PeerClean: Unveiling Peer-to-Peer Botnets through Dynamic Group Behavior Analysis

Qiben Yan, Yao Zheng, Tingting Jiang, Wenjing Lou, Y. Thomas Hou
Virginia Polytechnic Institute and State University, VA, USA

***Abstract*—Modern botnets adopt peer-to-peer (P2P) infrastructure for more resilient command and control (C&C) without relying on a central server. Traditional signature, sandboxing, and blacklist based detection techniques become less effective in identifying bots that communicate using a P2P structure. In this paper, we present PeerClean, a novel system that detects P2P botnets in real time using only high-level features extracted from C&C network flow traffic. PeerClean reliably distinguishes P2P bot-infected hosts from legitimate P2P hosts by jointly considering flow-level traffic statistics and network connection patterns. Instead of working on individual connection or host, PeerClean clusters hosts with similar flow traffic statistics into groups. It then extracts the collective and dynamic connection pattern of each group by leveraging a novel dynamic group behavior analysis. Comparing to the individual host-level connection patterns, the collective group patterns are more robust and differentiable. Multi-class classification models are then used to identify different types of bots based on the established patterns. To increase the detection probability, we further propose to train the model with average group behavior, but to explore the extreme group behavior for the detection. We develop a prototype system, and evaluate it on real-world flow records from a campus network with IPv4 space of size /16. Our evaluation shows that PeerClean is able to achieve high detection rates with few false positives in identifying P2P botnets.**

I. Introduction

Botnet has become a major threat to the health of modern networks. Through large-scale compromise of end hosts, botmasters can commit organized cyber-crimes, such as launching distributed denial-of-service (DDoS) attacks, sending spam, performing click fraud, or stealing sensitive information.

The C&C channel is one of the most essential components of a botnet, through which a botmaster manages the bot army of compromised end hosts. One common type of C&C infrastructure relies on a central C&C server, which has recently drawn a great deal of attention from security researchers and law enforcement forces. For the attackers' point of view, such centralized architecture suffers from a *single point of failure* problem, because if the C&C server is identified and taken down, the botmaster will lose control over the whole botnet. As a response, sophisticated botnet developers attempt to build more advanced and resilient P2P C&C infrastructure [1]. P2P C&C allows the bots to exchange C&C messages via their connected peers in a P2P manner. Therefore, despite of numerous takedown attempts [2], P2P botnets kept reviving. Some notable examples of active P2P botnets include Sality [3], ZeroAccess [4], and Kelihos [5], which have survived in the wild for a long time and will likely continue to be alive in the near future [6].

To date, a few solutions have been proposed to detect P2P botnets [7]–[10]. Host-level malware detection techniques such as traditional signature-based approaches and more recently behavior-based approaches [11] have been designed. However, these approaches are subject to advanced malware obfuscation or polymorphism and require host-side installation, and thus appear unappealing to the network administrators aiming at uncovering a network-wide botnet. Alternatively, network-level techniques have been proposed to correlate the traffic patterns of suspicious bots [8], [9], [12]–[15] or collect network communication graphs to identify P2P bots [7], [16]. Some of them apply *deep packet inspection* (DPI), which is not only computationally expensive, but is also evadable through message encryption. Other approaches are based upon network flow traffic analysis. For instance, Yen et al. [8] described an algorithm to differentiate P2P file sharing applications with P2P bots based on network traffic features such as traffic volume, peer churn rate and interstitial time distribution. Recently, Zhang et al. [9] developed a botnet detection system to extract statistical fingerprints for every host, and identify the bots based on a set of traffic features such as communication persistency, fingerprint similarity, and shared contacts' number. However, the traffic features used in these approaches are not robust enough to identify bots in a dynamic network, as observed from our experiments. On the other hand, the communication graph-based approach [7] seems more reliable, but it can only identify structural P2P subgraphs regardless of whether the subgraphs contain bots. Also, it requires a list of *honeypot* hosts to bootstrap its detection algorithm, limiting its practicality.

In this paper, we jointly consider two sets of features: flow traffic statistics and network connection behaviors, to reveal the presence of P2P bots within a monitored network, such as a campus network or ISP network. We introduce *PeerClean* to utilize the best of these feature sets via a novel combination of clustering and classification. PeerClean identifies P2P bots by detecting their C&C communication patterns that characterize the botnets, regardless of how they perform malicious activities. In order to accomplish this goal, we extract the flow traffic statistics from the network flow data, based on which we group the hosts with similar traffic patterns into the same cluster. PeerClean leverages the fact that bots from the same botnet use the same C&C communication protocol which produces similar traffic patterns [9], [13], and the same type of bots thereby are highly likely to be included in the same cluster.

However, the flow traffic features are not robust enough for distinguishing specific bots from benign hosts due to the dynamics exhibited in the Internet traffic. Therefore, PeerClean further incorporates more robust network connection patterns for a more accurate bot identification. Rather than exploring the connection persistency of every individual bot [8] or the number of overlapping peers of every bot pair [9], we propose a *dynamic group behavior analysis* (DGBA) method to investigate the group-level connection behaviors inside botnets. We apply DGBA to every host cluster so as to extract the aggregated connection features. PeerClean then trains a *support vector machine* (SVM) classifier using the group-level training features, and labels each cluster subsequently. To improve the detection performance, we train the classifier using average group behavior, but explore the extreme group behavior for the detection. While all the existing network-based detection approaches identify the bots without specifying their corresponding botnet types, we consider the botnet type as a useful piece of information for the network administrators to evaluate the damaging impacts and decide corresponding bot-specific countermeasures. Furthermore, the PeerClean system is tailored to support real-time bot detection, enabling a quick response to the bot infections.

Specifically, this paper makes the following contributions:

- We propose a novel botnet detection framework, PeerClean, using the high-level features extracted from network flow data based on the flow-level traffic statistics and dynamic network connection patterns. It explores the best of these different features with a novel combination of unsupervised (clustering) and supervised (classification) machine learning methods.
- We design a dynamic group behavior analysis method to automatically extract the collective connection features from P2P host clusters. We show through experiments that the extracted group features are robust, reliable and effective in identifying various types of P2P botnets.
- We develop a prototype system, and evaluate the system using network traces from various real-world botnet families (including Sality, Kelihos and ZeroAccess), as well as background traces from a large campus network. We demonstrate through experiments that PeerClean can identify different types of bots with up to 95.8% accuracy, and negligible false positive rate.

The rest of the paper is organized as follows. We start by describing an overview of the proposed PeerClean system (§II), followed by the system design details (§III). Then, we provide further discussions (§IV). After that, we present a comprehensive evaluation of PeerClean (§V). Finally, we discuss related work (§VI) before we conclude (§VII).

II. Overview of PeerClean

Our primary goal is to design a detection system for the network administrators to identify P2P bots in a monitored network. Toward that goal, we present our data-driven detection framework, PeerClean, which exploits network flow data captured at the edge of the network. In this section, we give a brief overview of the PeerClean system.

Figure 1 shows the system flow of PeerClean. The upper part of the figure describes the training process, with inputs from two labeled data sets: one is a subset of monitored NetFlow data that is from the labeled legitimate P2P hosts, and the other one contains the data from the labeled P2P bots (we discuss the acquisition of training data in §V-A). For each type of legitimate P2P hosts and P2P bots, PeerClean then performs DGBA training to extract a collection of group-level connection features aggregated from all the hosts of this specific type, and trains a SVM classification model using the extracted group-level features. The bottom part of the figure presents the detection process with input of monitored NetFlow data. After identifying the P2P hosts in the network, PeerClean carries out P2P host clustering using the statistical features of their traffic flows, and applies DGBA detection to every cluster of interest with the goal of detecting clusters containing bots. Finally, the refined bot identification picks out the bots from the clusters for further processing. PeerClean can be regarded as a three-layer system, with the first-layer modules processing every host, the second-layer modules operating on the host group/cluster, while the third-layer modules further handling the identified bot clusters.

Input Data: The input data set consists of a training data set and a testing data set of NetFlow format. The testing NetFlow data can be one hour (or one day) flow traffic traces captured at the gateway router of a campus (or enterprise, ISP) network, while the training data set is constructed by the traffic from identified P2P bots and legitimate P2P hosts. Specifically, the training data of P2P bots can be imported from honeynets running in the wild, and the training data of legitimate P2P hosts can be derived via identifying legitimate P2P traffic in the captured payload-available traffic traces using a signature matching method [17].

