \in Set_d$ do
        if $t_{cur} - R.timestamp \geq T_{rec}$ then
            Remove $R$ from $Set_d$;
    foreach DstIP $dh_i$ in the Counting Sketch do
        if $dh_i$ is homo-server then
            $Arr.get(dh_i).score+ = step_{up}$ ;
            if $Arr.get(dh_i).score \geq TH_{syn-server}$ then
                $set_d.add(dh_i, t_{cur})$;
                $Arr.get(dh_i).score = TH_{syn-server}$;
        if $dh_i$ is non-homo-server then
            $Arr.get(dh_i).score- = step_{down}$ ;
            if $Arr.get(dh_i).score \leq TH_{down}$ then
                $Arr.get(dh_i).score = TH_{down}$;
    foreach SrcIP $sh_i$ in the Counting Sketch do
        if $sh_i$ is similar-client then
            $Arr.get(sh_i).score+ = step_{up}$ ;
            if $Arr.get(sh_i).score \geq TH_{syn-client}$ then
                $set_d.add(sh_i, t_{cur})$;
                $Arr.get(sh_i).score = TH_{syn-client}$;
        else
            $Arr.get(sh_i).score- = step_{down}$ ;
            if $Arr.get(sh_i).score \leq TH_{down}$ then
                $Arr.get(sh_i).score = TH_{down}$;
    return $Set_d$;
end
```

is a set of instant sampling rates for different priorities and *budget* is for the available budget.

The following equation illustrates how the budget allocation helps the sampling component to keep a target sampling rate. Suppose there are a total of K packets and the target sampling rate is P_t. Given n categories and suppose each category has f_i fraction of the total packets and we give budget b_i to this category, we can calculate the sampling probability for category i as $p_i = P_t \frac{b_i}{f_i}$. In this case, the number of sampled packets Q and overall sampling rate would be $Q = \sum_{i=1}^{n} K f_i p_i = K \sum_{i=1}^{n} f_i (P_t \frac{b_i}{f_i}) = K P_t \sum_{i=1}^{n} b_i$. According to this equation, as long as $\sum_{i=1}^{n} b_i = 1$, the overall sampling rate $\frac{Q}{K}$ would be P_t, the target sampling rate. Since f_i cannot be obtained precisely in advance, we dynamically estimate f_i using WMA (weighted moving average) based on the observed value for it in the previous and current intervals, which is $f_i = w_1 f_i^{prev} + w_2 f_i^{curr}$ where $w_1 = 0.2$ and $w_2 = 0.8$ in our current design. The system can dynamically assign $priority_1$ or $priority_2$ to syn-servers or syn-clients. The fewer the packets related to one of these two categories, the higher priority it has. The intuition behind such design is to use enough resource to build the accurate flows for the category that requires least resource. In practice, operators can also fix the priority or introduce more categories/priorities based on known knowledge (e.g., a category for the packets that are sent to confirmed bot peers). The packets related to the rest of IPs are labeled as the lowest priority ($priority_3$).

4.2 Flow Assembler

The Flow-Assembler assembles sampled packets to generate **raw flows**, where each raw flow is identified by 5-tuple key (SrcIP, SrcPort, DstIP, DstPort, Proto). For TCP flow, the first two handshake packets (SYN and $SYNACK$) can be used to identify the flow direction. However, since packet sampling may result in the loss of TCP handshake packets, we use following approaches to identify TCP flow direction. First, if one of these two handshake packets is sampled, we can easily identify the flow direction. Second, for a TCP flow without TCP handshake packets sampled, we take this flow as it is initiated from internal networks (e.g., its SrcIP is from internal network). These approaches guarantee that every TCP flow from internal network will be attributed to the correct direction. Flow-Assembler outputs a flow if the flow is finished (e.g., the TCP FIN/RST flag is observed) or it expires (e.g., no packet comes for this flow for 10 minutes). For one raw flow, we record information including $time_{Start}$, $time_{End}$, $size_{Actual}$ (# of packets observed), $byte_{Actual}$ (# of bytes observed) and $size_{Est}$. $size_{Est}$ is the estimated flow size based on the sampled packets and their corresponding instant sampling probabilities. Suppose there are n packets for one raw flow and each packet has b_i bytes and sampling probability of p_i, we compute the metrics for this raw flow as follows: $size_{Est} = \sum_{i=1}^{n} \frac{1}{p_i}$, $size_{Actual} = n$, $byte_{Actual} = \sum_{i=1}^{n} b_i$.

