Building a Scalable System for Stealthy P2P-Botnet Detection

Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo, and Unum Sarfraz

***Abstract*— Peer-to-peer (P2P) botnets have recently been adopted by botmasters for their resiliency against take-down efforts. Besides being harder to take down, modern botnets tend to be stealthier in the way they perform malicious activities, making current detection approaches ineffective. In addition, the rapidly growing volume of network traffic calls for high scalability of detection systems. In this paper, we propose a novel scalable botnet detection system capable of detecting stealthy P2P botnets. Our system first identifies all hosts that are likely engaged in P2P communications. It then derives statistical fingerprints to profile P2P traffic and further distinguish between P2P botnet traffic and legitimate P2P traffic. The parallelized computation with bounded complexity makes scalability a built-in feature of our system. Extensive evaluation has demonstrated both high detection accuracy and great scalability of the proposed system.**

***Index Terms*— Botnet, P2P, intrusion detection, network security.**

I. Introduction

A BOTNET is a collection of compromised hosts (a.k.a. bots) that are remotely controlled by an attacker (the botmaster) through a command and control (C&C) channel. Botnets serve as the infrastructures responsible for a variety of cyber-crimes, such as spamming, distributed denial-of-service (DDoS) attacks, identity theft, click fraud, etc. The C&C channel is an essential component of a botnet because botmasters rely on the C&C channel to issue commands to their bots and receive information from the compromised machines. Botnets may structure their C&C channels in different ways. In a centralized architecture, all bots in a botnet contact one (or a few) C&C server(s) *owned* by the botmaster. However, a fundamental disadvantage of centralized C&C servers is that they represent a *single point of failure*. In order to overcome this problem, botmasters have recently started to build botnets with a more resilient C&C architecture, using a peer-to-peer (P2P) structure [1]–[3] or hybrid P2P/centralized C&C structures [4]. Bots belonging to a P2P botnet form an overlay network in which any of the nodes (i.e., any of the bots) can be used by the botmaster to distribute commands to the other peers or collect information from them. Notable examples of P2P botnets are represented by `Nugache` [5], `Storm` [2], `Waledac` [4], and even `Confiker`, which has been shown to embed P2P capabilities [3]. `Storm` and `Waledac` are of particular interest because they use P2P C&C structures as the primary way to organize their bots. While more complex, and perhaps more costly to manage compared to centralized botnets, P2P botnets offer higher resiliency against take-down efforts (e.g., by law enforcement), since even if a significant portion of bots in a P2P botnet are disrupted the remaining bots may still be able to communicate with each other and with the botmaster.

Detecting botnets is of great importance. However, designing an effective P2P-botnet detection system is faced with several challenges. First, the P2P file-sharing and communication applications, such as `Bittorrent`, `emule`, and `skype`, are very popular and hence C&C traffic of P2P botnets can easily blend into the background P2P traffic. This challenge is further compounded by the fact that a bot-compromised host may exhibit mixed patterns of both legitimate and botnet P2P traffic (e.g., due to the coexistence of a file-sharing P2P application and a P2P bot on the same host). Second, modern botnets tend to use increasingly stealthy ways to perform malicious activities that are extremely hard to be observed in the network traffic. For example, some botnets send spam through large popular webmail services such as `Hotmail` [6], which is likely transparent to network detectors due to encryption and overlap with legitimate email use patterns. Third, as the volume of network traffic grows rapidly, the deployed detection system is required to process a huge amount of information efficiently.

To date, a few approaches capable of detecting P2P botnets have been proposed [7]–[9]. However, these approaches cannot address all the aforementioned challenges. For example, BotMiner [7] identifies a group of hosts as bots belonging to the same botnet if they share similar communication patterns and meanwhile perform similar malicious activities, such as scanning, spamming, exploiting, etc. Unfortunately, the malicious activities may be stealthy and non-observable,

Manuscript received November 3, 2012; revised July 5, 2013 and October 24, 2013; accepted October 24, 2013. Date of publication November 11, 2013; date of current version December 16, 2013. This work was supported in part by the Research Initiation Grant at Wright State University under Grant 282040, in part by the GRF PolyU 5389/13E, NSF under Grant CNS-1149051 and Grant 0831300, in part by the Department of Homeland Security under Contract FA8750-08-2-0141, and in part by the Office of Naval Research under Grant N000140710907 and Grant N000140911042. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. C.-C. Jay Kuo.

J. Zhang is with the Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435 USA (e-mail: junjie.zhang@wright.edu).

R. Perdisci is with the Department of Computer Science, University of Georgia, Athens, GA 30602 USA (e-mail: perdisci@cs.uga.edu).

W. Lee is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: wenke@cc.gatech.edu).

X. Luo is with the College of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: csluo@comp.polyu.edu.hk).

U. Sarfraz was with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA. She is now with Cisco, San Jose, CA 95134 USA (email: usarfraz@cisco.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2013.2290197

thereby making BotMiner ineffective. In addition, BotMiner's scalability is significantly constrained [10]. Yen et al. [8] proposed an algorithm that aims to distinguish between hosts that run legitimate P2P file sharing applications and P2P bots. Nevertheless, this algorithm [8] does not take into account the fact that a bot may coexist with a legitimate P2P application on the same host. As a consequence, the mixed traffic profile of the compromised host may disguise the communication patterns related to the bot, rendering the algorithm [8] ineffective. BotGrep [9] analyzes network flows collected over multiple large networks (e.g., ISP networks), and attempts to detect P2P botnets by analyzing the communication graph formed by overlay networks. Although BotGrep does not rely on malicious activities for detection, it requires a global view of Internet traffic and *a priori* detection results from additional systems to bootstrap the detection. However, it is extremely hard to acquire such information in practice.

In this paper, we present a novel *scalable* botnet detection system capable of detecting *stealthy* P2P botnets. We refer to a stealthy P2P botnet as a P2P botnet whose malicious activities may not be observable in the network traffic. Particularly, our system aims to detect stealthy P2P botnet even if P2P botnet traffic is overlapped with traffic generated by legitimate P2P applications (e.g., Skype) running on the same compromised host and ii) achieve high scalability. To this end, our system identifies P2P bots within a monitored network by detecting the C&C communication patterns that characterize P2P botnets, regardless of how they perform malicious activities in response to the botmaster's commands. Specifically, it derives *statistical fingerprints* of the P2P communications generated by P2P hosts and leverages them to distinguish between hosts that are part of legitimate P2P networks (e.g., file-sharing networks) and P2P bots. The high scalability of our system stems from the parallelized computation with bounded computational complexity. To summarize, our work makes the following contributions:

1) A new *flow-clustering*-based analysis approach to identify hosts that engage in P2P communications.
2) An efficient algorithm for P2P traffic *profiling*, where we build statistical fingerprints to profile various P2P applications and estimate their *active time*.
3) A P2P botnet detection method that can effectively detect stealthy P2P bots even if the P2P botnet traffic is overlapped with traffic generated by legitimate P2P applications (e.g., Skype) running on the same compromised machine.
4) A scalable design based on an efficient detection algorithm and parallelized computation.
5) A prototype system and extensive evaluation based on real-world network traffic, which has demonstrated high detection accuracy (i.e., a detection rate of 100% and 0.2% false positive rate) and great scalability (i.e., processing 80 million flows in 0.8 hour) of our design.

Compared to our preliminary version of this work [11], we have made the following substantial improvements. First, we simplified the system design by eliminating a coarse-grained analysis component for P2P client detection based on failed network connections without sacrificing its detection performance. The new design eradicates the necessity of keeping failed connections, reducing the storage cost by 60%. Second, we redesigned the clustering-based P2P client detection algorithm to enhance its efficiency, which decreases the processing time by at least 68% compared to the original design. Third, we parallelize our system to boost its scalability and demonstrated its efficiency. Finally, we have empirically evaluated the extent to which configurable parameters affect the detection performance and manifested that our system is effective over a large range of parameter values.

II. Related Work

A few approaches capable of detecting P2P botnets have been proposed [7]–[9], [12]–[14]. Compared with the existing methods [7]–[9], the design goals of our approach are different in that: 1) our approach does not assume that malicious activities are observable, unlike [7]; 2) our approach does not require any botnet-specific information to make the detection, unlike [9]; 3) our approach needs to detect the compromised hosts that run both P2P bot and other legitimate P2P applications at the same time, unlike [8]; and 4) different from [7]–[9], our approach has high scalability as a built-in feature. Other methods [12]–[14] use machine learning for detection, which require labeled P2P botnet data to train a statistical classifier. Unfortunately, acquiring such information is a challenging task, thereby drastically limiting the practical use of these methods.