P2P Host Identification: The high-speed networks generate a huge amount of NetFlow data, which would potentially overwhelm the processing capability of our detection system. Thus, the first step of PeerClean is to reduce the traffic volume by filtering out the hosts that are unlikely to be related to P2P communications. Our approach is based on the observation that the hosts engaging in P2P communications exhibit high failed connection rates mainly caused by the high peer churn rate [18]. Therefore, we compute the percentage of failed connections inside each time epoch (e.g. 1 hour). The hosts with failed connection rate higher than an empirical threshold are selected as *candidate* P2P hosts [8]. This selection process allows us to retain hosts engaging in P2P communications, while eliminating a vast majority of non-P2P hosts.

Flow Statistical Feature Extraction: Having eliminated most of the traffic from non-P2P hosts, we extract a collection of flow statistical features from the network flows of each candidate P2P host. We propose two sets of features, including flow size statistical features (e.g., the average number of bytes per flow transmitted from each host) and host access pattern features (e.g., the outgoing/incoming flow interarrival time

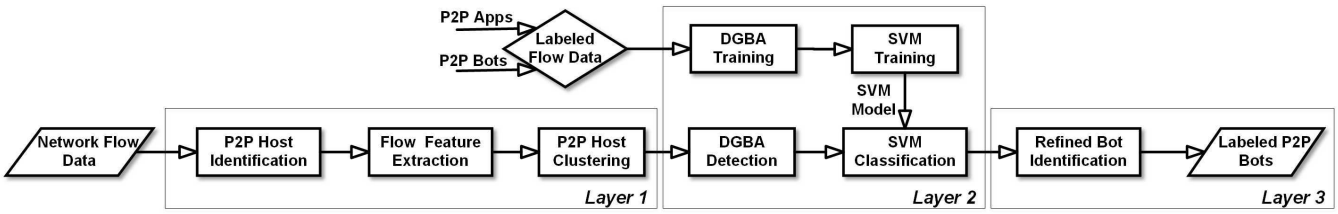


Fig. 1: PeerClean system flow

from each host). We will explore the motivation behind these features in §III-A.

P2P Host Clustering: The objective of host clustering is to group together the hosts with similar flow patterns. Since P2P bots from the same botnet share the same P2P network and communication protocol, their flow features are likely to be clustered together in the feature space. Clustering techniques aim at finding meaningful clusters that are both compact and well separated from each other. PeerClean applies a clustering algorithm to generate the clusters of hosts solely based on their flow statistical features. Ideally, we will gather the bots belonging to different types of botnets into separated and compact clusters without including any legitimate hosts, which is often not the case in reality.

Dynamic Group Behavior Analysis: DGBA investigates the aggregated connection behaviors of the entire cluster of hosts. The group-level connection features include cluster connectivity feature (i.e., how the cluster connects to the outside world), shared neighbor feature (i.e., the shared contacts of the hosts inside the cluster), significant connection feature (i.e., the connections contributing a significant amount of network traffic), and temporal feature (i.e, how the connection behavior evolves over time), the details of which are presented in §III-C. We divide DGBA into two phases for training and detection respectively. DGBA training extracts average group-level connection features from the group of P2P bots or legitimate P2P hosts of each type, while DGBA detection examines extreme group-level connection features of every unlabeled cluster in order to identify clusters containing P2P bots more accurately.

Training and Classification: As network traffic pattern is often distorted by noise, PeerClean trains a non-linear SVM classifier due to its robustness against noisy data, using the group-level connection features from DGBA. After the classifier is constructed with the training data set, PeerClean applies this classifier for classifying the unlabeled clusters generated from the testing data set. Hopefully, each cluster containing a certain type of P2P bots will be assigned a label corresponding to that specific bot type.

Refined Bot Identification: Finally, for each cluster classified as a cluster with bots, we further identify the P2P bots inside the cluster based on their individual connection behaviors. We leverage a simple threshold-based approach to discriminate P2P bots from falsely included benign users in the cluster. The identified P2P bots are then labeled and confirmed as "infected", calling for subsequent bot cleanup actions from the network administrators.

Detection Period: Since bot memberships are dynamically changing with some bots been cleaned up and others been newly infected, we propose to perform bot detection periodically. PeerClean supports various configurations of detection periods, as long as bots generate enough network flows with representable flow and connection features during that period. In this paper, we select one hour as the detection period in response to agile bot infections. Specifically, PeerClean produces one SVM model for each hour of the day. Then, by examining each hour of testing traces collected from the edge routers, PeerClean identifies specific types of bots existed in the network within that hour. In this manner, PeerClean enables real time bot detection which supports a fast response to the bot infections (i.e. one hour response time in this paper).

III. System Design

PeerClean systematically incorporates two categories of features including flow statistical features and network connection features. The effectiveness of PeerClean largely hinges upon the discriminative ability of the selected features to set apart various P2P bots and legitimate P2P hosts. In this section, we discuss about the rationale behind the feature selection, and look into the strength and weakness of the selected features. Meanwhile, two machine learning techniques performing clustering and classification are described, which are used to gather, identify and subsequently label the P2P bots.

A. Flow Statistical Features

The performance of host clustering relies on a set of carefully selected network flow features. A common criticism of early attempts using machine learning methods over network flow data is that the selected features were often not *robust*, resulting in an overfit model to some specific features of the training set, such as a particular port or IP address used by a bot. Dedicated bots can simply adapt their used ports and IP addresses to invalidate such flow analysis. To avoid the overfitting issue, we select flow features that are both robust and distinctive among the botnets, including *flow size statistical features* and *host access pattern features*. As some P2P botnets mostly use TCP protocol for C&C (e.g. Kelihos), while others simply carry out UDP transmissions, we divide the traces into TCP and UDP flow segments, and examine their flow statistical features separately. To yield credible statistical features, we only consider the hosts with

Feature	Descriptions
Bytes-per-flow pattern	The distribution of the number of bytes per flow sent from (received by) a host
Packets-per-flow pattern	The distribution of the number of packets per flow sent from (received by) a host

TABLE I: Flow size statistical features

Feature	Descriptions
Flow interarrival pattern	The distribution of the incoming (outgoing) flow interarrival time at a host
Flow density pattern	The fraction of the time units with more than x flows at a host
Diurnal pattern	The percentage of the flow numbers in the peak (dip) period of the day at a host

TABLE II: Host access pattern features

100+ TCP/UDP outgoing flows and 100+ incoming flows during the one hour detection period E. Note that, at this stage, we only extract the flow features of candidate P2P hosts who survived the P2P host identification process.

Flow Size Statistical Features: Flow size statistical features capture the flow size distribution for both outgoing flows and incoming flows at a specific host. Let $F_i^{(ob)} = \{f_j^{(ob)}\}_{j=1..m}$ and $F_i^{(ib)} = \{f_j^{(ib)}\}_{j=1..n}$ denote the series of flows sent from or received by host i inside E. We consider the basic flow size related features such as: *bytes-per-flow* (bpf) feature and *packets-per-flow* (ppf) feature, as shown in Table I. Note that each feature records the distribution of the flow sizes among all the outgoing (incoming) flows at the corresponding host. In particular, we extract the mean $\mu_{F_i^{(ob)}}$, $\mu_{F_i^{(ib)}}$ and the standard deviation $\sigma_{F_i^{(ob)}}$, $\sigma_{F_i^{(ib)}}$ of bpf and ppf from both the outgoing and incoming flows respectively.

This group of features characterizes the regularity of traffic flow size over time for each host. The reason for selecting flow size features is because the flows carrying C&C information are preferred by the botmaster to be as short as possible to remain stealthy under the surveillance of various network monitoring tools. Moreover, due to the limited types of C&C messages, only a small fluctuation on these features is expected. On the other hand, legitimate P2P applications usually generate flows with large, yet highly variable flow sizes, with a few exceptions such as skype application, which has a small flow size when used as instant messenger or voice-over-IP client. Therefore, flow size statistical features are promising for differentiating bots from legitimate hosts, but it alone may not be enough to create dense bot clusters without including a large number of legitimate hosts such as the skype hosts.

Host Access Pattern Features: We introduce host access pattern features to capture the flow arrival patterns. Table II lists the adopted features, including flow interarrival pattern, flow density pattern and diurnal pattern. Assume $T_i^{(ob)}$ ($T_i^{(ib)}$) is a time series of the starting time of outgoing (incoming) flows from host i inside E, based on which we can compute a sequence of flow interarrival time $I_i^{(ob)}$ ($I_i^{(ib)}$) by taking the difference of the starting time of two consecutive flows. Flow interarrival feature represents the statistical features of flow interarrival time sequences, including the minimum, maximum, median and standard deviation.