Algorithm 2: Priority-based Sampling Algorithm

```
Input: $P_t$, $f_1, f_2, \ldots, f_n$
Output: $p_1, p_2, \ldots, p_n$
begin
    $budget = 1$;
    foreach $i = 1 \ldots n$ do
        if $f_i == 0$ or $budget \leq 0$ then
            $p_i = 0$;
            continue;
        else
            $p_i = budget * \frac{P_t}{f_i}$;
            $p_i = p_i > 1?1 : p_i$;
            $budget- = p_i * \frac{f_i}{P_t}$;
    return $\{p_1, p_2, \ldots, p_n\}$;
end
```

5. FLOW CORRELATION

The goal of Flow-Correlation is to identify hosts with persistently similar communication patterns. By evaluating the capacity of the fine-grained detectors and the monitored network, operators can estimate the percentage of hosts Per_{Exp} (as described in Figure 1) that fine-grained detectors can afford to monitor. For example, if we assume that the traffic is evenly distributed over the hosts in the monitored network, the capacity of a fine-grained detector ($Cap_{detector}$bps) and the network speed ($Cap_{network}$bps) indicate a $Per_{Exp} = \frac{Cap_{detector}}{Cap_{network}}$. The Flow-Correlation component identifies groups of hosts (up to Per_{Exp}) that share most similar communication patterns and show persistence.

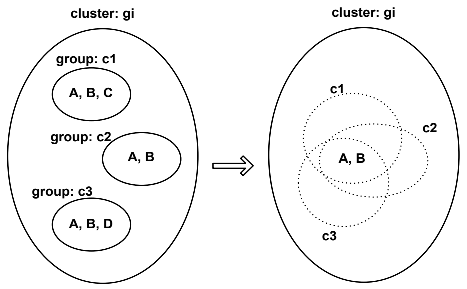

Figure 3: An Example of Cross-Epoch-Correlation

5.1 Flow Aggregation

We use **C-flow** to represent the communication pattern from a host to a remote host and port. We define a C-flow as a set of raw flows sharing same tuple of (SrcIP, DstIP, DstPort, Proto) in a certain epoch E (currently $E = 12$hours), denoted as $c = \{f_1, \ldots, f_n\}$. To get C-flows, we filter out the raw flows that satisfy either of two conditions: i) The raw flow is initiated from external network to internal network, where the reason is discussed in Section 4.1. ii) The raw flow has traffic in only one direction, which indicates an unsuccessful connection. We represent a C-flow ($c = \{f_1, \ldots, f_n\}$) using the following 10 features.

1. The $mean$s and $variance$s of fph (the number of flows per hour), ppf (the number of packets per flow), *bpp* (the number of bytes per packet), *pps* (the number of packets per second), which have similar definition in BotMiner [7]. We use $size_{Est}$ to compute ppf and *pps*, while $\frac{byte_{Actual}}{size_{Actual}}$ is used for *bpp*.
2. fph_{max}: the maximum number of flows per hour.
3. $time_m$: the median time interval of two consecutive flows.

5.2 Cross-Epoch Correlation

Given Per_{Exp}, cross-epoch correlation identifies pairs of IPs where each pair shares persistently similar communication patterns for at least M epochs out of totally N epochs ($M \leq N$).

We get a set (G) of C-flows over multiple epochs, and each C-flow has an epoch-tag. After clustering C-flows, we get a set of clusters $\{g_1, g_2, \ldots, g_n\}$ where each cluster g_i represents a set of similar communication patterns ($G = g_1 \cup g_2 \cup \cdots \cup g_n$). For C-flows in one cluster g_i, we further aggregate them into different groups (denoted as $\{c_i^1, c_i^2 \ldots c_i^N\}$ and $g_i = c_i^1 \cup c_i^2 \cdots \cup c_i^N$) according to their epoch-tags. For example, c_i^j represents the C-flows that are similar in jth epoch (spatial-similarity). For each cluster g_i, if a pair of SrcIPs share at least M common groups, it indicates that they share persistently similar communication patterns over at least M epochs. Therefore, we label this pair of SrcIPs as suspicious. We denote the percentage of all the detected suspicious IPs over all the SrcIPs as Per. Figure 3 presents an example of cross-epoch correlation. $A/B/C/D$ is the C-flow associated with the host $h_A/h_B/h_C/h_D$, and the remote host and port of $A/B/C/D$ are not necessarily to be the same over multiple epochs (e.g., A represents $< h_A, h_{remote}, port_{remote} >$). Some similar C-flows associated with $h_A/h_B/h_C/h_D$ are clustered together in a cluster g_i. By investigating the epoch-tag related to each C-flow, we aggregate these C-flows to three groups ($c^1/c^2/c^3$), as described in the left part of Figure 3. The right part of Figure 3 presents that h_A and h_B share 3 common groups, which indicates that they share similar communication patterns for 3 epochs. If we set $M \leq 3$, h_A and h_B are labeled as suspicious.

