Enhanced PeerHunter: Detecting Peer-to-peer Botnets through Network-Flow Level Community Behavior Analysis

Di Zhuang, Student Member, IEEE, and J. Morris Chang, Senior Member, IEEE

Abstract—Peer-to-peer (P2P) botnets have become one of the major threats in network security for serving as the fundamental infrastructure for various cyber-crimes. More challenges are involved in the problem of detecting P2P botnets, despite a few works claimed to detect centralized botnets effectively. We propose Enhanced PeerHunter, a network-flow level community behavior analysis based system, to detect P2P botnets. Our system starts from a P2P network flow detection component. Then, it uses “mutual contacts” to cluster bots into communities. Finally, it uses network-flow level community behavior analysis to detect potential botnets. In the experimental evaluation, we propose two evasion attacks, where we assume the adversaries know our techniques in advance and attempt to evade our system by making the P2P bots mimic the behavior of legitimate P2P applications. Our results showed that Enhanced PeerHunter can obtain high detection rate with few false positives, and high robustness against the proposed attacks.

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

I. INTRODUCTION

A botnet is a set of compromised machines controlled by botmasters through control and command (C&C) channels. Botnets may have different communication architectures. Traditional botnets are known to use centralized architectures, which have potential single point of failure. Peer-to-peer (P2P) network is modeled as a distributed architecture, where even if a certain number of peers do not function properly, the whole network is not compromised. Most of the recent botnets (e.g., Storm, Waledac and ZeroAccess) attempted to use P2P architectures, and P2P botnets were proved to be highly resilient even after a certain number of bots being identified or taken down [1]. P2P botnets provide a fundamental infrastructure for various cyber-crimes, such as distributed denial-of-service (DDoS), email spam, click fraud, etc. For instance, recent botnet attacks including those carried out by WhiskeyAlfa (responsible for Sony Pictures Entertainment attack) and WannaCry (responsible for ransoming healthcare facilities in Europe) showed the scale and scope of damage that P2P botnets can cause. As such, detecting P2P botnets effectively is rather important for securing cyberspace.

Designing an effective P2P botnets detection systems is very challenging. First, botnets tend to act stealthily [2] and spend most of their time in the waiting stage before performing any malicious activities [3]. Approaches using malicious activities would have small window of opportunities to detect such botnets. Second, botnets tend to encrypt the C&C channels, causing deep-packet-inspection (DPI) based methods ineffective. Third, the role of a single bot can be changed dynamically depending on the current structure of a botnet [4] (e.g., P2P bot can shift its functionality to act as a botmaster when the prior botmaster has been taken down). Hence, it is difficult to characterize a botnet just by looking at a single bot.

In this work, we present Enhanced PeerHunter, an extension of PeerHunter [5], aiming to use network-flow level community behaviors to detect waiting stage P2P botnets, even in the scenario that P2P bots and legitimate P2P applications are running on the same set of hosts. We consider a botnet community as a group of compromised machines that communicate with each other or connect to the same set of botmasters through the same C&C channel, are controlled by the same attacker, and aim to perform similar malicious activities. In the “waiting stage”, no malicious activities could be observed. As discussed in [4], the dynamic change of communication behaviors of P2P botnets makes it extremely hard to identify a single bot. Nonetheless, bots belonging to the same P2P botnet always operate together as a community and share the same set of community behaviors. Our system starts from a P2P network flow detection component, and builds a network-flow level mutual contacts graph (MCG) depending on the mutual contacts characteristics [6] between each pair of the P2P network flows. Afterwards, it employs a community detection component to cluster the same type of bots into the same community, and separate bots and legitimate applications or different types of bots into different communities. Finally, our system uses the destination diversity (the “P2P behavior”) and the mutual contacts (the “botnet behavior”) as the natural behaviors to detect P2P botnet communities.

In the experiments, we mixed a background network dataset [7] with 5 P2P botnets datasets and 4 legitimate P2P applications datasets [8]. To make our experimental evaluation as unbiased and challenging as possible, we propose a network traces sampling and mixing method to generate synthetic experimental datasets. To be specific, we evaluated our system with 100 synthetic experimental datasets that each contains 10,000 internal hosts. We implemented our P2P network flow detection component using MapReduce framework, which dramatically reduced the number of hosts subject to analysis by 99.03% and retained most of the P2P hosts. Also, the MapReduce design and implementation of our system could be deployed on cloud-computing platforms (e.g., Amazon EC2),

The authors are with the Department of Electrical Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: dizhuang@mail.usf.edu; chang5@usf.edu).
which ensures the scalability of our system (i.e., processing an average of 97 million network flows in about 20 minutes). To summarize, our work has the following contributions:

- We present a novel, effective and efficient network-flow level community behavior analysis based system, Enhanced PeerHunter, which is capable of detecting P2P botnets when (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) the botnet traffic are overlapped with legitimate P2P traffic on the same host; and (d) none statistical traffic patterns are known in advance (unsupervised).

- We experimented our system using a wide range of parameter settings. With the best parameter settings, our system achieved 100% detection rate with zero false positive.

- We propose two evasion attacks (i.e., passive and active mimicking legitimate P2P application attacks), where we assume the adversaries know our techniques in advance and attempt to evade our system via instructing P2P bots to mimic the behavior of legitimate P2P applications. The experiment results showed that our system is robust to both attacks.

- We compared Enhanced PeerHunter with PeerHunter [5] (i.e., our previous work) and Zhang et al. [2]. Extensive experiments were conducted to show that (a) our system outperforms Zhang et al. [2] in terms of the detection rate of different botnets, the overall precision, recall and false positives, and (b) our system is more robust to MMKL attacks compared with PeerHunter [5] and Zhang et al. [2].