Different from all the aforementioned features which are extracted inside each detection period E, the last two types of features, flow density pattern and diurnal pattern, are determined anew every day. We define a time unit as a three-hour period with one whole day been divided into 8 time units. We assume the flow amounts of each time unit during the day pertaining to a certain host as $N_j, j = 1, 2, \cdots, 8$. Flow density pattern records the fraction of time units having $\geq x$ flows per day, i.e., $\frac{\sum_{j=1}^{8} \sigma(N_j \geq x)}{8}$, where $\sigma()$ is a step function yielding one when $N_j \geq x$ satisfies, and zero otherwise. In our prototype, x is empirically set as 1000. In addition, to assess whether the flow arrival displays a diurnal pattern, we take two percentages regarding to the flow amounts during the peak period or dip period[1] as diurnal pattern features, i.e., $\frac{N_P}{\sum_{j=1}^{8} N_j}$ and $\frac{N_D}{\sum_{j=1}^{8} N_j}$, where N_P and N_D denote the flow numbers at the peak period and dip period respectively. These two types of features are inserted as additional features for the last hour of the day, further elevating the chance of differentiating various P2P bots from legitimate P2P hosts.

B. P2P Host Clustering

The basis of host clustering relies on the following observation: bots that belong to the same botnet run the same P2P communication protocol and share the same C&C messages. In the literature, there are a wide variety of clustering methods, but a well-suited algorithm for PeerClean should be cautiously selected, because: first, the clustering algorithm for gathering hosts with similar flow patterns not only determine the subsequent bot detection, but also affect the system efficiency; second, P2P host clustering is a challenging task, since the percentage of infected hosts in the network is generally small compared with the benign hosts. Thus, the clustering objective is to separate the small number of P2P bots from a large number of benign P2P hosts. In this respect, partition-based clustering methods suit our problem well [19].

Affinity Propagation (AP) is a recently proposed partition-based clustering method by Frey and Dueck [20]. Compared with K-means, one of the most popular clustering methods [21], the performance of AP does not rely on an initial selection of *exemplars*[2] or cluster centers. Rather than specifying the number of clusters, AP can automatically determine it solely based on the data. Whereas K-means clustering follows a greedy heuristics to find the optimum of a combinatorial

[1]The peak time is expressed as $P = \arg\max_j N_j$ with flow amount $N_P = \max_j N_j$, and the dip time as $D = \arg\min_j N_j$ with flow amount $N_D = \min_j N_j$, $j = 1, \cdots, 8$.

[2]Exemplar represents for the cluster center that best accounts for the data in the cluster [20].

optimization problem, which is prone to local minima, AP considers all data points as potential exemplars and tackles the optimization problem by exchanging messages between pairs of points until the clusters gradually emerge. Thus, AP provides a guarantee of quasi global optimization [20].

The similarity $s(i,k)$ of AP indicates how well data point x_k is suited to be the exemplar of data point x_i. With the goal of minimizing squared error, we use negative squared error (*Euclidean distance*) as the similarity measure, i.e., $s(i,k) = -\|x_i - x_k\|^2$ [20]. Since unsupervised learning is a notoriously difficult task, it seems impossible to obtain a perfect clustering result. In consequence, besides of several clearly separated *bot clusters* (i.e. clusters of bots) and *benign clusters* (i.e. clusters of benign hosts), we expect some clusters to include both benign hosts and bots, which we call *mixed clusters*. For ease of exposition, the bot clusters and mixed clusters are collectively called *bot-included clusters*. In the following section, we will show how we use supervised learning to identify and further examine bot-included clusters, as well as the method of spotting bots inside them.

C. Dynamic Group Behavior Analysis

In this section, we introduce DGBA with the objective of identifying bot-included clusters. DGBA is based on our intuition that the bot-included clusters have cluster-level aggregated characteristics that are distinguishable from benign clusters. Whereas the connection activity of an individual host is highly dynamic and unidentifiable, we believe the group connection behavior will help us identify bots' communications.

To enhance the detection capability, we propose two modules: DGBA training and DGBA detection, to extract features from the training set and testing set respectively. The purpose of DGBA training is to extract the representative group behavior from a collection of labeled P2P hosts to build SVM classifiers, whereas DGBA detection searches for the abnormal behaviors from every unlabeled cluster to spot P2P bots. Thus, we propose to use different statistics of the collected host-level features from a group to represent group-level training and detection features, respectively. Specifically, the training features capture the average group behavior, while the detection features capture the extreme group behavior (i.e. the maximum or the minimum). Note that all the features below are extracted from the collection of traces inside each detection period if not otherwise stated.

1) Cluster Connectivity Feature: Cluster connectivity feature captures the aggregated connectivity of the peers inside each cluster. A connection between two hosts can be successful or failed. We define a *good connection* as a successfully established connection between two hosts, one from the cluster and one from its outside. We consider a TCP connection as good if it takes a complete SYN, SYN/ACK, ACK handshake flags, and also a UDP connection as good if at least one response packet is followed by a request packet. We denote the good connection set of host h_i as $\mathcal{C}_i$ which includes all the successful connections of host h_i.

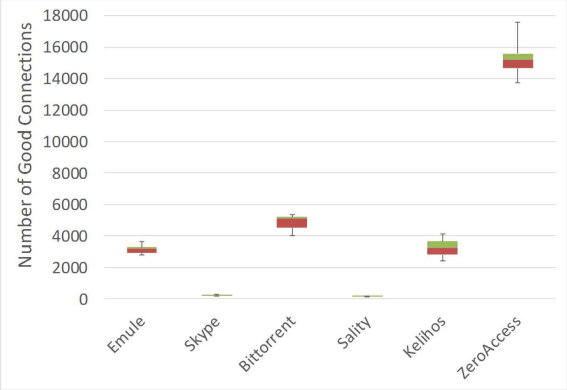


Fig. 2: Cluster connectivity feature (measured in 24 hours)

Training feature: Cluster connectivity feature for DGBA training is defined as the average number of good connections among all the P2P hosts of each type, i.e., $\sum_{i=1}^{M} |\mathcal{C}_i|/M$, assuming M hosts of one specific type exist in the training set.

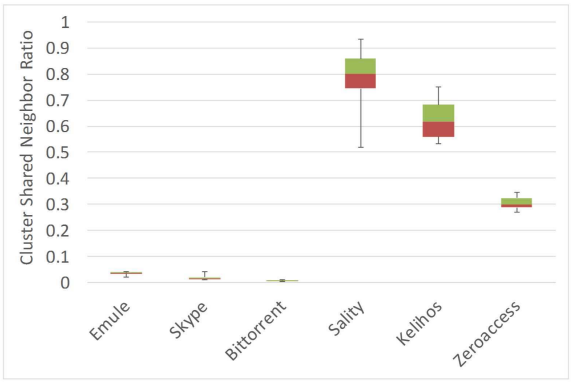
In order to see the discriminatory strength of this feature, we run an experiment using 24-hour training data (refer to §V-A for the data sets used in the experiment) to show the cluster connectivity features of different P2P bots and legitimate hosts running various P2P applications. The box-plot results (measured in 24 hours) are shown in Fig. 2, from which we notice different types of P2P hosts indeed exhibit varied cluster connectivity features. In particular, ZeroAccess bot stands out with a significantly larger amount of good connections. We attribute the difference to several factors including: (1) the botnet network size; (2) the botnet peer discovery mechanisms. For instance, the bots in a populous network with a more aggressive peer discovery mechanism are supposed to have more network connections.

Detection feature: Cluster connectivity feature for DGBA detection is defined as the maximal number of good connections among all the hosts in the unlabeled cluster, i.e., $\max_{i=1}^{M'} |\mathcal{C}_i|$, assuming M' hosts in the cluster. Fig. 2 shows a notable gap between ZeroAccess bots and other types of hosts, which means ZeroAccess bots can be detected solely based on the cluster connectivity feature. Assigned with the maximum of the connection numbers of M' hosts, the connectivity feature of the cluster containing ZeroAccess bots will be exceptionally high, which reveals the presence of ZeroAccess bots.