To get clusters of C-flows that represent similar communication patterns, we use clustering algorithm. BotMiner uses two-level clustering scheme (X-Means and Hierarchical) that cannot scale well for large number of C-flows as shown in Figure 7. To process C-flows in an efficient manner, we use a scalable clustering algorithm Birch [24]. Given a certain value of "diameter", Birch can first efficiently discover clusters of C-flows within such distance. Second, cross-epoch correlation can detect suspicious IPs based on the clustering results. We repeat these two steps by increasing the value of "diameter". This process terminates when the percentage of suspicious IPs Per for the next step reaches at the expected percentage Per_{Exp} or the number of rounds reaches at a pre-defined $MaxRound$ (currently 50).

Trace	# of Pkts	Dur	Info
Mar25	205,079,914	12h	header
Mar26	280,853,924	24h	header
Mar27	318,796,703	24h	header
Mar28	444,260,179	24h	header
Mar31	102,487,409	1.5h	full

Table 1: Background Traces

Trace	Dur	Bots
Bot-IRC-A	4days	3
Bot-IRC-B	4days	4
Bot-HTTP-A	4days	3
Bot-HTTP-B	4days	4
Bot-HTTP-C	4days	4
Bot-P2P-Storm	4days	2
Bot-P2P-Waledac	4days	3

Table 2: Botnet traces

6. EVALUATION

We implemented a prototype system and evaluated it using traces of real-world network traffic and different botnets. The results show that Flow-Capture can achieve a significantly higher sampling rate for botnet-related packets compared to the pre-defined sampling rate. We compared B-Sampling to FlexSample, and the experimental results indicate that B-Sampling outperforms FlexSample regarding sampling rate for botnet packets and detection rate of Flow-Correlation. The cross-epoch correlation can effectively and efficiently identify bots given a small percentage of suspicious hosts. The fine-grained detector can achieve high detection rate and low false positive rate by only inspecting packets related to a small percentage of suspicious hosts.

6.1 Experiment Setup and Data Collection

We mounted our monitors on a span port mirroring a backbone router at the college network (200Mbps-300Mbps at daytime) to collect data. The traffic covers various applications and we believe such kind of traffic provides good traces to evaluate our system. The dataset contains TCP and UDP headers for continuous 3.5 days and full packets for 1.5 hours in Table 1. We eliminated a `B/16` subnet for dynamic IPs allocated for wireless connections, which are frequently changed and can not accurately represent the same hosts for multiple epochs. We observed a total of 1460 different IP addresses in 3.5 days. We also collected 1.5 hour traces with full payload.

We collected the traces of 7 different botnets including IRC-, HTTP- and P2P-based botnets, as described in Table 2. `Bot-IRC-A` and `Bot-HTTP-A` were collected by running bot instances ("TR/Agent.1199508.A" and "Swizzor.gen.c") in multiple hosts in the honeypot. `Bot-IRC-B` and `Bot-HTTP-B/C` were generated using *Rubot* [13], a botnet emulation framework. In `Bot-HTTP-B`, bots periodically contacted the C&C server every 10 minutes. And in `Bot-HTTP-C`, the bots contacted the C&C server in a more stealthy way by adding a random time interval between 0 to 10 minutes on each time of visiting. Both of them conducted scanning attack on receiving the "scan" command. Bots in `Bot-IRC-A` send packets much more frequently to C&C server in the IRC session, resulting in much larger C&C flows compared to `Bot-IRC-B`. We collected traces of two P2P-based botnets, *Storm* [7] and *Waledac* [10], by running binaries in the controlled environment.

After aligning the timestamp of each packet in botnet traces according to the time of the first packet in background traces, we mixed 3.5 consecutive days of botnet traces into the college traces by overlaying them to randomly picked client IPs in college network. We took one epoch E as 12hr so there are 7 epochs in total. The filter covers major local DNS, email servers in the college, the IP ranges of the popular service networks (e.g., MICROSOFT, GOOGLE, YAHOO, SUN, etc.), popular content distribution networks (e.g., AKAMAI) , whose IP ranges are unlikely to be used for Botnet C&Cs, and IPs of top 10000 *alexa* domains (corresponding to 12230 IPs).