The rest of this paper is organized as follows: Section II presents the related work. Section III explains the motivation and details of the features applied in our system. Section IV describes the system design and implementation details. Section V presents the experimental evaluation. Section VI discusses the evasions and possible solutions, deployment and the potential extensions of our system. Section VII concludes.

II. RELATED WORK

To date, a few methods attempting to detect P2P botnets were proposed [2]–[6], [8]–[14]. From the data perspective, recent approaches can be divided into two categories [14]: payload-based and flow-based. Payload-based systems [9], [15], [16] use payload content and header information of network packets to detect botnets. For instance, BotHunter [9] is a well-known packet inspecting bot detection system that relies on a modified Snort [17] (i.e., a rule-based intrusion detection system that requires the access to the full payload) to detect potential malicious activities and further identify infected hosts. Lu et al. [15] proposed to use decision tree models trained on the n-gram features extracted from the network traffic payload to detect botnets. Wang et al. [16] proposed to use lexical features of HTTP header (TCP payload) to discover malicious behaviors of Android botnets.

Flow-based systems [2]–[6], [8], [10]–[14], [18], [19] use header information of network packets (i.e., network flow characteristics) to capture botnets behaviors. Compared with payload-based systems, flow-based systems use less information from the network packets. Since recent botnets tend to use encryption to hide their payload information from the detection systems, most of the packet-based systems that applying deep packet inspection (DPI) on the payload information (e.g., BotHunter [9]) will be foiled. Zhang et al. [20] proposed to add a high-entropy flow detector into BotHunter to detect bots, when part of the packets payloads of botnets’ network flows are encrypted. Their assumption is that the presence of high-entropy flows (detected from the encrypted packets payloads) together with existing botnets events (detected from the non-encrypted packets payloads by BotHunter) could identify botnets using encrypted network traffic. However, if all the packets payloads are encrypted [14], it will be hard for their approach to perform. The flow-based detection systems have advantage over the packet-based systems that applying deep packet inspection (DPI) on the payload information (e.g., BotHunter [9]) given that they can be applied to encrypted traffic. Some flow-based systems applied one or several different supervised machine learning algorithms on a set of well-extracted network flow features to model the botnets behaviors. For instance, Jianguo et al. [21] applied three supervised machine learning algorithms (i.e., SVM, Logistic Regression and Neural Network) on network flow features extracted from Netmate and Tranalyzer to detect botnets. They obtained very high performance metrics, while employing a fully labelled dataset. Khanchi et al. [19] proposed an approach using genetic programming and ML on data streams to detect botnets flows. However, since most of the supervised ML-based approaches usually generate models that are focusing on specific types of botnets (existing in the training data), those approaches will not be effective to detect botnets not appeared in the training data (unknown botnets).

Some flow-based systems utilized a combination of different heuristics to model P2P botnets behaviors. For instance, Botgrep [10] proposed to detect P2P botnets through localizing structured communication graphs, where they found that the communication graph of P2P applications have fast convergence time of random walks to a stationary distribution. However, their method can only identify structured communication subgraphs, rather than ensure those subgraphs containing P2P botnets. Entelecheia [8] proposed to use a synergistic graph-mining approach on a super-flow graph built from network flow features (i.e., volume per hour, duration per flow) to identify a group of P2P bots, where they claimed that P2P botnet network flow tend to have low volume and long duration. Group or community behavior based methods [4]–[6], [11] considered the behavior patterns of a group of bots within the same P2P botnet community. Coskun et al. [6] developed a P2P botnets detection approach that started from clustering P2P hosts into groups, and then used supervised machine learning methods (e.g., SVM) to identify bots through a set of group-level behavior features. Since their
approach relied on supervised classification methods (e.g., SVM) which required to train the model of each botnet on fully labelled dataset in advance, it would be hard for their method to detect unknown botnets. Chen et al. [23] applied three unsupervised machine learning algorithms (i.e., self-organising map, local outlier factor and k-NN outlier) to build a normal behavior profile to detect botnet. They obtained a very high detection rate (91.3%), but with inherited high false positive rates due to the nature of the unsupervised ML algorithms employed. PeerHunter [5], our previous work, proposed to use the host level community behavior analysis to detect P2P botnets, which did not consider the scenario that P2P bots and legitimate P2P applications could run on the same set of hosts. Zhang et al. [2] proposed a scalable botnet detection system capable of detecting stealthy P2P botnets (i.e., in the waiting stage), where no knowledge of existing malicious behavior was required in advance. They also claimed to work in the scenario that the botnet traffic are overlapped with the legitimate P2P traffic on the same host. However, their experimental dataset was slightly biased and less challenging. For example, in their dataset, the number of bots was twice as many as the number of legitimate P2P hosts, which was much easier for bots to form clusters than legitimate P2P hosts.

In this work, we present Enhanced PeerHunter, a network-level flow-based system that relies on community behavior analysis to detect P2P botnets. We compared Enhanced PeerHunter with PeerHunter [5] and Zhang et al. [2] on a more challenging and comprehensive experimental datasets, and showed that our system outperforms both systems in terms of detection rate, false positives and the performance under the proposed mimicking legitimate P2P application attacks.

III. BACKGROUND AND MOTIVATION

In this section, we investigate the characteristics being used to detect P2P network traffic, and introduce the concept of “mutual contacts”, which motivated us to formulate the P2P botnet detection problem as a network community detection problem. Also, we explore the P2P botnet community behaviors being used to identify botnets communities. To demonstrate the features discussed in this section, we conducted some preliminary experiments using the dataset shown in Table I and Table IV. Table I shows the notations and descriptions, and Table II shows the measurements of features.