To achieve the aforementioned design goals, our system includes multiple components. The first one is a *flow-clustering*-based analysis approach to identify hosts that are mostly likely running P2P applications. In contrast to existing approaches of identifying hosts running P2P applications [15]–[19], our approach differs in the following ways: 1) unlike [16], our approach does not need any content signature because encryption will make content signature useless; 2) our approach does not rely on any transport layer heuristics (e.g., fixed source port) used by [15], [17], which can be easily violated by P2P applications; 3) we do not need training data set to build a machine learning based model as used in [18], because it is very challenging to get traffic of P2P botnets before they are detected; 4) in contrast to [19], our approach can detect and profile various P2P applications rather than identifying a specific P2P application (e.g., `Bittorrent`); and 5) our analysis approach can estimate the active time of a P2P application, which is critical for botnet detection.

III. System Design

System Overview: A P2P botnet relies on a P2P protocol to establish a C&C channel and communicate with the botmaster. Therefore P2P bots exhibit some network traffic patterns that are common to other P2P client applications (either legitimate or malicious). Thus, we divide our systems into two phases. In the first phase, we aim at detecting all hosts within the monitored network that engage in P2P communications. As shown in Figure 1, we analyze raw traffic collected at the edge of the monitored network and apply a pre-filtering step to discard network flows that are unlikely to be generated by P2P

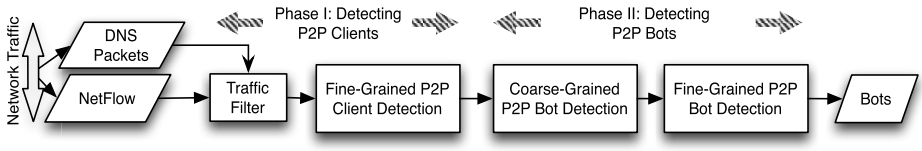


Fig. 1. System overview.

TABLE I
NOTATIONS AND DESCRIPTIONS

notation	Description
T_{p2p}	the active time of P2P application
No-DNS Peers	the percentage of flows associated with no domain names
N_{clust}	the number of clusters left by enforcing Θ_{bgp} and Θ_{p2p}
N_{bgp}	the largest number of unique bgp prefixes in one cluster
$\hat{T_{p2p}}$	the estimated active time for P2P application

TABLE II
MEASUREMENT OF FEATURES

Trace	T_{p2p}	*No-DNS Peers*	N_{clust}	N_{bgp}	$\hat{T_{p2p}}$
T-Bittorrent	24hr	96.85%	17	12857	24hr
T-Emule	24hr	99.99%	8	1133	24hr
T-Limewire	24hr	99.97%	36	5661	24hr
T-Skype	24hr	99.93%	12	12806	24hr
T-Ares	24hr	99.99%	16	1596	24hr

applications. We then analyze the remaining traffic and extract a number of statistical features to identify flows generated by P2P clients. In the second phase, our system analyzes the traffic generated by the P2P clients and classifies them into either *legitimate* P2P clients or P2P *bots*. Specifically, we investigate the active time of a P2P client and identify it as a *candidate* P2P bot if it is persistently active on the underlying host. We further analyze the overlap of peers contacted by two *candidate* P2P bots to finalize detection.

To illustrate the statistical features and motivate the related thresholds used by our system, we ran five popular P2P applications, including `Bittorrent`, `Emule`, `Limewire`, `Skype`, and `Ares`, for 24 hours to collect their traffic traces. For the Bittorrent application, we generated two separate 24-hour traces (`T-Bittorrent` and `T-Bittorrent-2`). In this section we report a number of measurements on the obtained traffic traces to better motivate the design of statistical features, whose notations are summarized in Table I. Table II reports the feature values measured on the collected traffic traces. We now elaborate on each component of our system.

A. Identifying P2P Clients

Traffic Filter The Traffic Filter component aims at filtering out network traffic that is unlikely to be related to P2P communications. This is accomplished by passively analyzing DNS traffic, and identifying network flows whose destination IP addresses were previously *resolved* in DNS responses. Specifically, we leverage the following feature: P2P clients usually contact their peers directly by looking up IPs from a routing table for the overlay network, rather than resolving a domain name. This feature is supported by Table II (*No-DNS Peers*), which illustrates that the vast majority of flows generated by P2P applications do not have destination IPs resolved from domain names. The remaining small fraction of flows are corresponding to a possible exception that a peer bootstraps into a P2P network by looking up domain names that resolve to stable *super-nodes*) Since most non-P2P applications (e.g., browsers, email clients, etc.) often connect to a destination address resulting from domain name resolution, this simple filter can eliminate a very large percentage of non-P2P traffic, while retaining the vast majority of P2P communications.

Fine-Grained Detection of P2P Clients: This component is responsible for detecting P2P clients by analyzing the remaining network flows after the Traffic Filter component. For each host h within the monitored network we identify two flow sets, denoted as $S_{tcp}(h)$ and $S_{udp}(h)$, which contain the flows related to successful outgoing TCP and UDP connection, respectively. We consider as successful those TCP connections with a completed `SYN, SYN/ACK, ACK` handshake, and those UDP (virtual) connections for which there was at least one "request" packet and a consequent response packet.

In order to detect P2P clients, we first consider the fact that each P2P client frequently exchanges control messages (e.g., ping/pong messages) with other peers. Besides, we notice that the characteristics of these messages, such as the size and frequency of the exchanged packets, are *similar* for nodes in the same P2P network, and *vary* depending on the P2P protocol and network in use. As a consequence, if two network flows are generated by the same P2P application and they carry the same type of P2P control messages, they tend to share similar flow size. In addition, a P2P client will exchange control messages with a large number of peers distributed in many different networks. Consequently, the destination IP addresses of network flows that carry these control messages will spread across a large number of networks where each network can be represented by its `BGP` prefix.

To identify flows corresponding to P2P control messages, we first apply a flow clustering process intended to group together similar flows for each candidate P2P node h. Given sets of flows $S_{tcp}(h)$ and $S_{udp}(h)$, we characterize each flow using a vector of statistical features $v(h) = [Pkt_s, Pkt_r, Byte_s, Byte_r]$, in which Pkt_s and Pkt_r represent the number of packets sent and received, and $Byte_s$ and $Byte_r$ represent the number of bytes sent and received, respectively. The distance between two flows is subsequently defined as the *euclidean distance* of their two corresponding vectors. We then apply a clustering algorithm to partition the set of flows into a number of clusters. Each of the obtained clusters of flows, $C_j(h)$, represents a group of flows with similar size. For each $C_j(h)$, we consider the set of destination IP addresses related to the flows in the clusters, and for each of these IPs we consider its `BGP` prefix (using `BGP` prefix announcements).

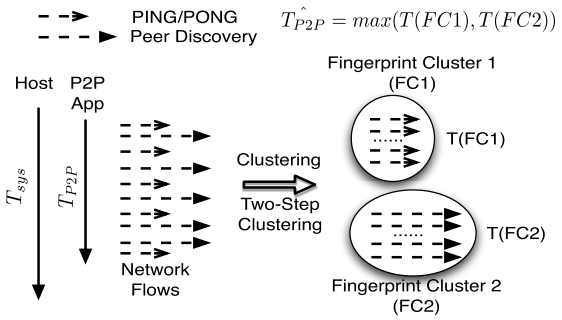


Fig. 2. Example of flow clustering to identify P2P hosts.

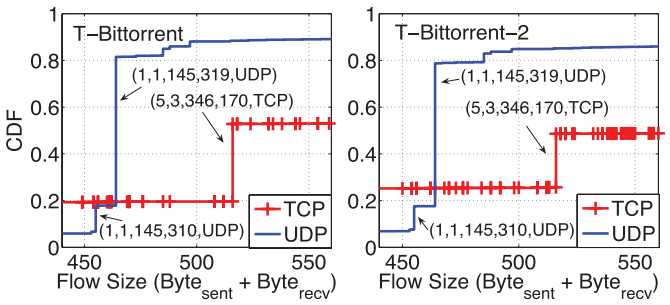


Fig. 3. CDF of flow size.

Finally, we count the number of distinct BGP prefixes related to destination IPs in a cluster $bgp_j = BGP(C_j(h))$, and discard those clusters of flows for which $bgp_j < \Theta_{bgp}$. We call *fingerprint clusters* the remaining cluster of flows. Therefore, each host h can now be described by a set of fingerprint clusters $FC(h) = \{FC_1, \ldots, FC_k\}$. We label h as P2P node if $FC(h) \neq \emptyset$, namely if h generated at least one fingerprint cluster.