6.2 Evaluation of Sampling Algorithm

We evaluated B-Sampling algorithm using the mixed traces with different target sampling rates (0.01, 0.025, 0.05, 0.075 and 0.1). We compared B-Sampling to FlexSample [2], a state-of-the-art sampling algorithm that can be configured with different "conditions" for different purposes. FlexSample used a specific condition (Figure 10 in FlexSample [2]) to capture botnet packets by allocating the majority of budgets to packets related to "servers with high indegree of small flows". However, since the number of infected machines could be small in real-world, the "high fan-in" feature may not hold and thus will probably miss the botnet packets. As illustrated in Table 8 in Appendix A, this condition causes very low sampling rates on botnet packets in our traces. Therefore, we modify the condition and only use the condition related to flow size for FlexSample. We configured FlexSample using a condition presented in Table 7 with $(size \leq 20, budget = 0.95)$, which means that FlexSample uses 95% resource to capture the packets in flows with sizes smaller than 20.

Table 3 presents the overall sampling rates and sampling rates for botnet-related packets on the mixed dataset, using both B-Sampling and FlexSample. The first column (SR_T) reports the pre-defined target sampling rates we experimented with. The second column (SR_{Actual},B) and the third column (SR_{Actual},$Flex$) report the actual overall sampling rates achieved by B-Sampling and FlexSample. The results show that both B-Sampling and FlexSample keep the actual sampling rate close to the target sampling rate. The remaining columns report the sampling rates related to different types of botnet-related packets, where we "zoom" in the sampled packets and evaluate the actual sampling rates for packets of each botnet. For example, the $4th$ column ($SR_{IRC-A/B}$,B) reports the actual sampling rate for packets in Bot-IRC-A and Bot-IRC-B using B-Sampling, whereas the $5th$ column ($SR_{IRC-A/B}$,$Flex$) presents the sampling rate using FlexSample. We can find that B-Sampling captures a higher percentage of botnet packets, compared to FlexSample. For example, considering the second row (target sampling rate is 0.025), B-Sampling achieves a sampling rate of 0.93 ($SR_{IRC-A/B}$, B column) while FlexSample achieves that of 0.002 ($SR_{IRC-A/B}$, $Flex$ column) for packets in `Botnet-IRC-A`, where the C&C flows are large flows. The remaining columns report a comparison of B-Sampling and FlexSampling on the sampling rates for other botnets. As we can see, B-Sampling achieves higher sampling rate for botnet-related packets, compared to FlexSample. It is possible to increase the flow size in the FlexSample condition or reduce the budget for small flows to make FlexSample capture more packets in `Botnet-IRC-A`. However, it will cause FlexSample to decrease the sampling rates for packets related to botnets whose C&Cs are small flows such as `Bot-HTTP-` and `Bot-P2P-`. The reason is that the feature of flow size and server indegree are not intrinsic for botnets and different botnets can diverse greatly regarding these features. B-Sampling gave higher sampling rate for packets in `Bot-IRC-` and `Bot-HTTP-` than those in `Bot-P2P-`, because that the number of packets related to syn-server is much smaller than that related to syn-clients, and thus syn-servers have higher priority as illustrated in Section 4.1.3.

We evaluated the parameters, C and $step_{up}$, in the B-Sampling algorithm in Section 4.1. Given $SR_T = 0.05$, we report the experimental results in Table 9 in Appendix A. The results demonstrate that the results of B-Sampling are stable over these values.

6.3 Evaluation of Flow Correlation

We evaluated the cross-epoch correlation with B-Sampling using the mixed traces for two properties, detection accuracy and scalability. We set $M = \llcorner \frac{N}{2} \lrcorner$ $(N = 7, M = 3)$, which means that two hosts sharing similar communication patterns for any 3 out of 7 epochs will be labeled as suspicious.

Given SR_T and Per_{Exp}, each cell in Table 4 shows the detection rate of bots(/23) and percentage of noises(/1460) identified by Flow-Correlation using B-Sampling. The results show that Flow-Correlation can achieve high detection rate with low Per_{Exp}. For example, with $Per_{Exp} \geq 5\%$, for all the SR_T evaluated, Flow-Correlation can successfully identify all the bots. While for the very low Per_{Exp} (e.g., 2% and 3%), more than half of the bots were still captured. We also compared the detection rate of Flow-Correlation using B-Sampling to that of Flow-Correlation using FlexSample (in Table 10). Figure 4 illustrates the average detection rates over different Per_{Exp} for each target sampling rate, and Figure 5 and Figure 6 present the detection rates using B-Sampling and FlexSample with Per_{Exp} of 0.01 and 0.05. The comparison results show that by using B-Sampling, Flow-Correlation can achieve higher detection rate.