### Table I: Notations and Descriptions

| Notations | Descriptions |
|-----------|--------------|
| MNF       | the management network flow |
| AVGDD     | the average # of distinct /16 MNF dstIP prefixes |
| AVGDDR    | the average destination diversity ratio |
| AVGMC     | the average mutual contacts ratio |
| Θdd       | the threshold of destination diversity |
| Θmcr      | the threshold of mutual contacts ratio |
| Θavgdd    | the threshold of AVGDDR |
| Θavgmcr   | the threshold of AVGMCR |
| BSI       | Bot Separation Index |
| BAI       | Bot Aggregation Index |
| BLSI      | Bot-Legitimate Separation Index |

### Table II: Measurements of Features

| Trace   | AVGDD | AVGDDR | AVGMC | AVGMCR |
|---------|-------|--------|-------|--------|
| eMule   | 8,349 | 17.6%  | 5,380 | 3.7%   |
| FrostWire | 11,420 | 15.2%  | 7,134 | 4.5%   |
| uTorrent | 17,160 | 8.7%   | 13,888 | 3.5%  |
| Vuze    | 12,983 | 10.1%  | 18,850 | 7.9%   |
| Storm   | 7,760  | 25.1%  | 14,684 | 30.2%  |
| Waledac | 6,038  | 46.6%  | 7,099  | 57.0%  |
| Salty   | 9,803  | 9.5%   | 72,495 | 53.2%  |
| Kelihos | 305    | 97.4%  | 310    | 98.2%  |
| ZeroAccess | 246    | 96.9%  | 254    | 100.0% |

A. P2P Network Characteristics

Due to the nature of P2P networks, P2P hosts usually communicate with their peers through IP addresses directly, without any queries from DNS services [24], namely, non-DNS connections (NoDNS). Also, peer churn is another typical behavior in P2P networks [25], which results in a significant number of failed connections in P2P network flow. Furthermore, due to the decentralized nature of P2P network, a P2P host usually communicates with peers distributed in a large range of physical networks, which results in destination diversity (DD) [8] of P2P management network flow (MNF). To be clearer, P2P host generate two types of network flow: (1) management network flow, which maintains the function and structure of the P2P network, and (2) other network flow, such as data-transfer flow, which does not necessarily have the P2P network characteristics. The P2P network flow mentioned in this section and the rest all refers to P2P MNF.

Zhang et al. [2] proposed to remove a decent number of non-P2P network flow using NoDNS, and then performed a fine-grained P2P hosts detection using DD. Based on their experiment results, DD plays a much more important role in detecting P2P hosts than NoDNS. Therefore, in this work, we decided to only use DD to simplify and speed up the P2P network flow detection procedure. In addition, we used the number of distinct /16 IP prefixes of each host’s network flow, rather than BGP prefix used in [2] to approximate DD, since /16 IP prefix is a good approximation of network boundaries. For instance, it is very likely that two IP addresses with different /16 IP prefixes belong to two distinct physical networks. This is also supported by Table I which shows the network flow in a P2P network spreading across many distinct physical networks according to the number of /16 IP prefixes.

B. Mutual Contacts

The mutual contacts (MC) between a pair of hosts is a set of shared contacts between them [6]. Consider the network illustrated in Fig. 1 which contains an internal network (A, B, C, D and E) and an external network (1, 2, 3, 4 and 5). A link between a pair of hosts means communication between them. In Fig. 1, 1, 2 are the mutual contacts shared by A, B.

Mutual contacts are the natural characteristic of P2P botnet. Compared with legitimate hosts, a pair of bots within the same P2P botnet has higher probability to share mutual contacts [6]. Because bots within the same P2P botnet tend to receive the same C&C messages from the same set of botmasters [26]. Moreover, in order to prevent bots (peers) from churning,
of a P2P host is the number of distinct /16 IP prefixes of its network flows’ destination IPs. The destination diversity ratio (DDR) of each host is its DD divided by the total number of distinct destination IPs of its network flows. Due to the decentralized nature of P2P networks, P2P network flow tend to have higher DDR than non-P2P network flow. Furthermore, network flow from P2P botnets usually have higher AVGDDR than network flow from legitimate networks. Network flow from bots within the same botnet tend to have similar DDR, since those bots are usually controlled by machines, rather than humans. However, the destinations of legitimate P2P network flow are usually user-dependent, which result in their DDR varying greatly from user to user. Besides, our approach aims to cluster bots within the same botnets together, rather than attempting to cluster the legitimate hosts. Therefore, legitimate communities might contain both P2P hosts and non-P2P hosts, leading to lower AVGDDR. As shown in Table II, both legitimate hosts and bots spread across a wide range of distinct networks. However, most of the botnets have higher AVGDDR than legitimate applications, except Sality.

**Average Mutual Contacts Ratio:** This captures the “botnet behavior” of P2P botnets. The mutual contacts ratio (MCR) between a pair of hosts is the number of mutual contacts between them, divided by the number of total distinct contacts of them. This is based on three observations: (a) P2P botnets are usually formed by at least two bots, otherwise they cannot act as a group, (b) the MCR of a pair of bots within the same botnet is much higher than the MCR of a pair of legitimate applications or a pair of bots from different botnets, and (c) each pair of bots within the same botnet has similar MCR. Thus, we define AVGMCR as the average MCR among all pairs of hosts within one network community. As shown in Table II, both botnets and certain legitimate network communities have a considerable number of mutual contacts. That is because those legitimate communities have much more “base” contacts than botnets. However, botnets have much higher AVGMCR.

3) **Structural Community Feature:** This captures the structural characteristics of a botnet. As discussed above, every pair of bots within the same botnet tends to have a considerable number or ratio of mutual contacts. If we consider each host as a vertex and link an edge between a pair of hosts when they have mutual contacts, the bots within the same botnet tend to form cliques. On the contrary, the contacts of different legitimate hosts usually diverge into different physical networks. Thus, the probability that legitimate communities form certain cliques is relatively low. Then, we can consider P2P botnets detection as a clique detection problem, which detects cliques from a given network with certain requirements. However, since clique detection problem is NP-complete, we cannot directly apply such method to detect botnets, without any preprocessing. We propose to combine all three botnet community behaviors, and use the previous two community behaviors as the “preprocessing” of the clique detection problem.