2) Shared Neighbor Feature: Shared neighbor feature captures the amount of shared connections between every pair of hosts in each cluster. The set of shared neighbors of host h_i and h_j can be written as: $\mathcal{C}_i \bigcap \mathcal{C}_j$. We further define *pairwise shared neighbor ratio* of a host pair as the ratio between the number of shared neighbors and the number of total neighbors, i.e., $s_{ij} = (\mathcal{C}_i \bigcap \mathcal{C}_j)/(\mathcal{C}_i \bigcup \mathcal{C}_j)$ for the host pair (h_i, h_j).

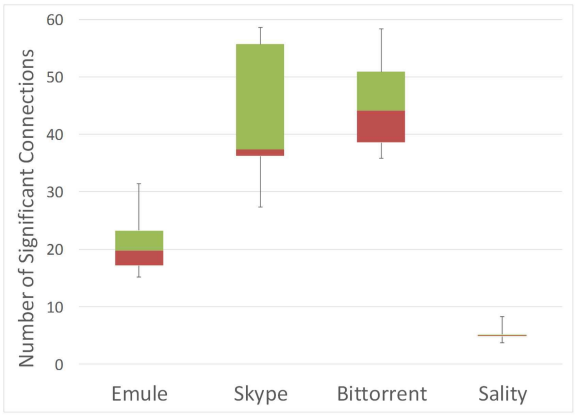
Training feature: Given the above definitions, shared neighbor feature for DGBA training is represented by

(a)

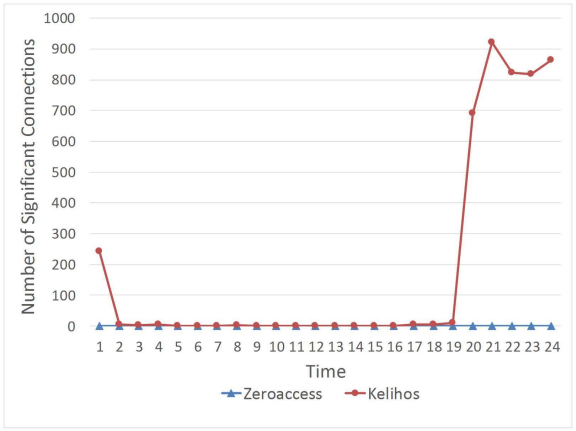


(b)

Fig. 3: (a): The shared neighbor ratio of one Emule host pair compares with that of one ZeroAccess host pair (b): Group shared neighbor ratio (measured in 24 hours)



(a)



(b)

Fig. 4: (a): Significant connection feature (measured in 24 hours) (b): Significant connection feature of Kelihos and ZeroAccess bots in 24 hours

group/cluster shared neighbor ratio, simply defined as the average pairwise shared neighbor ratio among all the host pairs of one type, i.e., $\sum_{i,j\in[1,M],i\neq j}\{s_{ij}/\frac{M(M-1)}{2}\}$. Previous work has adopted pairwise shared neighbor ratio s_{ij} [9] to distinguish between bots and benign hosts. However, according to our experiment, pairwise shared neighbor ratio seems ineffective in identifying certain pairs of P2P bots. In Fig. 3(a), we compare the pairwise shared neighbor ratio of an emule host pair (who download the same file) with that of a ZeroAccess bot pair. We find it almost impossible to make a distinction between these two pairs, which brings false positives or false negatives. In contrast, group shared neighbor ratio clearly differentiates ZeroAccess bots from the emule hosts with a large gap between them, via feature aggregation from multiple hosts.

In addition, Fig. 3(b) shows different types of P2P bots and P2P hosts exhibit distinguishable shared neighbor features measured in 24 hours, where we observe P2P bots have much higher group shared neighbor ratios compared with legitimate P2P hosts. The reason is obvious - the bots from the same botnet *search for the same commands* published by the botmaster [9], which makes their contacted peers more likely to be shared by other companions. Furthermore, although P2P botnets have decentralized C&C architecture, botmasters still strive to make their P2P network robust against peer churns and provide end-to-end communication with a minimum delay. This inherent C&C objective translates into a convergence of contacted peers by a group of bots to ensure the reliable delivery of C&C messages. On the other hand, different legitimate P2P hosts generally search for different contents from their peers, which yields a more dispersed peer list.

Detection feature: Correspondingly, shared neighbor feature for DGBA detection is defined as the maximal pairwise shared neighbor ratio among all the host pairs in each cluster, i.e., $\max_{i,j\in[1,M'],i\neq j} s_{ij}$. The shared neighbor feature of every bot-included cluster will again be determined by the bots, since bots have significantly higher shared neighbor ratios than benign hosts, which will help uncover the presence of bots in the cluster.

3) Significant Connection Feature: Significant connection (SC) feature captures the amount of *hot links* in the network, i.e., the connections that contribute significantly larger amounts of network flows compared with the other connections. The SCs extracted from the Internet traffic data have been used to diagnose the network operation and quickly identify

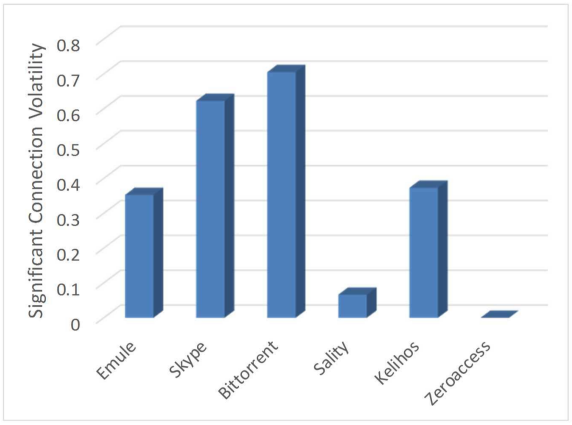


Fig. 5: Significant connection volatility

the anomalous events [22]. Similarly, we try to identify SCs of bot groups for better understanding the bots' behaviors and accurately identifying bots' presence. We leave the details of SC feature extraction in Appendix A.

Training feature: We define the SC feature for DGBA training as the average number of SCs for all the hosts of one type. Fig. 4 shows the SC features of Sality bots and three other types of P2P hosts measured in 24 hours. Compared with Sality bots, these legitimate P2P hosts produce a larger number of SCs.

This distinctive phenomenon may be attributed to the following fact: the SCs in a botnet indicate the existence of some active bots that are critical to the P2P botnet infrastructure. These active bots may be well connected with a high bandwidth connection, or may be close to the botmaster. Few number of distinctive connections favors the botnets in remaining stealthy under the radar of numerous intrusion detection systems. Conversely, benign P2P hosts yield a much higher number of SCs due to their unorganized nature.

Interestingly, the traffic flows from ZeroAccess and Kelihos bots reveal unique SC patterns as shown in Fig. 4(b). ZeroAccess bots simply have none SCs, while Kelihos bots suddenly generate a large amount of SCs from a "hot" period between $7pm$ to $1am$. This period perhaps can be interpreted as a peak period of C&C message exchanging, with so many suddenly emerging hot links. The study of this abrupt phenomenon and the exact origin of SCs of botnets are out of scope of this paper, but may become research topics on their own rights.

Detection feature: Among all the hosts in the cluster, SC feature for DGBA detection is defined as the minimal number of SCs, or the maximal number if it exceeds an empirical threshold α. Thus in most cases, the SC feature of bot-included cluster will be dominated by the bots with less SCs. However, the number of SCs of Kelihos bots skyrockets during the "hot" period, which far exceeds that of the normal hosts. Hence, by setting an appropriate threshold α (e.g. 200), the group-level feature of the cluster containing Kelihos bots will again be ruled by the bots, revealing the existence of Kelihos bots.

4) Temporal Feature: Lastly, temporal feature captures the dynamic evolvement of SC sets. Instead of performing feature extraction per one-hour detection period, temporal features are computed at the end of each day to combat noise and disturbance, which are represented by *significant connection volatility*, measuring whether the cluster has the same set of SCs over time. We assume the number of distinct SCs for host h_i over the whole day is U_i, and the number of SCs during k-th hour is S_{ik}, $i = \{1, \ldots, M\}$, $k = \{1, \ldots, 24\}$. SC volatility of host i is defined as: $\Phi_i = \frac{U_i}{\sum_{k=1}^{24} S_{ik}}$. Obviously, if the SC sets of the 24 hours are all different, we have $\Phi_i = 1$. On the contrary, when the same set of SCs appears every hour, we have $\Phi_i = 1/24$. In general, the less volatile the set of SCs is, the closer Φ_i is toward zero.