Figure 7 presents the time consumption (in a 4G memory and 2-core CPU computer) for cross-epoch correlation and the C-Plane clustering of BotMiner as the number of C-flows

SR_T	SR_{Actual}		$SR_{IRC-A/B}$		$SR_{HTTP-A/B/C}$		SR_{Storm}		$SR_{Waledac}$	
	B-	Flex	B-	Flex	B-	Flex	B-	Flex	B-	Flex
0.01	0.012	0.01	0.65/0.68	0.001/0.07	0.55/0.69/0.68	0.06/0.07/0.06	0.02	0.05	0.02	0.07
0.025	0.027	0.025	0.93/0.92	0.002/0.16	0.72/0.93/0.93	0.16/0.17/0.16	0.16	0.11	0.18	0.16
0.05	0.052	0.05	0.96/0.96	0.004/0.32	0.74/0.96/0.96	0.32/0.35/0.33	0.48	0.23	0.48	0.33
0.075	0.076	0.075	0.97/0.97	0.006/0.48	0.75/0.97/0.97	0.50/0.50/0.48	0.72	0.33	0.7	0.48
0.1	0.1	0.1	0.98/0.98	0.008/0.6	0.76/0.98/0.98	0.6/0.64/0.61	0.83	0.41	0.81	0.61

Table 3: Sampling Rate

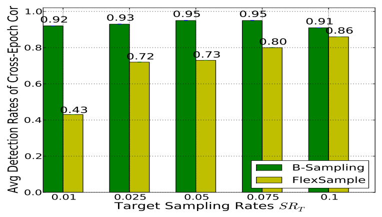

Figure 4: Average detection rate for cross-epoch correlation, over different Per_{Exp}

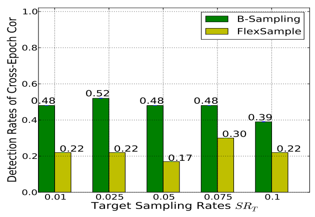

Figure 5: Detection rate for cross-epoch correlation, $Per_{Exp} = 0.01$

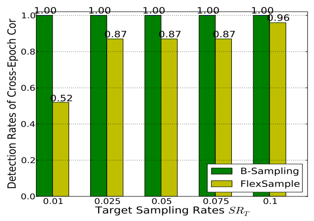

Figure 6: Detection rate for cross-epoch correlation, $Per_{Exp} = 0.05$

increases. We configured `Birch` to run $MaxRound = 50$ to simulate the process of identifying up to Per_{Exp} suspicious hosts. The exponential time increment for C-Plane clustering of BotMiner indicates its limited scalability. The cross-epoch correlation shows linear pattern and its linear regression model is $t = 0.0035x$.

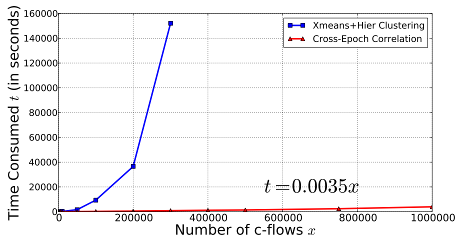

Figure 7: Scalability of Cross-Epoch Correlation

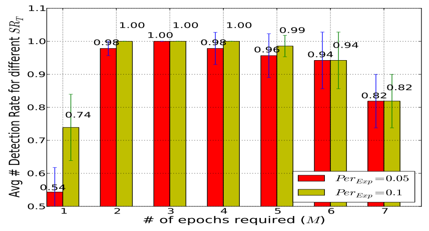

Figure 8: Avg detection rate (over SR_Ts) of Cross-Epoch Correlation using B-Sampling

Figure 8 presents the mean and standard deviation for detection rates by Flow-Correlation with B-Sampling for different M, given Per_{Exp} (5% or 10%) for all SR_T. First, the results demonstrate the effectiveness of cross-epoch correlation. When no cross-epoch correlation is used ($M = 1$), many legitimate IPs show stronger similarity than bots in a single epoch. Therefore, given a certain Per_{Exp}, more than 50% bots are missed. While cross-epoch correlation can effectively eliminate these legitimate IPs that show strong similarity in one epoch but do not have persistently similar patterns. For example, cross-epoch correlation with $M = 2$ can successfully detect most bots. Second, the results indicate that cross-epoch correlation is not sensitive to the value of M. For example, for $M = 3/4/5$, the cross-epoch correlation achieves similar detection rate. Such observation also indicates that $\frac{N}{2}$ is a good value for M.

6.4 Botnet Detection

Fine-grained botnet detector inspects all the packets related to suspicious IPs detected by Flow-Correlation. Using 1.5hr trace mixed with botnet traces, we evaluated the detection rate and performance of the fine-grained detector.