**IV. SYSTEM DESIGN**

Enhanced PeerHunter has three components, as shown in Fig. 2 that work synergistically to (a) detect P2P network flow, (b) distinguish community behaviors. We consider three types of community behaviors: (a) flow statistical feature, (b) numerical community feature and (c) structural community feature.

**1) Flow Statistical Feature:** Botnet detection methods using flow statistical features, have been widely discussed [2]–[5]. For the MNFs of a specific P2P application, most of its statistical patterns depend on its P2P network protocol. However, the statistical patterns of other network flows, such as data-transfer flow, are usually situation-dependent, which vary a lot even in the same P2P network. In this work, we use the ingoing and outgoing bytes-per-packets (BPP) of MNFs in one P2P network as its community flow statistical feature.

**2) Numerical Community Feature:** We consider two numerical community features: average destination diversity ratio (AVGDDR) and average mutual contacts ratio (AVGMCR).

**Average Destination Diversity Ratio:** This captures the “P2P behavior” of P2P botnets. The destination diversity (DD)
(b) construct the network-flow level mutual contacts graph, and
c(c) detect P2P botnets.

A. P2P Network Flow Detection

This component aims to detect network flow that engage in P2P communications using the features described in Section III-A. The input is a set of 5-tuple network flow \([ip_{src}, ip_{dst}, proto, bpp_{out}, bpp_{in}]\), where \(ip_{src}\) is the source IP, \(ip_{dst}\) is the destination IP, \(proto\) is either tcp or udp, and \(bpp_{out}\) and \(bpp_{in}\) are outgoing and ingoing bytes-per-packets (BPP) statistics. First, we group all network flows \(F = \{f_1, f_2, \ldots, f_k\}\) into flow clusters \(FC = \{FC_1, FC_2, \ldots, FC_m\}\) using the 4-tuple \([ip_{src}, proto, bpp_{out}, bpp_{in}]\). Then, we calculate the number of distinct /16 prefixes of \(ip_{dst}\) (destination diversity) associated with each flow cluster, \(dd_i = DD(FC_i)\). If \(dd_i\) is greater than a pre-defined threshold \(\Theta_{dd}\), we consider \(FC_i\) as a P2P MNF cluster, and its source hosts as P2P hosts. We retain all the network flows within the P2P MNF clusters for the next component, and eliminate all the other network flows. As shown in Algorithm 1 we designed this component using a MapReduce framework [27]. For a mapper, the input is a set of 5-tuple network flow, and the output is a set of key-value pairs, where the key is the 4-tuple \([ip_{src}, proto, bpp_{out}, bpp_{in}]\) and the value is its corresponding \(ip_{dst}\). For a reducer, the input is the set of key-values pairs that outputs by the mapper. Then, the reducer aggregates all values with the same key to calculate the DD of each flow cluster, and finally output the detected P2P MNF based on \(\Theta_{dd}\).

Algorithm 1 P2P Network Flow Detection

1: function Map([ip_{src}, ip_{dst}, proto, bpp_{out}, bpp_{in}])
2: Key \leftarrow [ip_{src}, proto, bpp_{out}, bpp_{in}]
3: Value \leftarrow [dd]
4: output (Key, Value)
5: end function
6: function Reduce(Key, Value] )
7: \(k \leftarrow Key\)
8: \(dd_k = \emptyset\)
9: for \(v \in Value[ ]\) do
10: \(dd_k \leftarrow dd_k \cup \{v\}\)
11: end for
12: if \(|dd_k| \geq \Theta_{dd}\) then
13: for \(v \in Value[ ]\) do
14: output \((k, v)\)
15: end for
16: end if
17: end function

B. Network-Flow Level Mutual Contacts Graph Extraction

This component aims to extract mutual contacts graph (MCG) using the network-flow level mutual contacts. We call a pair of P2P network flow clusters the same type, if they have the same 3-tuple \([proto, bpp_{out}, bpp_{in}]\). As illustrated in Fig. 3 each host might contain one type or several different types of P2P network flow clusters generated by either P2P botnets or legitimate P2P applications running on it. If a pair of the same type of P2P network flow clusters generated by different hosts, have at least one (network-flow level) mutual contacts, we create an edge between them in the corresponding network-flow level MCG.

To be specific, the input is a set of P2P network flow clusters \(FC = \{FC_1, FC_2, \ldots, FC_m\}\) and their corresponding P2P network flows, \(F = \{f_1, f_2, \ldots, f_{n_1}, f_2, \ldots, f_{n_2}, \ldots, f_{n_3}, \ldots, f_{n_m}\}\), such that \(f_i^{FC_j}\), \(f_i^{FC_j}\), \(f_i^{mcr}\), \(f_i^{16}\) denotes the flow \(i\) of \(FC_j\). The output is a MCG, \(G_{mc} = (V, E)\), where each vertex \(v_i \in V\) represents network flow cluster \(FC_i\) and has a DDR score \(ddr_i\), and each edge \(e_{ij} \in E\) represents the existence of mutual contacts between \(FC_i\) and \(FC_j\) with nonnegative MCR weight \(mcr_{ij}\). Algorithm 2 shows the detailed steps.