Training feature: The temporal feature for DGBA training is represented by the average SC volatility of all the hosts of the same type, expressed as: $\frac{1}{M}\sum_{i=1}^{M}\Phi_i$. Fig. 5 shows different temporal features for various P2P bots and legitimate P2P hosts. We notice that Sality and Zeroaccess bots have small volatility features, while emule hosts and Kelihos bots have a moderate value of SC volatility. SC volatility is related to a number of factors, such as the number of SCs, the size of P2P networks and how dynamic the network connections are.

Detection feature: The temporal feature for DGBA detection captures the minimal SC volatility of all the hosts in the cluster, i.e., $\min\Phi_i$. Therefore, the temporal feature of bot-included cluster will be determined by the bots, whose SC sets appear less volatile. In the end, a small value of temporal feature reveals the presence of bots (i.e., Sality or ZeroAccess).

D. Training and Classification

Data Preprocessing: Data preprocessing tries to cope with the issue that the collective features extracted from the network flow data have different data ranges. To make sure every feature in the feature sets is given equal importance, we perform feature-wise normalization to shift and re-scale each feature value so that they lie within the range of $[0, 1]$.

Multi-class Classification: Support vector machine is adopted as our main classification method due to its robustness, efficiency and excellent non-linear classification performance. In particular, we use multi-class SVM classification to assign each cluster one label corresponding to a specific type of botnet or a non-bot host. We denote the multiple labels as $\{B_1, B_2, \ldots, B_k\}$, assuming $k-1$ classes of botnets with the last class representing non-bot label. The basic component of SVM method is a binary classification mechanism, which classifies an unlabeled cluster based on the distance of its feature to the decision hyperplane with norm vector $\mathbf{w}$ and constant b:

$$f(x) = \mathbf{w}^T\mathbf{x} + b = \sum_{\forall i} y_i\alpha_i K(\mathbf{x}_i, \mathbf{x}) + b, \qquad (1)$$

where $\mathbf{x}_i$ is the feature vector of host i from the training set, $y_i \in \{-1, 1\}$ denotes the label of the training data, and the parameters α_i determines whether the host i is a support vector ($\alpha_i > 0$) or not ($\alpha_i = 0$). The feature vector $\mathbf{x}_i$ is transformed

into a higher dimensional space by a non-linear kernel function $K(\mathbf{x}_i, \mathbf{x})$.

Two-class SVM determines $\mathbf{w}$ and b by searching for the optimal hyperplane to separate the feature space into two parts. This is also termed as a maximum margin approach, since the objective is to maximize the distance between training data and decision hyperplane. Multi-class SVM model is built by combining multiple two-class SVM models. For a K-class SVM model ($K > 2$), we use "*one-versus-one*" approach [23], in which $K(K-1)/2$ classifiers are trained on all possible pairs of classes, and then a voting strategy is used to classify the clusters according to which class has the highest number of votes. The clusters labeled as specific types of botnets become bot-included clusters, demanding a further inspection.

E. Refined Bot Identification

After labeling bot-included clusters, the final step is to extract bots from the cluster based on their individual connection features. Utilizing the experimental results of training features, we devise a *feature test* to separate bots from benign hosts who happen to be in the same cluster. The feature test exploits the differences of various connection features between bots and benign hosts, which is shown in Algorithm 1. A number of threshold values are defined to empirically set apart bots and benign hosts (e.g. $\lambda_1 = 8000$, $\lambda_2 = 0.2$, $\lambda_3 = 10$, $\lambda_4 = 200$, $\lambda_5 = 0.2$). As long as one type of features satisfies the statement, the host is identified as bot.

Algorithm 1 Feature Test

```
for each bot-included cluster do
    for each host in the cluster do
        host ← benign host label
        if number of connections > λ1 or
    shared neighbor ratio with any peer > λ2 or
    number of significant connections < λ3 or > λ4 or
    significant connection volatility < λ5 then
            host ← bot label
        end if
    end for
end for
```

F. Putting Them All Together

A summary of PeerClean detection system is presented in this section. For the labeled hosts in the training data set, PeerClean extracts their training features corresponding to the average of each group-level feature using DGBA training, then builds a multi-class SVM model. For the hosts in the testing data set, PeerClean first clusters the hosts based on their flow statistical features. Each cluster undergoes DGBA detection to extract the detection features corresponding to the maximum or minimum of each group-level feature. Then, each cluster is designated a label by the SVM model to indicate whether it contains a specific type of bots. Finally, PeerClean employs a refined bot identification algorithm to pick up the bots from each bot-included cluster.

G. Evasion mechanisms and Limitations

PeerClean detects botnets without relying on deep packet inspection, which already raises the bar for botnet authors. In the following, we discuss the potential evasion mechanisms that botnet authors might use to thwart PeerClean.

The bots may disrupt the clustering mechanism by not following the same transmission protocol. However, that will increase the complexity of bot implementations and will also affect the efficiency of C&C message exchange. Evading DGBA is even harder to achieve. The possible attempts to evade the DGBA detection include lowering the connection number, lowering the shared neighbor ratio, raising the significant connection number and raising the significant connection volatility. The change of one or more connection features will greatly affect the P2P network operation and may compromise the stealthiness of the botnets. The collective features enlarge the gaps between the bots and benign hosts. To make the collective features indistinguishable from those of benign hosts will require substantial work on designing a complex botnet. We leave the design of botnets to evade DGBA as future work.

Since PeerClean identifies the bots based on the traffic flow statistics from every host, a limitation of PeerClean is in identifying the bot-infected hosts running P2P legitimate application simultaneously and persistently. In this case, the bot traffic might be obscured by the traffic from P2P legitimate applications. Since PeerClean performs detection per hour, the smart bots would have to run P2P legitimate applications all the time to prevent from being discovered. However, most P2P nodes have fast peer churn rate with short communication sessions [18]. Thus, it is unlikely for the P2P hosts to run P2P legitimate applications with P2P bot protocol persistently, which makes the bots reveal themselves at a certain point of time. On the other hand, the future bots might intentionally run the bot protocol together with legitimate P2P applications. Nevertheless, this will affect the communication efficiency of P2P bots, which might lead to a high peer churn rate or even a complete disruption of C&C communications.

IV. Discussion

In the PeerClean system, we identify a host using its IP address captured in the traffic flow data. However, in practice, the IP address is not a reliable source for correctly identifying a particular host, due to the dynamic address assignments enabled by the DHCP protocol. Thus, some IP addresses may belong to the same host, which will bring some unexpected errors to the detection system. Fortunately, the hosts tend to finish running the applications before they go offline, and later they will return with a different IP address. In addition, the DHCP protocol will not assign IP addresses very frequently. Therefore, during the one-hour detection period, few IP addresses, if any, associated with the same hosts will be clustered together. These small number of duplicate hosts will only cause negligible impacts to the PeerClean system. Thus, we simply regard different IP addresses as different hosts.

Another issue is that some hosts may be behind NATs, such that they will appear to have the same IP addresses to the

outside. We do not consider this type of NATed bots in our system design, since these bots are typically not recruited in the botnets. Because these bots cannot be contacted by the other peers in the Internet, they will not become the P2P overlay of the botnets. On the other hand, legitimate hosts may also be behind NATs, such that the traffic flows from this NATed IP will be the aggregated traffic from multiple hosts. As long as they are not clustered into the same cluster with bots, they will not cause serious issues. Luckily, the probability that these NATed IPs are clustered together with bots will be a small value, due to their distinctive traffic flow patterns. Therefore, we simply regard the NATed IP address as corresponding to one particular host, which will also cause only negligible impacts to the PeerClean system.

V. Evaluation

In this section, we evaluate the bot identification performance of PeerClean system. We first describe the collected data sets (§V-A). Then, we show that PeerClean can well separate different types of P2P bots into different clusters, but may falsely incorporate some benign P2P hosts who have bot-like traffic patterns (§V-B). After generating host clusters, DGBA is carried out to extract group-level connection features from each cluster. By separating the data set into training and testing sets, a multi-class SVM model is trained using the labeled training set. We then evaluate the classification performance and refined bot identification performance in §V-C and §V-D, respectively. The performance comparison with two existing approaches based on network flow patterns and pairwise shared neighbor ratio is detailed in §V-E.

A. Data Collection

We use the traffic trace captured from the edge routers of a large campus network, which have two $/16$ subnets. The traffic rate is about 5000 flows per second, and was captured for one whole day in April 2013. We focus on the TCP and UDP traffic in this traffic trace. However, as the network flow trace does not include traffic payload, we cannot unveil the ground truth about whether or not the active hosts are running legitimate P2P applications.