By analyzing the similarity among IRC messages, "IRC Message Correlation" component in our detector detected bots in `Bot-IRC-A/B`. Other bots were detected by the "Correlation" component. For example, Bots in `Bot-HTTP-B/C` trigger alerts when they scan the local network. `Bot-HTTP-A` bots trigger alerts when they make update requests. `Storm` and `Waledac` trigger alerts when they discover peers. These bots were detected by correlating such activities/alerts with corresponding pairs of IPs from Flow-Correlation. Table 5 presents the detection rates and false positive rates for the fine-grained detector for different SR_Ts and Per_{Exp}s. The corresponding cells in Table 11 in Appendix A present the percentage of packets that our fine-grained detector needs to inspect. For most combinations of SR_T and Per_{Exp}, our framework can reduce traffic volume by more than 90% for fine-grained detector but still keep high detection rates and low false positives. For example, for $SR_T = 0.01$ and $Per_{Exp} = 0.05$, the fine-grained detector can detect all bots with false positive of 0, and it only needs to focus on 1.7% percentage of packets.

	With Flow-Corr ($Per_E = 5\%, M = 3$)						direct
SR_T	0.01	0.025	0.05	0.075	0.1	1	
Per of Pkts	1.7%	2.9%	2.1%	3%	4.3%	2%	100%
Time	33s	39s	35s	40s	49s	33s	858s

Table 6: Performance of Fine-Grained Detector

Table 6 presents the performance comparison, including the percentage of packets inspected and the processing time of the fine-grained detector in two situations: i) the detector is directly applied, ii) the detector is applied with Flow-

SR_T	For each Per_{Exp}, TP(bots/23), FP(noises/1460)									
	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
0.01	48%, 0.1%	83%, 0.5%	96%, 1%	96%, 2%	100%, 3%	100%, 4%	100%, 5%	100%, 6%	100%, 6%	100%, 8%
0.025	52%, 0%	87%, 0.5%	100%, 1%	100%, 2%	100%, 3%	100%, 4%	100%, 5%	100%, 6%	100%, 7%	100%, 8%
0.05	48%, 0.1%	100%, 0.3%	100%, 1%	100%, 2%	100%, 3%	100%, 4%	100%, 5%	100%, 5%	100%, 7%	100%, 7%
0.075	48%, 0.2%	100%, 0.3%	100%, 1%	100%, 2%	100%, 3%	100%, 4%	100%, 5%	100%, 6%	100%, 7%	100%, 8%
0.1	39%, 0.3%	78%, 0.8%	100%, 1%	100%, 2%	100%, 3%	100%, 3%	100%, 5%	100%, 5%	100%, 7%	100%, 8%
1	30%, 0.5%	65%, 0.8%	96%, 1%	100%, 2%	100%, 3%	100%, 4%	100%, 5%	100%, 5%	100%, 7%	100%, 8%

Table 4: Detection Rates of Cross-Epoch Correlation using B-Sampling

SR_T	For each Per_{Exp}, TP(bots/23), FP(noises/1460)									
	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
0.01	48%, 0	83%, 0	96%, 0	96%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0
0.025	52%, 0	87%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0
0.05	48%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0
0.075	48%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0
0.1	39%, 0	78%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0
1	30%, 0	65%, 0	96%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0	100%, 0

Table 5: Detection Rates of Fine-Grained Detectors

Correlation and B-Sampling ($Per_{Exp} = 0.05$ and $M = 3$). By using Flow-Correlation, fine-grained detector to reduce 95% time to process off-line traces, indicating a great workload reduction in real time.

7. DISCUSSION

To answer the question "how high speed networks our approach can handle?", we consider the performance of two key components, B-Sampling and cross-epoch correlation. B-Sampling is intended to be implemented with hardware support, where we can design the Counting-Sketch and Sampling-Sketch in fast memory (e.g., SRAM) while the SID and PSPC in slow memory (e.g., DRAM). The system can periodically but parallel read the data from SRAM to DRAM for identifying synchronized hosts and computing sampling probabilities, and then write the sets of IPs to SRAM. And for Counting-Sketch, recent study has shown the hardware implementation of a specific hash function with a throughput of over 10Gbps [3], indicating the potential performance of 10Gbps of B-Sampling with hardware implementation. Given an expected time consumption of 2hr for cross-epoch correlation, the linear model $t = 0.0035x$ (in seconds) implies 2M C-flows. If we assume the number of C-flows is proportional to the traffic volume (e.g., 200K C-flows in our experiment is corresponding to 200Mbps), 2M C-flows correspond to a network with speed of 2Gbps. Since 2Gbps is less than the potential performance of 10Gbps of B-Sampling, such results indicate that our approach can be used in 2Gbps networks (e.g., campus backbone networks) and has the potential to be deployed in faster network as the expected time consumption of cross-epoch correlation increases.