First, for each P2P network flow cluster \(FC_i\), we generate a contact set \(C_i\), that contains all the destination IPs of its network flows. Each P2P network flow cluster \(FC_i\) also contains a flow statistical pattern set \(S_i\), which contains all the 3-tuple \([proto, bpp_{out}, bpp_{in}]\) of its network flows. Let \(DD(C_i)\) be the set of distinct /16 prefixes of all the IPs in \(C_i\). Then, \(ddr_i\) and \(mcr_{ij}\) can be calculated as follows.

\[
ddr_i = \frac{||DD(C_i)||}{||C_i||} \quad mcr_{ij} = \frac{||C_i \cap C_j||}{||C_i \cup C_j||}
\]

Furthermore, as discussed in Section III-C1 the network flows from different hosts (or network flow clusters) within the same network communities (generated by the same type of P2P botnet or legitimate P2P application) should have similar statistical patterns. Thus, for each pair of input P2P network flow clusters, say \(FC_i\) and \(FC_j\), we calculate the intersection between \(S_i\) and \(S_j\). If \(S_i \cap S_j = \emptyset\), then there should be no edge between \(FC_i\) and \(FC_j\) in MCG. Otherwise, they share at least one network flow statistical pattern, and we calculate \(mcr_{ij}\) as shown in equation (1). Let \(\Theta_{mcr}\) be a pre-defined threshold. Then, if \(mcr_{ij} > \Theta_{mcr}\), there is an edge between \(FC_i\) and \(FC_j\), with weight \(mcr_{ij}\). Otherwise, there is no edge between \(FC_i\) and \(FC_j\) (i.e., \(mcr_{ij} = 0\)).

C. P2P Botnet Detection

This component aims to detect P2P bots from MCG. First, we cluster the bots and the other hosts into their own communities using a community detection method. Afterwards, we detect botnet communities using numerical com-
Fig. 3: An example of network-flow level mutual contacts graph extraction and community detection. Each triangle represents a network flow cluster, and the same color triangles represent the same type of network flow clusters. The areas separated by the dash-dot line with different color represents different communities.

**Algorithm 2 Network-Flow Level MCG Extraction**

```
input: FC, F, Θ_mcr
output: G_mc = (V, E)
1: E = ∅, V = ∅
2: for FC_i ∈ FC do
3:   C_i = ∅
4:   S_i = ∅
5: end for
6: for f_j ∈ F do
7:   C_j ← C_j ∪ {ip_dst}
8:   S_j ← S_j ∪ {proto, bpp_out, bpp_in}
9: end for
10: for FC_i ∈ FC do
11:   avgddr_i ← \|DD(C_i)\| \|C_i\|^{-1}
12:   vertex v_i ← avgddr_i
13:   V ← V ∪ {v_i}
14: end for
15: for ∀ FC_i, FC_j ∈ FC and i < j do
16:   if S_i ∩ S_j ≠ ∅ then
17:     mcr_ij ← \|C_i ∩ C_j\| \|mcr\|^{-1}
18:     if mcr_ij > Θ_mcr then
19:       edge e_ij ← mcr_ij
20:       E ← E ∪ {e_ij}
21: end if
22: end if
23: end for
24: return G_mc = (V, E)
```

Community behavior analysis. Finally, we use structural community behavior analysis to further identify or verify each bot candidate. Algorithm 3 shows the detailed steps.

1) **Community Detection**: Given MCG \( G_{mc} = (V, E) \), \( ∀ e_{ij} ∈ E \), we have \( mcr_{ij} ∈ [0.0, 1.0] \), where \( mcr_{ij} = 1.0 \) means all contacts of \( FC_i \) and \( FC_j \) are mutual contacts and \( mcr_{ij} = 0.0 \) means there is no mutual contact between \( FC_i \) and \( FC_j \). Furthermore, the same type of P2P network flow clusters that generated by different bots within the same botnet tend to have a higher ratio of mutual contacts. As such, the P2P bots clustering problem can be considered as a network community detection problem. As shown in Fig. [3](#), each host might be running P2P bots or legitimate P2P applications or both, and each P2P bot or each legitimate P2P application generates different types of network flow clusters. Our community detection aims to cluster the same type of P2P network flow clusters generated by different bots into the same network flow cluster community. As such, each network flow cluster should only belong to a single network flow cluster community, but each host might belong to different host communities. Also, each botnet might contain several different network flow cluster communities. Once one network flow cluster community has been detected as belonging to a botnet, we consider the corresponding hosts as bots.

We used Louvain method, a modularity-based community detection algorithm \([28]\), due to (a) its definition of a good community detection result (high density of weighted edges within communities and low density of weighted edges between communities) is perfect-suited for our P2P botnet community detection problem; (b) it outperforms many other modularity methods in terms of computation time \([28]\); and (c) it can handle large network data sets (e.g., the analysis of a typical network of 2 million nodes takes 2 minutes \([28]\)).

Given \( G_{mc} = (V, E) \) as input, Louvain method outputs a set of network flow cluster communities \( Com = \{com_1, com_2, \ldots, com_{|Com|}\} \), where \( com_i = (V_{com_i}, E_{com_i}) \). \( V_{com_i} \) is a set of network flow clusters in \( com_i \). \( E_{com_i} \) is a set of edges, where \( ∀ e_{jk} ∈ E_{com_i} \), we have \( e_{jk} ∈ E \) and \( v_j, v_k ∈ V_{com_i} \).