Figure 2 illustrates an example of the flow clustering process for a P2P node. Flows corresponding to `ping/pong` and `peer-discovery` share similar sizes, and hence they are grouped into two clusters (FC_1 and FC_2), respectively. Since the number of destination BGP prefixes involved in each cluster is larger than Θ_{bgp}, we take FC_1 and FC_2 as its fingerprint clusters. A *fingerprint cluster summary*, ($\overline{Pkt_s}, \overline{Pkt_r}, \overline{Byte_s}, \overline{Byte_r}$, proto), represents the protocol and the average number of sent/received packets/bytes for all the flows in this fingerprint cluster. We implemented the flow analysis component (with $\Theta_{bgp} = 50$) and identified fingerprint cluster for the sample P2P traces including two `Bittorrent` traces (`T-Bittorrent`, `T-Bittorrent-2`), and one Skype trace. Figure 3 describes the distribution of flow sizes for these two `Bittorrent` traces and a large number of flows share similar sizes. The summaries of the fingerprint clusters are illustrated in Table III, which can successfully cluster flows with similar sizes as presented in Figure 3. Particularly, manual investigation of the flow payload has confirmed flows corresponding to two fingerprint clusters, "(1 1 145 319, UDP)" and "(1 1 109 100, UDP)", are used for node discovery and exchanging `ping/pong` messages, respectively.

TABLE III
SUMMARIES OF FINGERPRINT CLUSTERS

Trace	Fingerprints
T-Bittorrent	1 1 145 319, UDP 1 1 109 100, UDP 1 1 146 340, UDP 5 3 346 170, TCP 1 1 145 310, UDP
T-Bittorrent-2	1 1 145.01 317.66, UDP 1 1 109 100, UDP 1 1 146 342, UDP 5 3 346 170, TCP 2 2 466 461, UDP

B. Detecting P2P Bots

Coarse-Grained Detection of P2P Bots: Since bots are malicious programs used to perform profitable malicious activities, they represent valuable assets for the botmaster, who will intuitively try to maximize utilization of bots. This is particularly true for P2P bots because in order to have a functional overlay network (the botnet), a sufficient number of peers needs to be always online. In other words, the active time of a bot should be comparable with the active time of the underlying compromised system. If this was not the case, the botnet overlay network would risk *degenerating* into a number of disconnected subnetworks due to the short life time of each single node. In contrast, the active time of legitimate P2P applications is determined by users, which is likely to be transient. For example, some users tend to use their file-sharing P2P clients only to download a limited number of files before shutting down the P2P application [20]. In this case, the active time of the legitimate P2P application may be much shorter compared to the active time of the underlying system. It is worth noting that some users may run certain legitimate P2P applications for as long as their machine is on. For example, `Skype` is a popular P2P application for instant messaging and voice-over-IP (VoIP) that is often setup to start after system boot, and that keeps running until the system is turned off. Therefore, such `Skype` clients (or other "persistent" P2P clients) will not be filtered out at this stage.

Hence, the first component in the "Phase II" of our system ("*Coarse-Grained Detection of P2P Bots*") aims at identifying P2P clients that are active for a time T_{P2P} close to the active time T_{sys} of the underlying system they are running on. While this behavior *is not unique* to P2P bots and may be representative of other P2P applications (e.g., Skype clients that run for as long as a machine is on), identifying persistent P2P clients *takes us one step closer* to identifying P2P bots.

To estimate T_{sys} we proceed as follows. For each host $h \in \mathbf{H}$ that we identified as P2P clients according to Section IV-B, we consider the timestamp $t_{start}(h)$ of the first network flow we observed from h and the timestamp $t_{end}(h)$ related to the last flow we have seen from h. Afterwards, we divide the time $t_{end}(h) - t_{start}(h)$ into w epochs (e.g., of one hour each), denoted as $T = [t_1, \ldots t_i, \ldots, t_w]$. We further compute a vector $A(h, T) = [a_1, \ldots a_i, \ldots, a_w]$ where a_i is equal to 1 if h generated any network traffic between t_{i-1} and t_i. We then estimate the active time of h as $T_{sys} = \sum_{i=1}^{w} a_i$.

In order to estimate the active time of a P2P application, we can leverage obtained fingerprint clusters. It is because that a P2P application periodically exchanges network control (e.g., ping/pong) messages with other peers as long as

the P2P application is active. For each host h (again, we consider only the hosts in **H**, which we previously identified as P2P clients), we examine the set of its fingerprint clusters $FC(h) = \{FC_1, \ldots FC_j \ldots, FC_k\}$ (see Section III). Based on the flows belonging to a fingerprint cluster FC_j, we use the same approach of computing T_{sys} to calculate its active time, denoted as $T(FC_j)$. Then, we estimate the active time (T_{P2P}) of a P2P application as $\hat{T_{P2P}} = \max(T(FC_1), \ldots T(FC_j), \ldots T(FC_k))$.

If the ratio $r(h) = \frac{\hat{T_{P2P}}}{T_{sys}} > \Theta_{P2P}$, we say that h is running a persistent $P2P$ application, and add it to a set **P** of *candidate* P2P bots. Host h will then be input to next step, where h will be represented by a set of persistent fingerprint clusters for h, denoted as $FC_p(h) = \{FC_i^1, \ldots, FC_k^j\}$ where $T(FC_i)/T_{sys} > \Theta_{P2P}$ for any $FC_i \in FC_p(h)$.

As illustrated in Table II, the estimated active time $\hat{T_{P2P}}$ is the same as the actual active time (T_{P2P}) for each P2P application, which demonstrates that $\hat{T_{P2P}}$ can accurately approximate T_{P2P}. As we can see from Table II, when we leave a P2P application running for as long as the machine is on (24 hours for this particular experiment), we obtain a ratio $r(h) = 1$. Therefore, we decided to conservatively set $\Theta_{P2P} = 0.5$. N_{clust} in Table II illustrates the size of $FC_p(h)$, the number of fingerprint clusters (FCs) whose $BGP(FC) > \Theta_{bgp}$ and $T(FC) > \Theta_{p2p}$.

Fine-Grained Detection of P2P Bots: The objective of this component is to identify P2P bots from all persistent P2P clients (i.e., set **P**). We leverage one feature: the overlap of peers contacted by two P2P bots belonging to the same P2P botnet is much larger than that contacted by two clients in the same legitimate P2P network. Assume that two hosts in the monitored network, say h_A and h_B, are running the same legitimate P2P file-sharing application (e.g., Emule). Users of these two P2P clients will most likely have uncorrelated usage patterns. It is reasonable to assume that in the general case the two users will search for and download different content (e.g., different media files or documents) from the P2P network. This translates into a *divergence* between the set of IP addresses contacted by hosts h_A and h_B. The reason is that the two P2P clients will tend to exchange P2P control messages (e.g., ping/pong and search requests) with different sets of peers which "own" the content requested by their users, or peers that are *along the path* towards the content. On the contrary, if h_A and h_B are compromised with P2P bots, one common characteristic of bots is that they need to periodically *search for commands* published by the botmaster [21]. This typically translates into a *convergence* between the set of IPs contacted by h_A and h_B.

In order to leverage this feature, we represent each host $h \in \mathbf{P}$ using its persistent fingerprint clusters $FC_p(h) = \{FC_1, \ldots, FC_k\}$. Each fingerprint cluster FC_i is converted to a vector $[\overline{Byte}_{s,i}, \overline{Byte}_{r,i}, \Pi_i]$. In this vector, $\overline{Byte}_{s,i}$ ($\overline{Byte}_{s,i}$) is the average number of bytes sent (received) per flow in FC_i. Π_i is a set that contains the destination IP addresses (peers) of the flows in FC_i.

We further define two distance functions below, where $FC_i^{(a)}$ and $FC_j^{(b)}$ represent two fingerprint clusters from two persistent P2P clients, h_a and h_b, respectively.

- $d_{bytes}(FC_i^{(a)}, FC_j^{(b)}) = \sqrt{(\overline{Byte}_{s,i}^{(a)} - \overline{Byte}_{s,j}^{(b)})^2 + (\overline{Byte}_{r,i}^{(a)} - \overline{Byte}_{r,j}^{(b)})^2}$
- $d_{IPs}(FC_i^{(a)}, FC_j^{(b)}) = 1 - \frac{|\Pi_i^a \cap \Pi_j^b|}{|\Pi_i^a \cup \Pi_j^b|}$

If two P2P clients (say h_a and h_b) belong to the same P2P network, regardless of a legitimate P2P network or a P2P botnet network, these two clients will follow the same implementation of the identical P2P protocol. Hence, the network flows corresponding to the same type of P2P control messages (e.g., ping/pong messages) will exhibit similar flow sizes across P2P clients running the same P2P application. Since a fingerprint cluster summarizes network flows for the same type of control messages in one client, two fingerprint clusters corresponding to the same P2P control messages belonging to the same P2P application will have similar flow size. In other words, two P2P clients from the same P2P network will share at least one pair of fingerprint clusters $(FC_i^{(a)}, FC_j^{(b)})$, which have a small value of $d_{bytes}(FC_i^{(a)}, FC_j^{(b)})$ since they are corresponding to the same P2P control message. Otherwise, if two P2P clients belong to different P2P networks, d_{bytes} tends to be large.