Because of our assumptions on the persistent use of coordinated C&Cs in a botnet, any evasion attempts that violate our assumptions will likely succeed if the botmaster knows our algorithms, similar to any evasion attacks against an IDS. Bots may intentionally manipulate their communication patterns to decrease sampling probabilities or evade cross-epoch correlation. For example, bots can randomize communication patterns (e.g., number of packets per flow) to evade the syn-client/server detection. One potential solution is to dynamically tune the parameters used for identifying syn-servers and syn-clients for each round (e.g., randomly select $\frac{1}{4}$, $\frac{3}{4}$ quantiles or medium value of variances for identifying syn-server, and choose R and C from a predefined set of values/ranges for identifying syn-clients). Another solution is for B-Sampling to incorporate information from other systems. For example, we can set a category of IPs in rouge networks [16] or malicious fast-flux networks, which are likely related to botnets, to sample more related packets. For cross-epoch correlation, we can incorporate more detection features (e.g., using packet payload information for some tight clusters to do light-weight content checking) to make the evasion more difficult. Due to the nature of the arms race in existing intrusion detection and evasion practice, we should always study better and more robust techniques as a defender. Combining different complementary detection techniques to make the evasion harder is one possible future direction. We leave a deeper and more extensive study to handle these evasion attempts as future work.

8. CONCLUSION

Botnet detection in high-speed and high-volume networks is a challenging problem. Given the severity of botnets and the growing interest from ISPs to defend against botnets, research on botnet detection in high-speed and high-volume networks is important. In this paper, we have described a solution to this problem, which includes a botnet-aware adaptive packet sampling algorithm and a scalable spatial-temporal flow correlation approach. The adaptive packet sampling technique uses network characteristics of botnet C&Cs to capture more packets related to bots and adaptively tune the sampling probabilities to keep a target sampling rate. The flow correlation approach exploits the essential properties of botnets and detects bots by identifying hosts with persistently similar communication patterns. Based on evaluation using real-world network traces shows that our proposed solution yields good performance. The sampling algorithm can capture more botnet packets in comparison to pre-defined sampling rate and outperforms the state-of-the-art adaptive sampling algorithms. Based on the sampled packets, the correlation algorithm can successfully and scalably pinpoint various types of bots (including IRC-based, HTTP-based, and P2P-based). This approach will help the fine-grained botnet detectors to focus on inspecting packets of a smaller amount of suspicious traffic, thus allowing them to operating on increasingly more high-speed networks.

Acknowledgments

We thank Anirudh Ramachandran for providing the code for FlexSample and Chris Lee for the help in providing the code for Rubot. We also wish to thank the anonymous reviewers for their insightful comments and feedback. This material is based upon work supported in part by the National Science Foundation under grants No. 0831300 and No. CNS-0954096, the Department of Homeland Security under contract No. FA8750-08-2-0141, and the Office of Naval Research under grants No. N000140710907, No. N000140911042 and No. N000140910776. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Department of Homeland Security, or the Office of Naval Research.

9. REFERENCES

[1] A. Kumar and J. Xu. Sketch guided sampling – using on-line estimates of flow size for adaptive data collection. In *Proc. IEEE Infocom*, 2006.

[2] A. Ramachandran, S. Seetharaman, and N. Feamster. Fast monitoring of traffic subpopulations. In *Proc. ACM IMC*, 2008.

[3] R. K. B. Yang and D.A.McGrew. Divide and concatenate: An architectural level optimization technique for universal hash functions. In *Proc. of the Design Automation Conference*, 2004.

[4] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *Proc. USENIX SRUTI*, 2006.

[5] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proc. NDSS*, 2008.

[6] G. Gu, P. Porras, V. Yegneswaran, M. Fong, W. and Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *Proc. USENIX Security*, 2007.

[7] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. USENIX Security*, 2008.

[8] J. Goebel and T. Holz. Rishi: identify bot contaminated hosts by irc nickname evaluation. In *Proc. USENIX HotBots*, 2007.

[9] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *Proc. IEEE Infocom*, 2008.

[10] Infosecurity. Storm deadnet reanimates as waledac botnet. `http://infosecurity.us/?p=6262`, 2009.

[11] B. Kang, E. C. Tin, and C. P. Lee. Towards complete node enumeration in a peer-to-peer botnet. In *Proc. ACM AISACCS*, 2009.

[12] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. USENIX HotBots*, 2007.

[13] C. P. Lee. *FRAMEWORK FOR BOTNET EMULATION AND ANALYSIS*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, Nov. 2008.