2) **Botnet Communities Detection**: Given a set of communities \( Com \), for each community \( com_i ∈ Com \), we calculate its \( avgddr_i \) and \( avgmcr_i \) as follows.

\[
avgddr_i = \frac{\sum_{v_j ∈ V_{com_i}} dd_{ij}}{|V_{com_i}|} \tag{2}
\]

\[
avgmcr_i = 2 \times \frac{\sum_{e_{jk} ∈ E_{com_i}} mcr_{jk}}{|V_{com_i}| \times (|V_{com_i}| - 1)} \tag{3}
\]

We define two thresholds \( Θ_{avgddr} \) and \( Θ_{avgmcr} \). \( ∀ com_i ∈ Com \), if \( avgddr_i ≥ Θ_{avgddr} \) and \( avgmcr_i ≥ Θ_{avgmcr} \), we consider \( com_i \) as a botnet network flow cluster community.
3) Bot Candidates Detection: Recall from Section III-C3 the MCG of botnet communities usually have a structure of one or several cliques. Therefore, we used a maximum clique detection method CliqueDetection to verify each bot network flow cluster from botnet flow network flow communities, and further identify bot candidates. Each time it tries to detect one or several maximum cliques on the given botnet (network flow cluster) communities. If the maximum clique (at least containing 3 vertices) has been found, we consider the network flow clusters in that clique as bot network flow cluster, and run the maximum clique detection algorithm on the remaining parts, until no more qualified maximum cliques to be found. Afterwards, we report the corresponding source hosts of the identified bot network flow clusters as the bot candidates.

Algorithm 3 P2P Botnet Detection

input: $G_{mc}$, $\Theta_{avgddr}$, $\Theta_{avgmcr}$. 
output: $S_{bot}$

1. $S_{botFCCom} = \emptyset$, $S_{botFC} = \emptyset$, $S_{bot} = \emptyset$
2. $Com \leftarrow$ Louvain($G_{mc}$)
3. for $com_i \in Com$ do
4. $\text{avgddr}_i \leftarrow \frac{\sum_{v_{com_i} \in Com_i} \text{ddr}_j}{\|V_{com_i}\|}$
5. $\text{avgmcr}_i \leftarrow 2 \times \frac{\sum_{e_{jk} \in E_{com_i}} \text{mcr}_{jk}}{\|E_{com_i}\| \times (\|V_{com_i}\| - 1)}$
6. if $\text{avgddr}_i \geq \Theta_{avgddr}$ and $\text{avgmcr}_i \geq \Theta_{avgmcr}$ then
7. $S_{botFCCom} \leftarrow S_{botFCCom} \cup \{com_i\}$
8. end if
9. end for
10. for $com_i \in S_{botFCCom}$ do
11. $S_{botFC} \leftarrow$ CliqueDetection($com_i$)
12. for $FC_i \in S_{botFC}$ do
13. for $f_{jk} \in FC_i$ do
14. $S_{bot} \leftarrow S_{bot} \cup \{ip_{src}\}$
15. end for
16. end for
17. end for
18. return $S_{bot}$

V. EXPERIMENTAL EVALUATION

A. Experiment Setup

1) Experiment Environment: All the experiments were conducted on a PC with an 8 core Intel i7-4770 Processor, 32GB RAM, running 64-bit Ubuntu 16.04 LTS operating system. Our system was implemented using Java with JDK 8.

2) Data Collection and Analysis Tool: We used three main datasets: (a) 24 hours network traces of 4 popular legitimate P2P applications, (b) 24 hours network traces of 5 P2P botnets, and (c) 24 hours network traces from a Trans-Pacific backbone line between the United States and Japan as the background network traces (non-P2P & manually verified P2P).

Legitimate P2P network traces ($D_{p2p}^b$): Our legitimate P2P network traces $D_{p2p}^b$ were obtained from the University of Georgia [8], which collected the network traces of 4 popular P2P applications for several weeks. We obtained the network traces of 16 eMule hosts, 16 FrostWire hosts, 14 uTorrent hosts and 14 Vuze hosts by randomly selecting a set of continuous 24 hours network traces of each host (as shown in Table III).

P2P botnets network traces ($D_{bot}$): Part of our botnets network traces were from the University of Georgia [8], containing 24 hours network traces of 13 Storm hosts and 3 Waledac hosts. We also collected 24 hours network traces of another three P2P botnets, Sality, Kelihos and ZeroAccess. These network traces were all collected from the hosts manually infected by the binary samples of Kelihos, ZeroAccess, and Sality obtained from [29]. Our data collection was operated in a controlled environment, where all malicious activities were blocked. The same data collection settings were used in several previous works [2, 4, 8]. We collected the network traces of 8 Kelihos bots, 8 ZeroAccess bots and 5 Sality bots (as shown in Table IV).

Background network traces ($D_{non}^b$ and $D_{p2p}^b$): We used a dataset from the MAWI Working Group Traffic Archive [7] as our background network traces, containing 24 hours anonymized and payload-free network traces at the transit link of WIDE (150Mbps) to the upstream ISP on 2014/12/10 (as shown in Table V). The dataset contains approximate 407,523,221 flows and 48,607,304 unique IPs. 79.3% flows are TCP flows and the rest are UDP flows.

We investigated the background network traces, and made our best effort to separate the P2P traffic ($D_{p2p}^b$) from the non-P2P traffic ($D_{p2p}^b$). Since the WIDE dataset was anonymized and payload-free, it prevented us from using payload analysis to thoroughly check if P2P traffic, especially P2P Botnet traffic existing there. Instead, we used port analysis to manually detect P2P traffic within the background dataset. This is based on the simple concept that many P2P applications have default ports on which they function (see [30] for a list of default network ports of popular P2P applications). We manually examined all the network flows of each host in the background network traces. If a host involved in more than five flows using any of the default P2P port values in either source port or destination port, we considered the host as a P2P host. After this procedure, we identified 667 P2P hosts.