Given two P2P bots (say h_a and h_b) belonging to the same botnet, the sets of peers contacted by these two bots, denoted as Π_i^a and Π_j^b, will share a large overlap, thereby generating a small value of $d_{IPs}(FC_i^{(a)}, FC_j^{(b)})$. Otherwise, if two P2P clients belong to i) the same legitimate P2P network or ii) different P2P networks, they will share a small overlap and produce a large value of $d_{IPs}(FC_i^{(a)}, FC_j^{(b)})$.

We further define a distance function $dist(h_a, h_b)$ to quantify the similarity of two P2P clients by integrating d_{bytes} and d_{IPs}. $dist(h_a, h_b)$ tends to yield a small value if h_a and h_b are infected with bots from the same P2P botnet. Especially, even if h_a and h_b are infected with P2P bots from the same botnet and they run legitimate P2P applications simultaneously, the distance quantified by $dist(h_a, h_b)$ will be small. It is because that at least one pair of fingerprint clusters that are generated by P2P bots will yield small values for for both d_{bytes} and d_{IPs}.

$$dist(h_a, h_b) = \min_{i,j} \Big(\lambda * \frac{d_{bytes}(FC_i^{(a)}, FC_j^{(b)}) - min_B}{max_B - min_B} + (1 - \lambda) * d_{IPs}(FC_i^{(a)}, FC_j^{(b)}) \Big)$$

where

- $FC_k^{(x)}$ is the k-th fingerprint cluster of host h_x
- $min_B = \min_{i,j} d_{bytes}(FC_i^{(a)}, FC_j^{(b)})$
- $max_B = \max_{i,j} d_{bytes}(FC_i^{(a)}, FC_j^{(b)})$
- λ is a predefined constants, which we set to $\lambda = 0.5$.

After computing the distance between each pair of hosts (i.e., hosts in set **P**), we apply hierarchical clustering, and group together hosts according to the distance defined above. In practice the hierarchical clustering algorithm will produce a dendrogram (a tree-like data structure). The dendrogram expresses the "relationship" between hosts. The closer two

hosts are, the lower they are connected at in the dendrogram. Two P2P bots in the same botnet should have small distance and thus are connected at lower level (forming a *dense* cluster). In contrast, legitimate P2P applications tend to have large distances and consequently are connected at the upper level. We then classify hosts in *dense* clusters as P2P bots, and discard all other clusters and the related hosts, which we classify as legitimate P2P clients. In practice, we cut the dendrogram at Θ_{bot} ($\Theta_{bot} \in [0, 1]$) of the maximum dendrogram height ($\Theta_{bot} * height_{max}$). To set Θ_{bot}, we assume that: a) we do not have a labeled data set of botnet traffic; b) the distance between two legitimate P2P applications is much larger than that between two bots belonging to the same botnet (as motivated above). Therefore, we conservatively set $\Theta_{bot} = 0.95$.

IV. System Implementation

The implementation objective is to integrate high scalability as a built-in feature into our system. To this end, we first identify the performance bottleneck of our system and then mitigate it using *complexity reduction* and *parallelization*.

A. Performance Bottleneck

Out of four components in our system, "*Traffic Filter*" and "*Coarse-Grained Detection of P2P Bots*" have linear complexity since they need to scan flows only once to identify flows with destination addresses resolved from DNS queries or calculate the active time. Other two components, "*Fine-Grained Detection of P2P Clients*" and "*Fine-Grained P2P Detection of P2P Bots*", require pairwise comparison for distance calculation. Specifically, if we denote the number of flows generated by a host as n and the number of hosts as S, the time complexity of *Fine-Grained Detection of P2P Clients* approximates $O(S*n^2)$. Comparably, if we denote the number of persistent P2P clients as l, the time complexity of *Fine-Grained P2P Bot Detection* approximates $O(l^2)$. Since the number of flows generated by network applications (i.e., n) could be enormous (e.g., more than hundreds of thousands of flows are generated by a single P2P client in our experiments), the computation overhead of *Fine-Grained Detection of P2P Clients* may become prohibitive. On contrary, the percentage of P2P clients in the ISP network is relatively small (e.g., 3%-13% as reported in [22]). Consequently, *Fine-Grained P2P Bot Detection* is unlikely to introduce huge performance overhead. For instance, given a typical ISP network or a large enterprise network that has 65,536 hosts (/16 subnet), if we assume that 8% hosts run P2P applications and conservatively assume that half of them are persistent, the number of persistent P2P clients (i.e., l) subject to analysis by *Fine-Grained P2P Bot Detection* is 2,221, incurring negligible overhead. To summarize, "*Fine-Grained P2P Client Detection*" is the performance bottleneck.

B. Two-Step Flow Clustering

We use a two-step clustering approach to reduce the time complexity of "*Fine-Grained P2P Client Detection*". For the first-step clustering, we use an efficient clustering algorithm to aggregate network flows into K sub-clusters, and each sub-cluster contains flows that are very similar to each other. For the second-step clustering, we investigate the global distribution of sub-clusters and further group similar sub-clusters into clusters.

The distance of two flows is defined as the Euclidean distance of their corresponding vectors, where each vector $[Pkt_s, Pkt_r, Byte_s, Byte_r]$ represents the number of packets/bytes that are sent/received in a flow. In our original design [11] , we have adopted BIRCH [23], a streaming clustering algorithm. The number of clusters generated by BIRCH is mainly decided by a predefined parameter R, which quantifies the radius of a cluster. A greater value of R implies less clusters. Although BIRCH can perform approximate clustering of an arbitrarily large dataset given constrained memory space by scanning the dataset only once, estimating K from R remains a challenging task. To partially address this challenge in our original design, we adopted an empirical way: we start from a small R value (e.g., $R = 0$) and gradually increase it by δ until K clusters are generated. Since the number of clusters generated by BIRCH is sensitive to R, δ has to be very small to assure that R is not overlarge. As a result, a huge number of iterations have to be explored until we find appropriate R that yields K sub-clusters. This procedure results in a large amount of computation time.

In the current design, we employ K-means as the first-step clustering. The main reason is that K-Means can achieve bounded time complexity $O(nKI)$, where K explicitly indicates the number of expected clusters, n is the number of flows for each host, and I is the maximum number of iterations.

For the second-step clustering, we use *hierarchical clustering* with *DaviesBouldin* validation [24] to group sub-clusters into clusters. Each sub-cluster is represented using a vector $([\overline{Pkt_s}, \overline{Pkt_r}, \overline{Byte_s}, \overline{Byte_r}])$, which is essentially the average for all flow vectors in this sub-cluster. Hierarchical clustering is used to build a dendrogram. Finally, *DaviesBouldin* validation is employed to assess the global distribution of inter- and intra-cluster distances of clusters based on various clustering decisions and yield the best cut for the dendrogram. The two-step clustering algorithm has the time complexity of $O(nKI + K^2)$.

C. System Parallelization

Since the two-step clustering analyzes network flows for each single host, we can parallelize the computation for all hosts. We formulate the problem as follows: given S hosts denoted as $H = \{h_1, h_2, \ldots h_S\}$ and M computation nodes denoted as $C = \{c_1, c_2, \ldots c_M\}$, we partition H into M exclusive subsets $HT_1, HT_2..HT_M$ and assign HT_i to c_i for analysis, whose processing time is denoted as $exc(c_i, HT_i)$. Our target is to design a partition algorithm so that the overall processing time, denoted as $T = max(exc(c_i, HT_i))$, is minimized. If we assume each computation node has the same capacity, T will be minimized when the analysis workload is evenly distributed across all computation nodes.

TABLE IV
TRAFFIC STATISTICS FOR OUR ACADEMIC NETWORK

Trace	duration	# of TCP / UDP flows	# of clients
t-c	24h	61,745,989 / 20,226,837	953
Trace	duration	# of domains	# of IPs
t-dns	24h	328,965	268,753

To this end, we need a function that estimates the time consumption based on the number of flows generated by a host. We denote this function as $g(f_i)$, where f_i is the number of flows initiated by an host h_i (i.e., $f_i = |S_{tcp}(h_i)| + |S_{udp}(h_i)|$). An empirical approach is used to derive $g(f_i)$. We first randomly sample a small number of hosts from H, say P (e.g., $P = 10$), which have $f_i^1, f_j^2, \ldots, f_n^P$ flows respectively. We then apply the two-step clustering analysis to each sampled host and obtain their execution time, $t_1, t_2, \ldots, t_P$. Given $f_i^1, f_j^2, \ldots, f_n^P$ and $t_1, t_2, \ldots, t_P$, a function $g()$ can be finally learnt using the regression model. Since the time complexity of the two-step clustering algorithm is $O(nKI + K^2)$, the linear regression model is adopted. After getting $g(f_i)$, we introduce a cumulative function $G(f_i) = \sum_{w=1}^{w=i} g(h_w)$. We assign h_i to the dth $(d = 1, 2 \ldots M)$ computation node if $(d-1) * \frac{G(f_N)}{M} < G(f_i) \leq d * \frac{G(f_N)}{M}$ so that each node is given approximately the same workload.