[14] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. Using machine learning techniques to identify botnet traffic. In *Proc. IEEE WoNS*, 2006.

[15] A. Ramachandran, N. Feamster, and D. Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *Proc. USENIX SRUTI*, 2006.

[16] B. Stone-Gross, A. Moser, C. Kruegel, E. Kirda, and K. Almeroth. Fire: Finding rogue networks. In *Proc. ACSAC*, 2009.

[17] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proc. IEEE LCN*, 2006.

[18] T.-F. Yen and M. K. Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *ICDCS*, 2010.

[19] L. P. Wenjia Fang. Inter-as traffic patterns and their implications. In *IEEE Global Internet Symposium*, 1999.

[20] X. Hu, M. Knysz and K. Shin. Rb-seeker: Auto-detection of redirection botnets. In *Proc. NDSS*, 2009.

[21] Y. Zhang, S. Singh, S. Sen, N. Duffield and C. Lund. Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications. In *Proc. ACM IMC*, 2004.

[22] Y. Zhao and Y. Xie and F. Yu and Q. Ke and Y. Yu. Botgraph: Large scale spamming botnet detection. In *Proc. USENIX NSDI*, 2009.

[23] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proc. DIMVA*, 2008.

[24] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD*. ACM Press, 1996.

APPENDIX

A. TABLES

```
vars = 1
conditions = 1
var_1 := srcip.srcport.dstip.dstport.prot
counter var_1 := 8, 1000000, 0.01, 30
var_1 in (0, 20]: 0.95
```

Table 7: Condition for FlexSample

SR_T	$SR_{I-A/B}$	$SR_{H-A/B/C}$	SR_{Storm}	$SR_{Waledac}$
0.025	0.003/0.01	0.013/0.011/0.01	0.006	0.008
0.05	0.006/0.018	0.023/0.019/0.017	0.012	0.015

Table 8: Sampling Rate using condition in Figure 10 in FlexSample [2]

C, $step_{up}$	SR_{Actual}	$SR_{I-A/B}$	$SR_{H-A/B/C}$	SR_{Storm}	$SR_{Waledac}$
10, 0.8	0.051	0.97/0.96	0.71/0.96/0.96	0.50	0.49
10, 0.5	0.051	0.96/0.95	0.61/0.96/0.95	0.51	0.51
10, 1.2	0.052	0.96/0.96	0.77/0.96/0.96	0.46	0.46
5, 1	0.052	0.96/0.96	0.74/0.96/0.96	0.46	0.46
15, 1	0.052	0.96/0.96	0.74/0.96/0.96	0.48	0.48

Table 9: Sampling Rate using Different Parameters

SR_T	For each Per_{Exp}, TP(bots/23), FP(noises/1460)									
	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
0.01	22%, 0.6%	30%, 2%	30%, 2%	39%, 3%	52%, 4%	52%, 5%	52%, 6%	52%, 7%	52%, 8%	52%, 8%
0.025	22%, 0.6%	39%, 1%	52%, 2%	87%, 3%	87%, 3%	87%, 5%	87%, 6%	87%, 7%	87%, 7%	87%, 8%
0.05	17%, 0.6%	43%, 1%	70%, 2%	87%, 3%	87%, 4%	87%, 4%	87%, 5%	87%, 7%	87%, 7%	87%, 7%
0.075	30%, 0.4%	57%, 1%	83%, 2%	87%, 3%	87%, 3%	87%, 4%	87%, 6%	96%, 6%	96%, 7%	96%, 8%
0.1	22%, 0.3%	65%, 1%	83%, 2%	96%, 2%	96%, 3%	100%, 4%	100%, 5%	100%, 6%	100%, 7%	100%, 8%

Table 10: Detection Rates of Cross-Epoch Correlation using FlexSample

SR_T	For each Per_{Exp}, Percentage of Packets									
	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.1
0.01	0.1%	0.4%	1%	1.5%	1.7%	3.3%	3.5%	4.1%	4.2%	5%
0.025	0.2%	0.7%	1.2%	2.6%	2.9%	3.5%	3.8%	4%	4.2%	6%
0.05	0.6%	0.6%	1%	1.8%	2.1%	2.2%	2.7%	2.7%	3.3%	3.5%
0.075	0.6%	0.6%	2%	3%	3.2%	3.8%	4.5%	4.5%	4.4%	5%
0.1	0.2%	0.9%	1.3%	3.7%	4.3%	4.3%	4.6%	4.6%	5.5%	6.2%
1	0.7%	0.6%	1%	1.7%	1.9%	3.4%	3.3%	3.3%	4.9%	6.1%

Table 11: Percentage of Packets Investigated by Fine-Grained Detectors