One thing worth to be noticed is that despite the whole background network traces lasting for 24 hours, not all these P2P

| Date | Dur | # of hosts | # of flow | # of dstIP | Size |
|------|-----|------------|-----------|------------|------|
| 2014/12/10 | 24 hours | 48,607,304 | 407,523,221 | 788.7G | |

| Trace | # of hosts | # of flow | # of dstIP | Size |
|-------|------------|-----------|------------|------|
| eMule | 16 | 4,181,845 | 725,367 | 42.1G |
| FrostWire | 16 | 4,479,969 | 922,000 | 11.9G |
| uTorrent | 10 | 17,749,924 | 2,326,626 | 57.1G |
| Vuze | 14 | 7,577,039 | 1,208,372 | 20.3G |

| Trace | # of bots | # of flow | # of dstIP | Size |
|-------|-----------|-----------|------------|------|
| Storm | 13 | 8,603,399 | 145,967 | 5.1G |
| Waledac | 3 | 1,109,508 | 927,972 | 1.1G |
| Sality | 5 | 5,599,440 | 177,594 | 1.5G |
| Kelihos | 8 | 122,182 | 1,277 | 75.2M |
| ZeroAccess | 8 | 709,299 | 277 | 343.9M |

TABLE V: Traces of Background Network
Keep the connectedness of mutual contacts graph. The easiest way to obtain a list of background hosts is to sample the hosts randomly from $D^b_{non}$, with the respect of bipartite structure. However, since $D^b_{non}$ contains an extremely large number of hosts, simply sampling hosts randomly will result in that most of the sampled background hosts do not have a mutual contact with the other background hosts, which is much easier for our system to identify botnet communities. Because less number of mutual contacts among legitimate hosts means more disconnected legitimate communities in the corresponding MCG, which happens to be in favor of Louvain method to detect strongly connected botnet communities. Therefore, we need to sample a list of internal hosts in a way that every internal host should have at least one mutual contact with at least one another internal host.

To follow the criterions described above without making our evaluation tasks any easier, we propose the following synthetic experimental dataset generation procedure:

- Use a two-coloring approach to sample the network traces from $D^b_{non}$ without jeopardize the bipartite network structure and the connectedness of mutual contacts graph: (a) initialize two counters, $C_{black}$ and $C_{white}$, to count the number of hosts colored in black and white respectively; (b) coloring a random host $h_i$ as black, and $C_{black}$ plus one; (c) coloring all contacts of $h_i$ as white, and increase $C_{white}$ by the number of hosts colored as white in this round; (d) for each new colored host, color its contacts with the opposite color, and adjust the counters repeatedly, until we have $C_{black} \geq 9,333$ and $C_{white} \geq 9,333$; (e) select the colored host set with exactly 9,333 hosts as the internal hosts, the hosts in the other set will be the external hosts; and (f) extract the network traces of the 9,333 internal hosts from $D^b_{non}$. Then, it forms a bipartite graph, where each colored host set forms a bipartite component, and each host shares at least one mutual contacts with some other hosts from its own bipartite component.

- To maintain a bipartite network structure of botnets and legitimate P2P hosts, we eliminate all communications among bots in $D^b_{bot}$, and P2P hosts in $D^b_{p2p}$ and $D^b_{p2p}$.

- To mix $D^b_{bot}$ and $D^b_{p2p}$ with $D^b_{non}$, each time we randomly select 97 internal hosts out of 9,333 background hosts, map the 97 hosts IPs to 37 bots IPs ($D^b_{bot}$) and 60 legitimate P2P hosts IPs ($D^b_{p2p}$), and merge the corresponding network traces.

To evaluate our system, 100 synthetic experimental datasets were generated by running this procedure. Table VII illustrates the summaries of the experimental datasets (EDs).

### B. Evaluation on P2P Network Flow Detection

We evaluated the P2P network flow detection with different $\Theta_{dd}$. We applied this component on all 100 EDs, and Table VIII shows the average detection rate and false positives with different $\Theta_{dd}$, ranging from 2 to 13,500. If $\Theta_{dd}$ is set too small, non-P2P hosts are likely to be detected as P2P hosts, which results in many false positives. For instance, when $2 \leq \Theta_{dd} \leq 5$, at least 110 non-P2P hosts were falsely identified as P2P hosts. If $\Theta_{dd}$ is set too large, all P2P hosts will be removed, which results in false negatives. For instance, when $\Theta_{dd} = 10,000$, most of the P2P hosts were falsely discarded, and only 18 P2P hosts were detected.
given a set of bots belonging to n botnets $X = \{X_1, X_2, \ldots, X_n\}$ (the ground truth), and the community detection results, m communities $Y = \{Y_1, Y_2, \ldots, Y_m\}$, define Bot Separation Index (BSI) and Bot Aggregation Index (BAI) as $\text{BSI} = a/(a + c)$ and $\text{BAI} = a/(a + b)$, where $a$ is the number of pairs of bots that are in the same botnet in $X$, and in the same community in $Y$; $b$ is the number of pairs of bots that are in different botnets in $X$, and in the same community in $Y$. $\text{BSI}$ denotes the degree of that bots coming from different botnets being separated into different communities. BAI denotes the degree of that bots coming from the same botnet being clustered into the same community. Both BSI and BAI are between 0.0 and 1.0, and the higher the better. “BSI equals to 1.0” means all different types of bots are well separated, and “BAI equals to 1.0” means all the same types of bots are well clustered.

Given $p$ bots and $q$ legitimate applications, define Bot-Legitimate Separation Index (BLSI) as $\text{BLSI} = d/(p \times q)$, where $d$ is the number of pairs of a bot and a legitimate application being separated into different communities via our method. BLSI indicates the ability of our method to separate bots and legitimate applications. BLSI is between 0.0 and 1.0, and the higher the better. “BLSI equals to 1.0” means all pairs of one bot and one legitimate application are well separated.