By following these two approaches, we successfully eliminate the performance bottleneck by reducing the computational complexity from $O(S * n^2)$ to $O(\frac{S}{M} * (nKI + K^2))$.

V. EVALUATION

A. Network Traces

The traffic we collected from an academic network came from a span port mirroring all traffic crossing the gateway router (around 200-300Mbps) for the college networks. We used `Argus` [25] to efficiently collect network flow information of the traffic between internal and external networks for one entire day. Along with various flow statistics we also recorded the first 200 bytes of each *flow payload*, which we used to identify known legitimate P2P clients within our network. The DNS traffic was collected simultaneously with the network flow information, using `dnscap`, to keep track of all the domain-to-IP mappings needed to perform traffic volume reduction. Overall, we observed 953 active hosts, as reported in Table IV. We refer to the traffic collected from our academic network as NET_{CoC}. Different from our original design in [11], our new design ignores flows of failed connections, which account for 60% in NET_{CoC}, thereby reducing the storage cost by 60%.

In order to establish ground truth in terms of what hosts are running P2P applications, we used a signature-based approach that matches the signatures from [26] onto the first 200 bytes of each network flow. After manual validation, we identified a total of 3 hosts that were running `Bittorrent`, which we denoted as "BT1@C", "BT2@C" and "BT3@C". Furthermore, there exists no signature that can match P2P traffic generated by `Skype`, since `Skype` communications are encrypted. However, using the statistical traffic fingerprints, we were able to identify 5 likely `Skype` clients within our network (the details are discussed in [11]), denoted as "Skype1@C", "Skype2@C", ..., "Skype5@C". We refer to the network traces corresponding to these 8 P2P clients as $NET_{P2P@CoC}$.

TABLE V
TRACES OF POPULAR P2P APPLICATIONS

Trace	Dur	# of flows	# of Dst IPs	Avg Flow Size
Bittorrent-1/2	24 hr	250960/297785	17337/17657	68310/350205
Limewire-1/2	24 hr	229215/638103	11602/64994	1003/2038
Emule-1/2	24 hr	58941/110821	6649/14554	124267/22681
Skype-1/2	24 hr	88927/49541	10699/6264	514/1988
Ares-1/2	5 hr	17566/21756	1918/3118	69373/24755

TABLE VI
TRACES OF BOTNETS

Trace	duration	size	# of bots
Waledac	24hr	1.1G	3
Storm	24hr	4.8G	13
Zeus	24hr	7.3M	1

In order to increase the number and diversity of P2P nodes in our network, we ran 5 popular P2P applications, including `Bittorrent`, `Emule`, `Limewire`, `Skype`, and `Ares`. We ran each of the 5 P2P applications in two different (virtual) hosts for several hours (e.g., 24 or 5 hours) *simultaneously*. Each host was represented by a WindowsXP (virtual) machine with a public IP address selected within a /24 network. Given a P2P application among the 5 we considered, we manually interacted with one instance (on one host) to simulate typical human-driven application usage behavior, and we fed the second instance of the application (on the second host) with automatically generated user-interface input. This artificial user input was simulated using an `AutoIt` [27] script that randomly selects contents to be downloaded or uploaded using the P2P application at random time intervals. Therefore, overall we obtained 10 additional network traces related to traffic generated by P2P applications (Table V shows some statistics related to these network traces). We refer to these network traces as NET_{P2P}.

We were able to obtain network traces for two popular P2P botnets, `Storm` and `Waledac`, by purposely running their malware samples in a controlled environment and recording their network behavior. We refer to these network traces as NET_{bots}. The `Storm` traces included 13 different bot-compromised hosts and the `Waledac` included 3 different bot-compromised hosts, as shown in the first two rows in Table VI. In addition, we obtained network traces belonging to the `Zeus` botnet (denoted as NET_{Zeus}), which is a relatively new P2P botnet since it reportedly integrated P2P C&Cs from Oct 2011 [28]. Unfortunately, since NET_{Zeus} contain only one `Zeus` bot, it is insufficient to evaluate the detection performance of our system. Nevertheless, it can still be used to evaluate our system's capability on profiling new P2P bots.

B. Experiment Setup

We prepared a data set (D) for evaluation. Specifically, we randomly selected half (8) of the P2P bots from NET_{bots}. Then for each of the 5 P2P applications we ran, we randomly selected one out of its two traces from NET_{P2P} and overlaid its traffic to the traffic of a randomly selected host from NET_{CoC}. We further randomly chose 3 P2P hosts from

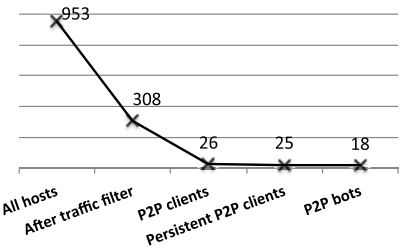


Fig. 4. Number of hosts identified by each component.

$NET_{P2P@CoC}$. We finally overlaid each of 8 P2P bot traces to each of the selected 8 P2P traces (5 from NET_{P2P} and 3 from $NET_{P2P@CoC}$). D represents the scenario that a host, which is compromised by a P2P bot, has an active legitimate P2P application running simultaneously. To summarize, D is composed of 26 P2P clients including i) 16 P2P bots, out of which 8 bots are overlaid with legitimate P2P traffic, ii) 5 legitimate P2P clients from NET_{CoC}, iii) 5 legitimate P2P clients from NET_{P2P}. In addition, we randomly selected one host from NET_{CoC} and mixed its traffic with NET_{Zeus} to get a data set NET'_{Zeus}.

For the following experiments including Section V-C.1, V-C.2, and V-C.3, we use $K = 4,000$, $\Theta_{BGP} = 50$, and $\Theta_{bot} = 0.95$. The number of hosts preserved by each component for these three experiments are presented in Figure 4. The evaluation of different parameter settings is discussed in Section V-C.1.

C. Experimental Results on Botnet Detection

1) DNS-Based Traffic Filter: We applied our detection system on data set D. The traffic filter drastically reduced the workload for the whole system. As indicated in Figure 4, it reduced the number of hosts subject to analysis by 67% (from 953 to 316) but retained all P2P clients.

2) Identifying and Profiling P2P Applications: Based on the traffic after the DNS-based filter, our system discovers fingerprint clusters. The discovered clusters can effectively not only identify P2P clients, but also profile P2P applications running in a host. Our system can successfully identify all 26 P2P clients in D. Examples of fingerprint cluster summaries ($[\overline{Pkt_s}, \overline{Pkt_r}, \overline{Byte_s}, \overline{Byte_r}, Proto]$) are presented in Table VII and Table VIII.

Table VII presents fingerprint cluster summaries for two `Storm` bots and two `Waledac` bots before their traffic was overlaid with legitimate P2P client traffic. In this table, we can find that bots from the same botnet share similar fingerprint clusters with respect to the flow size.

In contrast to Table VII, Table VIII presents several fingerprint clusters summaries for two bots (`Waledac2+BT2@C` and `Storm4+Skype4@C`) whose traffic is overlaid with traffic from legitimate P2P applications. The results in this table demonstrated that our system can effectively isolate fingerprint clusters belonging to different P2P applications even if these P2P applications are running in the same host. For example, `Waledac2+BT2@C`, the fingerprint clusters come from two P2P applications, where "(1 1 145 139, UDP)" and "(1 1 75 75, UDP)" are from `Bittorrent` (more fingerprint clusters for `Bittorrent` can be found in Table XI and XII in [11]), and "(4 3 224 170, TCP)" together with "(3 3 185 162, TCP)" are from `Waledac` (referring to Table VII).

TABLE VII
FINGERPRINT CLUSTER SUMMARIES FOR P2P BOTS

Trace	Fingerprints	Trace	Fingerprints
Storm1	2 2 94 554, UDP 2 2 94 1014, UDP 2 2 94 278, UDP ...	Storm2	2 2 94 554, UDP 2 2 94 1014, UDP 2 2 94 278, UDP ...
Waledac1	4 3 224 170, TCP 3 3 186 162, TCP 5 4 286 224, TCP ...	Waledac2	4 3 224 170, TCP 3 3 186 162, TCP 5 4 285 224, TCP ...