To provide the ground truth data from legitimate P2P hosts, we run three of the most popular P2P applications in our lab machines: emule, bittorrent and skype, and collect their network flow traces. To make the traffic traces more representative, we interact with the P2P hosts using AutoIt script [24] to randomly select contents to be downloaded/uploaded (for emule and bittorrent application), or randomly generate texts to be transmitted (for skype application) at random time periods. In total, we collected one-day traces from 100 bittorrent clients, 100 skype clients and 100 emule clients in April 2013.

We also collected the network traces for three recent P2P botnets: Sality, Kelihos and ZeroAccess. These network traces were gathered by purposefully running Sality, Kelihos and ZeroAccess samples in a controlled environment. They contain 24-hour traces for 6 Sality bots, 4 Kelihos bots and 4 ZeroAccess bots. The spamming and scanning activities were blocked during the collection of network traces, thus the collected network traces mainly include C&C traffic, e.g., for peer discovery, command exchanging, etc. Note that these traces are collected when the three botnets are fully active. The traffic summary is listed in Table III. The traffic data from 300 legitimate P2P clients and 14 P2P bots constitute our ground truth data set.

Trace	Size	Dur	Pkts	TCP/UDP Flows	clients
Campus	20.7G	24h	21.5G	401,661,350	34743
Bittorrent	6.7G	24h	854M	62,674,080	100
Skype	1.1G	24h	376M	12,615,840	100
Emule	1.6G	24h	406M	18,978,800	100
Sality	40M	24h	10.8M	565,490	6
Kelihos	224M	24h	23.5M	3,249,931	4
ZeroAccess	4.6G	24h	166.9M	69,896,829	4
P2P in campus	487M	24h	608M	7,127,054	783

TABLE III: Traffic summary ('P2P in campus' denotes the traffic flows of the campus network after P2P host identification)

To make the evaluation more realistic, we overlay the traffic traces from 300 legitimate P2P hosts and 14 P2P bots onto the campus traffic trace by assigning them to randomly selected hosts of the campus network. In order to reduce the traffic volume, we eliminate flows from well-known and extremely busy servers such as DNS servers, email servers, popular website servers (such as google, facebook, youtube, etc.). After that, P2P host identification searches for the hosts with a high percentage of failed connections (with threshold of 5%). In total, we find 1097 hosts involved in P2P applications during the day, including all the 314 P2P hosts serving as ground truths and additional 783 hosts in the campus network as shown in Table III, thereby validating P2P host identification by correctly identifying all the ground truth P2P hosts.

B. Clustering P2P hosts

In this section, we evaluate the P2P host clustering performance of PeerClean system. Based on the extracted flow features in §III-A, we perform AP clustering to group together P2P bots of the same type. During the flow feature extraction, we find that almost all of the traffic flows from Sality and Kelihos bots adhere to TCP protocol, while ZeroAccess bots generate both TCP and UDP traffic. Hence, only TCP traffic patterns are used for host clustering. The parameter setting for AP clustering is listed in Table V.

The data set contains 24-hour flow traces from 1097 P2P hosts, which is divided into 24 sections with one hour per section. For each data section, we extract the flow statistical features of every host who has $100+$ outgoing TCP flows and $100+$ incoming TCP flows for the purpose of building representative flow patterns. Then, host clustering is carried out using AP clustering method based on the extracted flow statistical features. Note that, since the last two features in Table II are refreshed at the end of the day, they will only be used for clustering at the last hour.

Hour	2	4	6	8	10	12	14	16	18	20	22
Cluster Num.	29	25	24	27	30	29	32	25	29	30	28
Sality Cluster	28th	22nd	22nd	26th	28th	27th 28th	31th	22rd	22rd	29th	26th
Kelihos Cluster	13rd	24th	23rd	14th	29th	6th	5th	24th	6th	14th	27th
ZeroAccess Cluster	29th	25th	24th	27th	30th	29th	32nd	25th	29th	30th	28th
BSR_1	1	1	1	1	1	1	1	1	1	1	1
BSR_2	1	1	1	1	1	1	1	1	1	1	1
BSR_3	1	1	1	1	1	1	1	1	1	1	1

TABLE IV: Clustering result (BSR_1, BSR_2, BSR_3 denotes the BSRs of Sality, Kelihos and ZeroAccess bots respectively)

We evaluate the clustering performance in terms of the ability of producing well separated and compact bot clusters, for which we propose two performance criterion. We define *mix-clustered bots* as the bots mistakenly residing in the cluster of other types of bots, and the complement of which are called *separate-clustered bots*. The two performance criterion for evaluating the separation and compactness performances of bot clustering are: (1) *Bot Separation Ratio* (BSR), which is defined for each type of bot as the ratio of the number of separate-clustered bots and the overall number of bots of this type; (2) *Bot Compactness Ratio* (BCR), which is defined as the number of separate-clustered bots and the overall number of hosts (whether benign or not) assigned to the same cluster.

Bot Separation Performance: As observed from the per-hour clustering result[3] in Table. IV, Sality, Kelihos and ZeroAccess bots are assigned into different clusters with the cluster index shown in the table, i.e., all three types of bots are well separated from each other, which makes their BSRs achieve one. Moreover, almost all the bots of one type are grouped into the same cluster, with an exception of Sality bots who are divided into two clusters at the 12-th hour. Nevertheless, none of these clusters contains more than one type of bots, which demonstrates the perfect separation of different types of bots.

Bot Compactness Performance: The excellent bot separation performance indicates that the bot-included cluster completely excludes other types of bots. However, every bot-included clusters may still incorporate some benign P2P hosts, who accidentally display a similar traffic pattern during the detection period. BCR quantifies the clustering capability to exclude benign P2P bosts out of the bot-included cluster. BCR achieves one if the bot-included cluster contains zero benign P2P host. Accumulating the 24-hour BCR results, we plot BCR box-plot performance of three types of bots in Fig. 6. On average, Sality and ZeroAccess bot clusters falsely include 3 benign hosts respectively, while Kelihos bot clusters falsely include 12 benign hosts, which indicates that the clustering mechanism is not effective in generating compact bot clusters. Further inspection of the falsely included benign hosts shows they have traffic profiles that are highly similar to the bots' traffic. Based on the experimental results, we claim that network flow features are not sufficient to discriminate P2P bots from benign P2P hosts.

[3]Note that we only count the clusters containing more than one node.

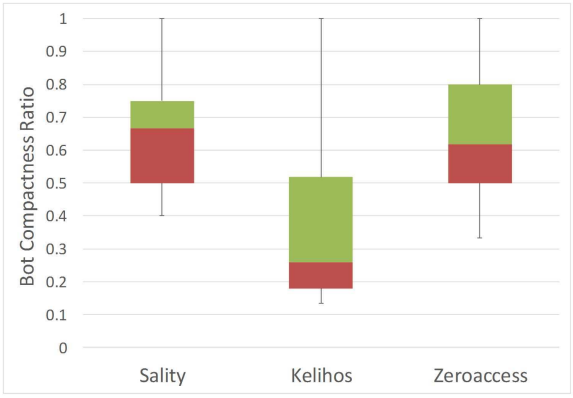


Fig. 6: Box plot of Bot Compactness Ratio

Parameter	Value
Preference Value	0.4
Damping Factor	0.9
Maxits	2000
Convits	200

TABLE V: Parameter setting for AP clustering

C. Identifying Bot-included Clusters via Classification

In this section, we evaluate the bot cluster identification using multi-class SVM classification with Gaussian kernel. Since we only have a limited number of labeled bots during every hour, we enlarge the training space by incorporating a half day of labeled bots and benign hosts into the training set. Consequently, the training set contains 36 clusters (3 clusters per hour) of labeled bots (with labels 'Sality', 'Kelihos', 'ZeroAccess') and 36 clusters of labeled legitimate P2P hosts running bittorrent, emule and skype (with labels 'Non-Bot'). We extract the training features from all the 72 labeled clusters to build the SVM classifier. Then, we use the next half day of bots and benign hosts as testing set, which includes a total of 37 bot-included clusters and 305 benign clusters.

After host clustering, DGBA detection process extracts four different types of group-level connection features from every cluster in the testing set. Then, the SVM model predicts the labels of clusters in the testing set. Since the classification module relies on four different types of features, we train the classifier on each individual group behavior feature in order to understand the importance of these features.