TABLE VIII
FINGERPRINTS FOR `Storm` AND `Waledac`

Trace	Fingerprints
Waledac2+BT2@C	1 1 145 319, UDP (Bittorrent) 4 3 224 170, TCP (Waledac) 3 3 185 162, TCP (Waledac) 1 1 75 75, UDP (Bittorrent) ...
Storm4+Skype4@C	2 2 94 554, UDP (Storm) 2 2 94 1014, UDP (Storm) 1 1 73 60, UDP (Skype) ...

TABLE IX
FINGERPRINTS FOR `Zeus`

Trace	Fingerprints
Zeus	2 2 468 709, UDP 2 2 344 817, UDP 2 2 339 699, UDP 3 3 543 988, UDP 2.5 2.5 520 827, UDP 3.2 3.2 742 1039, UDP

We used NET'_{Zeus} to evaluate whether our system can effectively profile a `Zeus` bot. Our system generated 6 statistical fingerprint clusters, whose summaries are illustrated in Table IX. Obtaining fingerprint clusters from this host demonstrates that our system can successfully identify a P2P application running on the host. Although a single `Zeus` bot is insufficient for the final component to perform detection, its fingerprint clusters offer valuable information: the identified P2P application (`Zeus`) adopts different P2P protocols compared to known P2P applications (5 legitimate P2P applications and 2 P2P botnets) since its fingerprint clusters have not been observed in our experiments.

3) Detecting P2P Bots: Among 26 P2P clients identified in the previous step, 25 out of them exhibit persistent P2P behaviors. We further evaluate the similarity of fingerprint clusters and peer IPs for each pair of persistent P2P clients and derive a dendrogram as shown in the Figure 5. As bots from the same botnet share similar fingerprint clusters and a large overlap of peer IPs, they will have small distance ($dist()$), resulting in a dense cluster of these bots in the dendrogram. As indicated in Figure 5, all 13 `Storm` bots and 3 `Waledac` bots form two dense clusters, respectively. We conservatively cut the tree at $\Theta_{bot} * height_{max} = 0.95$ ($\Theta_{bot} = 0.95$ and $height_{max} = 1$). Hence, three clusters are identified and overall 18 hosts were labeled as suspicious, which include *all* 16 P2P bots and 2 false positives, resulting a high detection rate of 100% and a low false positive rate of 0.2% (2/953).

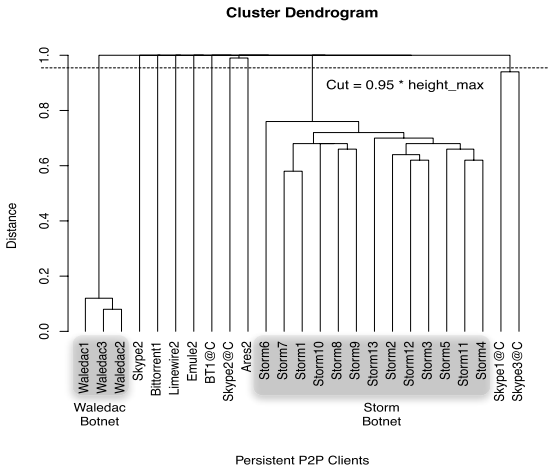


Fig. 5. Hierarchical tree on persistent P2P hosts.

The false positives appear to be two `Skype` clients. The reason for these two false positives is the conservatively configured value of Θ_{bot}, which is close to 1.

It is worth noting that the legitimate P2P application running on a bot-compromised host may present a significant challenge for the existing detection method such as [8]. It is mainly due to the fact that the traffic profile of a bot-compromised host might be completely distorted by the legitimate P2P application running on it simultaneously. For instance, in our experiments, when a host is running a `Waledac` and a `Bitorrent` application simultaneously, the traffic profile of the *whole host* is dominated by the `Bittorrent` application, concealing the traffic characteristics of the `Waledac`: the average flow size of the entire host is 348708 whereas the average flow size of the `Waledac` bot is only 11372; the number of peer IP addresses contacted by the entire host is 1359 compared to only 760 contacted by the `Waledac` bot. Since our detection algorithm is rooted in fingerprint clusters, which can discern different P2P applications, it is capable of detecting P2P bots (malicious P2P applications) even if the bot-compromised hosts are running legitimate P2P applications.

4) Analyzing the System Scalability: After optimization, the performance bottleneck of our system will have a low complexity of $O(\frac{S}{M} * (nKI + K^2))$, where K indicates the number of sub-clusters generated by K-Means and M represents the number of computation nodes (e.g., processors in a MapReduce infrastructure). As K decreases, the second-step clustering, hierarchical clustering, processes fewer sub-clusters and requires less time. The computation capacity grows as M increases. We set $I = 10$ but the K-Means clustering usually converged before 10 iterations in our experiments.

Figure 6a shows system performance as K decreases, where the processing time has been reduced by 93% (from 12 hours to 0.8 hours). In addition, Figure 6a demonstrates that our current design (i.e., K-Means with hierarchical clustering) outperforms the original design (i.e., $Birch$ with hierarchical clustering). For example, when $K = 2,000$, our current design used 0.3 hour while the original design [11] consumed 2.5 hours, reducing the processing time by 68%.

We parallelized the computation using M hosts ($1 \leq M \leq 10$ given $K = 4,000$). For each value of M, we repeated the experiments 5 times and report the average running time in Figure 6b. The experimental results demonstrated that the processing time is reduced from 1.4 hours to 0.4 hours given $M = 5$ and $K = 4,000$. Our load-balance partition method also surpasses the random partition method as indicated in Figure 6b. All of these experiments achieve the same detection accuracy with 100% detection rate and 0.2% false positive rate (given $\Theta_{bot} = 0.95$).

TABLE X
DETECTION RATE AND FALSE POSITIVE RATE FOR DIFFERENT Θ_{bot} AND K

		Θ_{bot}						
K	-	0.1	0.3	0.5	0.7	0.8	0.9	0.95
2000	DR	0	0	2/16	3/16	16/16	16/16	16/16
	FP	0	0	0	0	0	0	2/953
4000, 8000, and 10000	DR	2/16	3/16	3/16	16/16	16/16	16/16	16/16
	FP	0	0	0	0	0	0	2/953

When processing a huge amount of data, memory consumption is a concern. Since our system is designed to linearly analyze each host, the memory consumption is commensurate with the number of flows associated with each *single* host. Figure 6c presents the memory consumption of our system given $K = 4,000$ and $M = 1$, where the maximum consumption is around 1,800M. Therefore, a single computer is sufficient enough to operate our system.

5) Analyzing the Effect of System Parameters: While the measurement in Section III motivates the parameter values for Θ_{p2p}, the extent to which the parameters including K, Θ_{BGP}, and Θ_{bot} affect the detection results remains unknown. In this section, we empirically answer this question.

Although the number of sub-clusters for hierarchical clustering algorithm drops as K decreases, a small K may force dissimilar flows to be aggregated into one sub-cluster, thereby diminishing the similarity between two fingerprint clusters from two bots. To quantify the influence, we evaluate the detection performance when both K and Θ_{bot} are selected from large ranges (i.e., $K = 2000, 4000, 8000$ and 10000, and $\Theta_{bot} = 0.1, 0.3 \ldots 0.95$). The detection rates (DR) and false positive (FP) rates, which are reported in Table X, indicate that the detection performance is stable over a large range of K (e.g., ≥ 4000), and $\Theta_{bot} \in [0.7, 0.95]$ is a good candidate value. Table X also suggests that 0.8 or 0.9 may be a better value for Θ_{bot} compared to 0.95 used in our experiment. This implies that when a labeled data set of P2P botnet traffic is available we can tune this threshold (Θ_{bot}) to find a better trade-off between false positives and false negatives.

Θ_{BGP} is used to filter out those clusters that are unlikely to contain P2P control flows. If Θ_{BGP} is set too small, non-P2P hosts are likely to be identified as P2P hosts, resulting in false positives. Likewise, if Θ_{BGP} is too large, P2P bots may be discarded and false negatives would then be introduced. Table XI presents the number of detected bots and false positives as Θ_{BGP} is increased from 2 to 1,000. When $30 \leq \Theta_{BGP} \leq 200$, our system yields a small number of false positives and no false negative, demonstrating its effectiveness over a large range of settings.

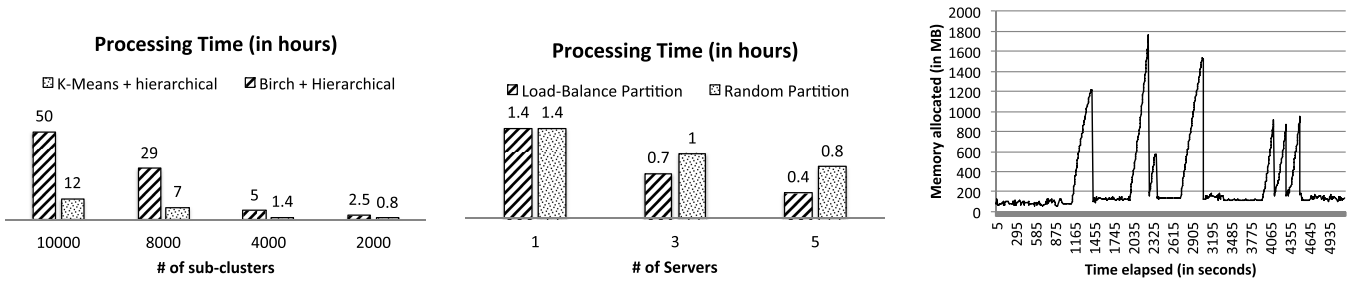


Fig. 6. System performance. (a) Processing time with different K. (b) Processing time with parallelization. (c) Memory consumption.

TABLE XI
FALSE POSITIVES (FPS) AND FALSE NEGATIVES (FNS) GIVEN DIFFERENT Θ_{BGP}

Θ_{BGP}	FPs	FNs	Θ_{BGP}	FPs	FNs
2	20	0	80 - 200	0	0
10	13	0	500	0	3
30	3	0	800	0	7
40 - 70	2	0	$\geq$ 1000	0	16

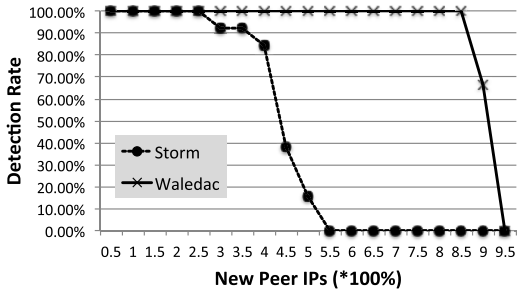


Fig. 7. Challenges for attackers to instruct bots to contact different peers to evade our detection.

D. Robustness Against Evasion By Mimicking Legitimate P2P Applications

Since our system aims at differentiating P2P bots from legitimate P2P applications, botmasters may instruct their bots to mimic legitimate P2P protocols in order to evade the detection. The most significant advantage for this evasion technique is that it will not introduce any unknown fingerprint clusters that may arouse anomaly. To be specific, botmasters can follow two steps. First, they could modify bots so that their network flows, which carry control messages, have similar flow sizes compared to those belonging to legitimate P2P applications. Further, they can instruct each bot to exchange control messages with more peers that are unlikely to be contacted by other bots, which will result in a small overlap of peers contacted by two bots. In order to evaluate the robustness of our system against such evasion technique, we perform the following experiment by simulating these two steps.

We first sort all fingerprint clusters of a P2P application/bot in the ascending order of its average flow size (i.e., $\overline{Byte_s} + \overline{Byte_r}$). We then select a legitimate P2P application as a target P2P application. For each bot, we alter the average flow size (i.e., $\overline{Byte_s}$ and $\overline{Byte_r}$) for each of its fingerprint cluster to the flow size of its corresponding fingerprint cluster belonging to the target P2P application. For example, if `T-Bittorrent` in Table III is selected as the target application, the average flow size ($\overline{Byte_s}$ and $\overline{Byte_r}$) of the first ("smallest") fingerprint clusters of the `Storm1` bot, (2 2 94 278, UDP) (see Table VII), will be changed to that of `T-Bittorrent`'s first ("smallest") fingerprint cluster, (1 1 109 100, UDP) (see Table III). If a bot has more fingerprint clusters than the target P2P application, we discard the those redundant ones.

Second, for each fingerprint cluster ($\overline{Byte_{s,i}}$, $\overline{Byte_{r,i}}$, Π_i) that belongs to a bot, we randomly generate $\gamma * |\Pi_i|$ peers and add them into Π_i, where Π_i contains distinct peers contacted by this bot in this particular fingerprint cluster. This procedure simulates that a bot contacts γ more random peers from the botnet in order to circumvent the detection.

We use all 5 legitimate P2P applications as target applications. For each target application, we increase γ from 0.5 to 9.5. Our system achieves a false positive rate of 0.2% for these experiments. Given a certain value of γ, Figure 7 presents the lowest detection rate among all the target applications, where the detection rate drops when γ increases. In order to completely bypass our system, `Storm` and `Waledac` need to contact 5.5 and 9.5 times more peers, respectively. Considering the fact that a `Storm` bot and a `Waledac` bot contacted 7,073 and 4,065 peers in average, they need to communicate with total 45,975 and 42,682 peers for successful evasion, respectively. Consequently, bots will be more "noisy" and thus more likely to arouse suspicion. In addition, once a bot is detected, it will reveal the presence of much more malicious peers, significantly facilitating the disruption efforts.

VI. POSSIBLE EVASIONS AND SOLUTIONS

If botmasters get to know about our detection algorithm, they could attempt to modify other bots' network behavior to evade detection. This situation is similar to evasion attacks against other intrusion detection systems. In this section, we discuss the possible evasions and solutions in addition to the "mimicking attacks" presented in Section V-D.

A. Evasion by Misusing the Traffic Filter

Botmasters may exploit our traffic filter for evasion. For example, a botmaster may set up a DNS server and then instruct each bot to query this server before contacting any peer. The (malicious) DNS server can respond the bot with a DNS response that contains the IP address of that peer.

In this case, our traffic filter could eliminate bot flows. To solve this problem, we can only filter out network flows whose destination IP addresses are resolved from popular domains, i.e., domains queried by a non-negligible fraction of hosts in the monitored networks.

B. Evasion by Joining Legitimate P2P Applications

Bots may "blend" with legitimate P2P clients to evade detection by directly joining legitimate P2P networks, in which botmasters can propagate commands. In fact, the initial version of `Storm` adopted this "blending" strategy, while recent P2P botnets, including the most recent version of `Storm`, `Waledac`, and `Zeus`, build their own P2P networks. Nevertheless, the "blended" P2P bots may make our detection more difficult because bots' fingerprint clusters will be identical to those of legitimate P2P applications. However, since two bots will search for the same commands and thus their fingerprint clusters are still likely to have a large overlap of contacted peers, which may serve as a discriminative feature by itself. In spite of that, we acknowledge that "blended" P2P bots make our detection very challenging in certain circumstances. For example, when two P2P bots relay queries from legitimate peers, their fingerprint clusters are unlikely to have large peer-overlaps.

C. Evasion by Limiting the Activity of P2P Bots

Since the Coarse-Grained P2P Bot Detection component leverages the feature that bots are usually persistently active on the underlying compromised hosts, botmasters may significantly reduce the active time of each P2P bot in order to evade our system. Specifically, P2P bots can repeat the following pattern: bots are active for only a short time and then silent for a long period. While this technique might be effective, it will introduce significant negative consequences to botnets. First, the usability of the whole botnet is tremendously limited as in most of the time bots will be inactive. Second, bots will frequently join and leave the botnet, which results in high churn rates [20]. The high churn rates affect the performance of P2P networks significantly, causing routing failures, loss of stored resources, and inconsistency. They may even lead to a complete disruption of the overlay network [29], [30]. In addition, botmasters could also reduce the number of peer IPs (or BGP prefixes) contacted by each bot to bypass the fine-grained P2P client detection component. Nevertheless, this approach could have a serious negative impact on the efficiency and resiliency of the C&C infrastructure [31].

D. Evasion by Building Topology-Aware P2P Bots

Our system analyzes network traffic exchanged between internal hosts and external networks and it assumes that at least two bots from the same botnet. Therefore, botmasters may design topology-aware P2P botnets for evasion. To be specific, P2P bots inside the monitored network can form a P2P network whereas only one among them is responsible for communicating with external peers. In order to address this challenge, we can further improve our system. First, we can implement our system in a distributed manner, where all components except the Fine-Grained Bot Detection component can be moved "close" to end hosts to gain complete view of their network activities. Fingerprint clusters of each single P2P client can then be acquired by taking into account the traffic it exchanged with other internal hosts. A centralized Fine-Grained P2P Bot Detection Component can perform detection based on all fingerprint clusters. As programmable network devices (e.g., switches) gain popularity, such improvement becomes practically feasible.

E. Evasion by Tunneling P2P Traffic Through Tor Networks

As suggested by Dennis Brown [32], botmasters could leverage Tor's hidden service to build P2P C&Cs that are extremely robust against disruption efforts. In a Tor-based P2P botnet, each peer can announce a service descriptor through Tor's hidden service with its IP address concealed. As a consequence, the last component in our system will be affected since bots' IP addresses are invisible to our system. However, the first three components can still generate accurate fingerprint clusters. In this case, we need to extract new features from fingerprint clusters and leverage them for detection.