Table VI shows the classification performance using different types of features for training. The classification based on either shared neighbor feature or significant connection feature have high accuracy and recall, but only achieve moderate precision. Looking into the classification results, we find that the classification produces few false negatives but many false positives, i.e., bot-included clusters are unlikely to be regarded as benign cluster with these two features, but many benign clusters are falsely considered as bot-included clusters. On

Group Behavior Feature	Accuracy	Precision	Recall
Cluster Connectivity Feature	51.8%	7.9%	34.3%
Shared Neighbor Feature	92.7%	68.8%	91.7%
Significant Connection Feature	91.8%	66.7%	90%
Temporal Feature	71.3%	3.1%	66.7%
All Features	98.8%	94.6%	100%

TABLE VI: Classification accuracy when trained on one type of feature. Shared neighbor feature and significant connection feature present the best individual classification accuracy. The classifier achieves the best performance when combing all the features. Accuracy=(TP+TN)/all; Precision=TP/(TP+FP); Recall=TP/(TP+FN).

Bot Type	Bot Num.	Benign Host Num.	Correctly Identified	Falsely Identified
Sality	72	36	**69**(95.8%)	**0** (0%)
Kelihos	48	123	**47**(97.9%)	**5** (4.1%)
ZeroAccess	48	42	**48**(100%)	**2** (4.8%)

TABLE VII: Refined bot identification performance (the percentages in the parenthesis denote the bot detection rate and false alarm rate respectively)

the other hand, cluster connectivity feature seems unable to discriminate bots from benign hosts, which produces lots of false positives and false negatives. We observe that the correctly classified bots mainly belong to ZeroAccess botnet. Temporal feature is designed to be updated at the end of the day, thus is only used for bot classification in the final hour. Again, many false positives arise due to the inseparability of bots' features and benign hosts' features. However, the combination of all features provide the best result for detecting bot-included clusters. Overall, we only find two false positives with none false negatives.

D. Refined Bot Identification Performance

Bot-included clusters contain a considerable amount of benign hosts as shown in Section V-B, thus we use refined bot identification to extract the bots inside each bot-included cluster. Feature test in Algorithm 1 is utilized to perform refined bot identification. We run feature test on all the 39 bot-included clusters identified through SVM classification, including 13 Sality clusters, 12 Kelihos clusters, 12 ZeroAccess clusters and 2 false positives. The bot identification performance is depicted in Table. VII, which shows the bot number and benign host number in the bot-included clusters. In summary, the refined bot identification correctly identifies more than 95.8% of bots, and falsely triggers less than 4.8% alarms.

Bot Type	Bot Num.	Falsely Identified: Method A	Correctly Identified: Method B	Falsely Identified: Method B
Sality	72	36	51 (70.8%)	14 (38.9%)
Kelihos	48	123	38 (79.2%)	21 (17.1%)
ZeroAccess	48	42	33 (68.8%)	10 (23.8%)

TABLE VIII: Performance comparison with method A and method B (threshold: 0.2).

E. Performance Comparison with Other Detection Approaches

In this section, we present performance comparison of PeerClean system with other types of detection approaches: method A and method B. Method A relies on network flow statistical features to detect bot communications [15]. As shown in Section V-B, a lot of benign hosts appear to have similar flow traffic patterns like bots. We perform host clustering based on flow statistical features to separate different types of traffic flows. Then, the clusters containing bots are considered as bot clusters, and all hosts inside bot clusters are labeled as bots, i.e., all the benign hosts inside the bot clusters are falsely labeled as bots. The number of false positives using method A is listed in Table. VIII, which is unacceptably high.

Method B builds upon PeerClean system, but utilizes pairwise shared neighbor ratios to identify bots [9] instead of a set of group-level connection features. For each cluster generated from host clustering, we compute the pairwise shared neighbor ratio for every pair of hosts in the cluster. If the pairwise shared neighbor ratio is higher than an empirical threshold, we identify this pair of hosts as bots. From the results in Table. VIII, we find method B not only has a low accuracy of identifying bots, but is also more likely to produce lots of false alarms. The reason for the ineffectiveness of such bot identification is mainly due to the traffic dynamics, which causes method B to miss many bots but falsely include some benign hosts. On the other hand, PeerClean is able to significantly improve the identification accuracy by extracting and modeling group-level features instead of individual or pairwise features, which manifests itself by comparing Table. VIII with Table. VII.

F. System Scalability

The running time of PeerClean system depends on the clustering method, the group feature extraction and the SVM classification method. The SVM classification method intends to classify a small number of meaningful clusters. The running time can be negligible. On the other hand, group feature extraction tries to collect the group behavior features in the whole cluster. If one cluster contains a large number of P2P hosts, the group behavior feature extraction will consume a considerable amount of time. In the case of large clusters, instead of extracting all the group features, we can only extract one single feature to tradeoff the complexity and performance.

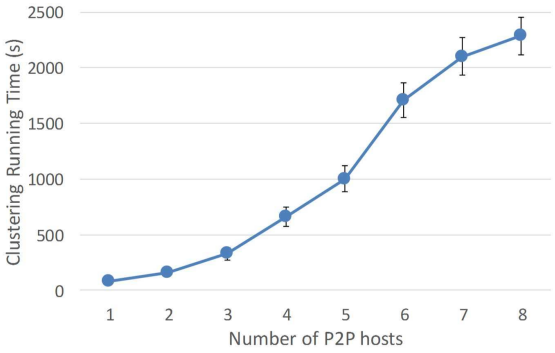


Fig. 7: Running time of AP clustering

Based on the results in Table VI, we can extract only the shared neighbor feature or significant connection feature for large clusters without degrading too much performance, which will greatly reduce the computational complexity.

Finally, we evaluate the time consumption of AP clustering algorithm. We use a commodity computer system with AMD FX(tm)-8120 processor and 16GB memory. The running time corresponding to different number of P2P hosts are shown in Fig. 7. The time consumption for a large amount of P2P hosts (up to 8000) is still constrained in one hour period. Since PeerClean works within a one-hour period, the time consumption is acceptable. Alternatively, leveraged affinity propagation method [20] can be used to deal with large data sets, and we leave it for future work.

VI. Related Work

The increasing popularity of P2P botnets have led to a vast amount of research that attempt to track and remove them. In those work, the detection mechanisms can be classified into two categories: host-based approaches and network-based approaches. The second category can be subdivided into network traffic-based approaches and communication graph-based approaches. We now review some additional works from the second category that we have not discussed yet.

Network traffic-based approaches: Some related work utilized attack traffic characteristics to identify hosts with similar abnormal network behaviors, such as spamming, port scanning, sharing the same packet contents [25], or, having common destinations, similar payloads and common host platforms [26]. However, these approaches can be evaded by manipulating attacking strategies, such as using social engineering as an infection vector instead of scanning.

Several work focused on identifying C&C traffic from the botnets. Bilge et al. [15] proposed to use NetFlow analysis to distinguish botnet C&C servers from benign servers by extracting flow-level features from the data. Wurzinger et al. [27] identify C&C by automatically extracting signatures from bot responses after receiving commands. However, this approach is hindered by traffic encryption. Moreover, the above approaches, which use only flow-level statistics, are not robust enough to produce accurate detection results. Instead, PeerClean greatly enhances the detection capability by jointly considering the flow-level traffic statistics and network connection behaviors.

Communication graph-based approaches: In [28], Coskun et al. proposed to identify the local members of P2P bots using mutual contacts graph. However, this method requires to start with a captured seed bot in the network, which may not be available. [29] attempted to identify spamming bots using large scale graph analysis by looking for tightly connected subgraph components. Jelasity et al. [30] argued that it is difficult to detect P2P bots using traffic dispersion graph (TDG) especially with a limited view of the Internet traffic at a single AS. Most recently, Li et al. [31] proposed to detect P2P community by identifying the densely connected subgraphs. However, this approach only focused on a backbone network which requires a very large communication graph. Also, solely relying on the connection patterns, it may falsely include lots of benign hosts in the discovered P2P botnets.

P2P botnet resilience: Several related work studied the resilience of P2P botnets. [32] provided a overview of bot structures, which shows the P2P bots with random graph networks are highly resilient to both random and targeted defense mechanisms. Most recently, [16] investigated an important characteristic of botnets: assortativity, and showed its impacts to the network resilience and recovery. [6] provided formal models to systemize the attacking strategies against P2P botnets. However, these takedown attempts require reverse engineering of the bot binaries to crawl or sinkhole the whole botnets, which prevents the wide applicability of these approaches.