F. Evasion by Randomizing Communication Behaviors

Bots could randomize their P2P communication patterns to prevent our system from getting accurate fingerprint clusters for P2P protocols. In this case, we can measure the number of failed connections to perform coarse-grained detection of P2P clients and also adopt more general features, such as the distribution of flow/packet sizes and flow/packet intervals, to profile P2P applications.

To summarize, although our system greatly enhances and complements the capabilities of existing P2P botnet detection systems, it is not perfect. We should definitely strive to develop more robust defense techniques, where the aforementioned discussion outlines the potential improvements of our system.

VII. Conclusion

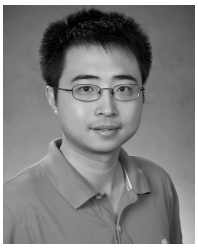
In this paper, we presented a novel botnet detection system that is able to identify *stealthy* P2P botnets, whose malicious activities may not be observable. To accomplish this task, we derive *statistical fingerprint*s of the P2P communications to first detect P2P clients and further distinguish between those that are part of legitimate P2P networks (e.g., file-sharing networks) and P2P bots. We also identify the performance bottleneck of our system and optimize its scalability. The evaluation results demonstrated that the proposed system accomplishes high accuracy on detecting stealthy P2P bots and great scalability.

References

[1] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the storm and nugache trojans: P2P is here," in *Proc. USENIX*, vol. 32. 2007, pp. 18–27.

[2] P. Porras, H. Saidi, and V. Yegneswaran, "A multi-perspective analysis of the storm (peacomm) worm," Comput. Sci. Lab., SRI Int., Menlo Park, CA, USA, Tech. Rep., 2007.

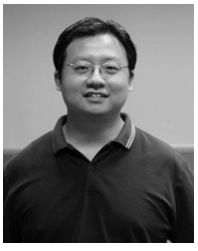
[3] P. Porras, H. Saidi, and V. Yegneswaran. (2009). *Conficker C Analysis* [Online]. Available: http://mtc.sri.com/Conficker/addendumC/index.html
[4] G. Sinclair, C. Nunnery, and B. B. Kang, "The waledac protocol: The how and why," in *Proc. 4th Int. Conf. Malicious Unwanted Softw.*, Oct. 2009, pp. 69–77.
[5] R. Lemos. (2006). *Bot Software Looks to Improve Peerage* [Online]. Available: http://www.securityfocus.com/news/11390
[6] Y. Zhao, Y. Xie, F. Yu, Q. Ke, and Y. Yu, "Botgraph: Large scale spamming botnet detection," in *Proc. 6th USENIX NSDI*, 2009, pp. 1–14.
[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, pp. 139–154.
[8] T.-F. Yen and M. K. Reiter, "Are your hosts trading or plotting? Telling P2P file-sharing and bots apart," in *Proc. ICDCS*, Jun. 2010, pp. 241–252.
[9] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "BotGrep: Finding P2P bots with structured graph analysis," in *Proc. USENIX Security*, 2010, pp. 1–16.
[10] J. Zhang, X. Luo, R. Perdisci, G. Gu, W. Lee, and N. Feamster, "Boosting the scalability of botnet detection using adaptive traffic sampling," in *Proc. 6th ACM Symp. Inf., Comput. Commun. Security*, 2011, pp. 124–134.
[11] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy P2P botnets using statistical traffic fingerprints," in *Proc. IEEE/IFIP 41st Int. Conf. DSN*, Jun. 2011, pp. 121–132.
[12] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, *et al.*, "Detecting P2P botnets through network behavior analysis and machine learning," in *Proc. 9th Annu. Int. Conf. PST*, Jul. 2011, pp. 174–180.
[13] D. Liu, Y. Li, Y. Hu, and Z. Liang, "A P2P-botnet detection model and algorithms based on network streams analysis," in *Proc. IEEE FITME*, Oct. 2010, pp. 55–58.
[14] W. Liao and C. Chang, "Peer to peer botnet detection using data mining scheme," in *Proc. IEEE Int. Conf. ITA*, Aug. 2010, pp. 1–4.
[15] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," in *Proc. ACM SIGCOMM*, 2005, pp. 229–240.
[16] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of P2P traffic using application signatures," in *Proc. 13th ACM Int. Conf. WWW*, 2004, pp. 512–521.
[17] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *Proc. 4th ACM SIGCOMM Conf. IMC*, 2004, pp. 121–134.
[18] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *Proc. ACM SIGMETRICS*, 2005, pp. 50–60.
[19] M. P. Collins and M. K. Reiter, "Finding peer-to-peer file sharing using coarse network behaviors," in *Proc. 11th ESORICS*, 2006, pp. 1–17.
[20] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proc. 6th ACM SIGCOMM Conf. IMC*, 2006, pp. 189–202.
[21] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proc. USENIX LEET*, 2008, pp. 1–9.
[22] G. Bartlett, J. Heidemann, C. Papadopoulos, and J. Pepin, "Estimating P2P traffic volume at USC," USC/Information Sciences Institute, Los Angeles, CA, USA, Tech. Rep. ISI-TR-2007-645, 2007.
[23] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proc. ACM SIGMOD*, 1996, pp. 103–114.
[24] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *J. Intell. Inf. Syst.*, vol. 17, nos. 2–3, pp. 107–145, 2001.
[25] (2011). *Argus: Auditing Network Activity* [Online]. Available: http://www.qosient.com/argus/
[26] Z. Li, A. Goyal, Y. Chen, and A. Kuzmanovic, "Measurement and diagnosis of address misconfigured P2P traffic," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[27] (2011). *Autoit Script* [Online]. Available: http://www.autoitscript.com/autoit3/index.shtml
[28] (2011). *Zeus Gets More Sophisticated Using P2P Techniques* [Online]. Available: http://www.abuse.ch/?p=3499
[29] A. Binzenhofer, D. Staehle, and R. Henjes, "On the stability of chord-based P2P systems," in *Proc. IEEE Global Telecommun. Conf.*, vol. 2. Nov./Dec. 2005, pp. 884–888.
[30] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a DHT," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.*, 2004, pp. 127–140.
[31] D. Dagon, G. Gu, C. Lee, and W. Lee, "A taxonomy of botnet structures," in *Proc. 33rd Annu. Comput. Security Appl. Conf.*, 2007, pp. 325–339.
[32] (2010). *Resilient Botnet Command and Control with Tor* [Online]. Available: http://www.defcon.org/images/defcon-18/dc-18-presentations/D.Brown/DEFCON-1%8-Brown-TorCnC.pdf



Junjie Zhang is an Assistant Professor with the Department of Computer Science and Engineering, Wright State University. He received the Ph.D. degree in computer science from the Georgia Institute of Technology in 2012, and the M.S. degree in systems engineering and the B.S. degree in computer science from Xi'an Jiaotong University, China, in 2006 and 2003, respectively. His current research focuses on network security and cyber-physical system security.

Roberto Perdisci is an Assistant Professor with the Computer Science Department, University of Georgia, an Adjunct Assistant Professor with the Georgia Tech School of Computer Science, and a Faculty Member with the UGA Institute for Artificial Intelligence. Before joining UGA, he was Post-Doctoral Fellow with the College of Computing, Georgia Institute of Technology. He was the Principal Scientist with Damballa, Inc., and received the Ph.D. degree with the University of Cagliari, Italy. His current research focuses on a multi-disciplinary approach towards better defending computer networks from cyber-attacks, by combining computer security principles with machine learning and big data mining. In 2012, he received the NSF CAREER Award for research proposal on automatic learning of adaptive network-centric malware detection models.

Wenke Lee is a Professor with the School of Computer Science, College of Computing, Georgia Institute of Technology. He is the Director of the Georgia Tech Information Security Center. His research interests include systems and network security, applied cryptography, network management, and data mining. He received the Ph.D. degree in computer science from Columbia University in 1999, the B.S. degree in computer science from Sun Yat-Sen University, China, and the M.S. degree in computer science from the City College of New York.



Xiapu Luo received the Ph.D. degree in computer science from Hong Kong Polytechnic University in 2007, and he was with the Georgia Institute of Technology as a Post-Doctoral Research Fellow. He is currently a Research Assistant Professor with the Department of Computing, Hong Kong Polytechnic University. His current research focuses on network security and privacy, internet measurement, and smartphone security.

Unum Sarfraz received the B.E. degree in information and communication systems from the National University of Science and Technology (NUST), Pakistan, in 2008, and the M.S. degree from the Georgia Institute of Technology, Atlanta, in 2010. From 2008 to 2009, she was a Research Assistant with the School of Electrical Engineering and Computer Science, NUST. Since 2011, she has been a Software Engineer with Cisco Systems. Her research interests focus on network security and anomaly detection.