VII. Conclusion

P2P C&C infrastructure has become a popular choice for the future botnets, which is extremely resilient to even sophisticated takedown attempts. The ability to identify botnets inside a network is particularly important to the network administrators. Toward this direction, we present PeerClean, a new network flow-based system to identify and classify botnets with a high accuracy. PeerClean leverages a novel dynamic group behavior analysis to extract and model a set of robust and reliable connection features at the group level. In our experimental evaluation on real-world network traces, PeerClean shows excellent detection accuracy on various types of botnets. An interesting future direction is to apply the group behavior analysis to other types of applications to help identify the network behaviors which would be otherwise unnoticeable. We can also further merge and correlate the group behaviors of different applications to detect anomalous communication patterns and identify newly emerging attacks.

References

[1] Damballa, "Peer-to-peer: A growing tactic used for threat command-and-control," https://www.damballa.com/downloads/r_pubs/wp_peer-to-peer_a_growing_tactic.pdf, 2013.

[2] "Latest kelihos botnet shut down live at rsa conference 2013," http://threatpost.com/.

[3] N. Falliere, "Sality: Story of a peer to-peer viral network," http://www.symantec.com/connect/downloads/whitepaper-sality-story-peer-peer-viral-network, July 2011.
[4] J. Wyke, "The zeroaccess botnet - mining and fraud for massive financial gain," Sep. 2012.
[5] T. Werner, "Kelihos.c: Same code, new botnet, 2012," http://www.crowdstrike.com/blog/kelihosc-same-code-new-botnet/index.html, Mar. 2012.
[6] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "P2PWNED: Modeling and evaluating the resilience of peer-to-peer botnets," in *IEEE Symposium on Security & Privacy*, May 2013.
[7] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "Botgrep: Finding p2p bots with structured graph analysis," in *Proc. of USENIX Security'10*, 2010.
[8] T.-F. Yen and M. K. Reiter, "Are your hosts trading or plotting? telling p2p file-sharing and bots apart," in *Proc. of ICDCS*, June 2010.
[9] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy p2p botnets using statistical traffic fingerprints," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, June 2011.
[10] Z. Xu, L. Chen, G. Gu, and C. Kruegel, "Peerpress: Utilizing enemies' p2p strength against them," in *Proc. of ACM CCS'12*, October 2012.
[11] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proc. of USENIX Security'09*, August 2009.
[12] G. Gu, P. Porras, V. Yegneswaran, and M. Fong, "Bothunter: Detecting malware infection through IDS-driven dialog correlation," in *Proc. of USENIX Security'07*, August 2007.
[13] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proc. of USENIX Security'08*, 2008.
[14] 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. of AsiaCCS*, March 2011.
[15] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "DISCLOSURE: Detecting botnet command and control servers through large-scale netflow analysis," in *Proc. of ACSAC*, Dec. 2012.
[16] T.-F. Yen and M. K. Reiter, "Revisiting botnet models and their implications for takedown strategies," in *Proceedings of the First international conference on Principles of Security and Trust*, 2012.
[17] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *Proc. of WWW'04*, 2004, pp. 512–521.
[18] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, October 2006.
[19] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., 2005.
[20] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *Science*, vol. 315, no. 5814, pp. 972–976, 2007.
[21] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.
[22] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic: Behavior models and applications," in *Proc. of SIGCOMM*, August 2005.
[23] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
[24] "Autoit script," http://www.autoitscript.com/site/autoit/.
[25] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.
[26] T.-F. Yen and M. K. Reiter, "Traffic aggregation for malware detection," in *Proc. of DIMVA '08*, 2008.
[27] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, "Automatically generating models for botnet detection," in *Proc. of ESORICS'09*, 2009, pp. 232–249.
[28] B. Coskun, S. Dietrich, and N. Memon, "Friends of an enemy: Identifying local membersof peer-to-peer botnets using mutual contacts," in *Proc. of ACSAC*, 2010.
[29] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum, "Botgraph: large scale spamming botnet detection," in *Proc. of NSDI'09*, 2009.
[30] M. Jelasity and V. Bilicki, "Towards automated detection of peer-to-peer botnets: on the limits of local approaches," in *Proc. of LEET'09*, 2009.
[31] L. Li, S. Mathur, and B. Coskun, "Gangs of the internet: Towards automatic discovery of peer-to-peer communities in the internet," in *Proc. of CNS*, 2013, pp. 167–175.
[32] D. Dagon, G. Gu, C. Lee, and W. Lee, "A taxonomy of botnet structures," in *Proc. of ACSAC'07*, 2007.

APPENDIX

Background of Entropy-based Approach: In the information theory, *entropy* quantifies the amount of uncertainty contained in the data. Given a random variable R that may take n_R discrete values, suppose we sample R for n times, we observe R takes the value of r_i for n_i times, which will produce a probability distribution for R, i.e., $P(r_i) = n_i/n, r_i \in R$. The entropy of R is defined as:

$$H(R) := -\sum_{r_i \in R} P(r_i) \log P(r_i),$$

where $H(R)$ is a function of support size n_R and sample size n. To eliminate the dependency on support and sample sizes, *standardized entropy* is proposed to provide a robust quantifier for data variety or uniformity [22]:

$$H_s(R) := \frac{H(R)}{\log \min\{n_R, n\}},$$

where $H_s(R) = 0$ means that all the observations of R have the same value, thus no data variety exists, while $H_s(R) = 1$ means all the observed values of R are different. Note that in this paper, $n \ll n_R$, such that $H_s(R) = H(R)/\log n$.

Let C be the set of all the observed values on R, then $H_s(R) = 1$ satisfies if and only if $|C| = n$ and $P(r_i) = 1/n$ for $r_i \in C$, which means the observed values are uniformly distributed over C. Thus, $H_s(R)$ provides a measure for the data uniformity in the observed value set C. Similar to conditional entropy, *conditional standardized entropy* $H_s(R|C)$ is derived by conditioning R based on C. As $H(R|C) = H(R)$, we have $H_s(R|C) = H(R)/log|C|$. Hence, when $H_s(R|C)$ getting close to 1, the observed values are more uniformly distributed over the set C, or less distinguishable from one another. On the other hand, $H_s(R|C) \ll 1$ depicts that some values are more frequently observed. This measure is used to define SCs as follows.

Significant Connection Feature Extraction: SC feature is quantified by the conditional standardized entropy, and extracted from the traffic flows of every host. Suppose a random variable R denotes the connections made by a host in the cluster. Given a detection period E, let N be the total number of flows observed from/towards the host, and $C = \{c_1, c_2, \ldots, c_m\}$, $m \geq 2$ be the set of distinct connections in R that the observed flows take. Then the induced probability distribution $\mathcal{P}_C$ on R is given by $\mathcal{P}_C(c_i) = N_i/N$, where N_i is the number of flows from the connection c_i. Based on $\mathcal{P}_C$, the conditional standardized entropy, $H_s(\mathcal{P}_C) := H_s(R|C)$, measures the degree of uniformity in the observed features C. In particular, with $H_s(\mathcal{P}_C)$ getting close to 1, the observed values in C are close to being uniformly distributed, or indistinguishable from one another. Otherwise, some values in C "jump out" from the rest. To extract SCs, we employ a dynamic threshold

algorithm [22] to separate C into two sets: the SC set C_s and the indistinguishable connection set C_i. In this algorithm, C_s becomes the SC set under two conditions: (1). if C_s is the smallest subset of C such that the probability of any value in C_s is larger than that of the remaining values; (2). if the values in C_i are nearly uniformly distributed, i.e., $H_s(\mathcal{P}_{C_i}) > \lambda$, where λ is the significance threshold that is close to one (e.g., 0.9).

We first sort the connections in C in a decreasing order of their probabilities $\mathcal{P}_C$ as $C = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_m\}$. Then, the dynamic threshold algorithm will find $C_s = \{\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k\}$, $C_i = \{\hat{c}_{k+1}, \ldots, \hat{c}_m\}$, where k is the smallest integer such that $H_s(\mathcal{P}_{C_i}) > \lambda$. Thus, $c^* = \hat{c}_k$ is called the *cut-off threshold* such that the probability distribution of the remaining connections almost becomes uniformly distributed. Curious readers please refer to [22] for a complete algorithm design. Finally, the SC feature can be derived directly according to the number of elements in the SC set C_s, which corresponds to the number of SCs of the host.