Table [IX] shows the community detection results with different $\Theta_{mc}$, ranging from 0.0 to 1.0. If $\Theta_{mc}$ is set too small, there will be more non-zero weight edges, which might result in less but larger communities. On the other hand, if $\Theta_{mc}$ is set too large, most of the vertices will be isolated, which results in more but smaller communities. For instance, as $\Theta_{mc}$ increasing, BSI decreased. When $\Theta_{mc} \leq 0.4$, BSI was around 0.8 to 0.85, meaning one or more botnets have been split into different communities. It turned out to be our algorithm separates the Storm botnet (13 bots) into two communities, one containing 10 bots and another containing 3 bots. Changing $\Theta_{mc}$ does not affect BSI and BLSI. BSI=1.0 means our system separates different types of bots into different communities. BLSI=1.0 means our system separates bots and legitimate P2P applications into different communities. The result demonstrated that our system is very effective and robust in separating bots and legitimate hosts, and separating different types of bots. Since larger $\Theta_{mc}$ will result

C. Evaluation on Community Detection

We evaluated the performance of community detection with different $\Theta_{mc}$. We applied this component on the remaining network flows (100 EDs) of the previous component (with $\Theta_{dd} = 30$). For each ED, our system generated a MCG $\mathcal{G}_{mc} = (V, E)$ with a pre-defined threshold $\Theta_{mc}$, where each edge $e_{ij} \in E$ contained a weight $mc_{ij} \in [0.0, 1.0]$. Afterwards, we applied Louvain method (with default resolution 1.0) on the MCG for community detection. The choice of $\Theta_{mc}$ would have an influence on the community detection results.

We evaluated the community detection performance in terms of (a) the ability to cluster a pair of bots belonging to the same botnet, (b) the ability to separate a pair of bots coming from different botnets, and (c) the ability to separate bots and legitimate applications. As such, we propose three criterions to evaluate the community detection performance below.
TABLE X: Botnet Detection Results For Different $\Theta_{avgddr}$ and $\Theta_{avgmcr}$.
(ZeroA.: the detection rate of ZeroAccess; FP: the number of false positives.)

| $\Theta_{avgmcr}$ | 0.0   | 0.05  | 0.1   | 0.15-0.35 | 0.4   | 0.6-0.8 |
|-------------------|-------|-------|-------|-----------|-------|---------|
|                    | 0.0   | 0.2   | 0.4   | 0.6   | 0.8   |         |
| ZeroA.            | 100%  | 100%  | 100%  | 100%  | 100%  |         |
| Waledac           | 100%  | 100%  | 100%  | 100%  | 100%  |         |
| Storm             | 100%  | 100%  | 100%  | 100%  | 100%  |         |
| Kelihos           | 100%  | 100%  | 100%  | 100%  | 100%  |         |
| Salty             | 100%  | 100%  | 100%  | 100%  | 100%  |         |
| Precision         | 28.9% | 29.1% | 29.3% | 38.1% | 34.8% |         |
| Recall            | 100%  | 100%  | 100%  | 100%  | 100%  | 86.5%   |
| FP                | 91    | 90    | 89    | 60    | 60    |         |
| F-score           | 44.8% | 45.4% | 45.4% | 55.2% | 49.6% |         |

Table X shows the P2P botnet detection results which supports our idea that the AVGDDR of legitimate P2P network flow cluster communities is lower than most of the P2P botnets network flow cluster communities. For instance, the AVGDDR of all (60/60) legitimate P2P network flow cluster communities were higher than 0.6, and the AVGDDR of 32 out of 37 botnets were higher than 0.8. The other 5 turned out to be 5 Salty bots, which could be detected by AVGMCR. Also, the legitimate P2P network flow clusters have lower AVGMCR than P2P bots (i.e., $\Theta_{avgmcr} \in [0.15, 0.35]$). For most of the botnets (i.e., ZeroAccess, Waledac, Kelihos and Salty), our system is effective (100% detection rate with zero false positive) and stable over a large range of $\Theta_{avgddr}$ (i.e., $[0.0, 0.6]$) and $\Theta_{avgmcr}$ (i.e., $[0.15, 0.8]$). Storm has a relative small AVGMC, hence the effective parameters narrowed down to $\Theta_{avgddr} \in [0.0, 0.6]$ and $\Theta_{avgmcr} \in [0.15, 0.35]$.

E. Evaluation on Enhanced PeerHunter

1) Analyzing the System Effectiveness: We applied Enhanced PeerHunter on 100 EDs, with $\Theta_{dd}=30$, $\Theta_{mcr}=0.1$, $\Theta_{avgddr}=0.6$ and $\Theta_{avgmcr}=0.15$, and all the results were averaged over 100 EDs. Using $\Theta_{avgddr}=0.6$ and $\Theta_{avgmcr}=0.15$ was based on our empirical study (shown in Table X). As illustrated in Table XI our system identified all 97 P2P hosts from 10,000 hosts, and detected all 37 bots from those 97 P2P hosts, with zero false positive, which demonstrated that Enhanced PeerHunter is effective and accurate in detecting P2P botnets.

2) Analyzing the System Scalability: The system scalability is to evaluate the practicality of our systems to deal with the real world big data. First, we applied Enhanced PeerHunter on 100 EDs of 10,000 internal hosts to analyze the processing time of each component. Our system has a scalable design based on efficient detection algorithm and distributed/parallelized computation. As shown in Table XI community detection and botnet detection had negligible processing time compared with P2P network flow detection and MCG extraction, since our first two steps (i.e., P2P network flow detection and MCG extraction) were designed to reduce a huge amount of the hosts subject to analysis (i.e., 99.03% in our experiments). The P2P network flow detection component has linear time complexity, since it scans all the input flows only once to get the flow clusters and further detect P2P flow clusters. However, since it is the very first component to process the input data (data could be large), it still costs the highest processing time (i.e., 15 minutes). To accommodate the growth of a real-world input data, we designed and implemented the P2P network flow detection component using a MapReduce framework, which could be deployed in distributed fashion on scalable cloud computing platforms (e.g., amazon EC2). The MCG extraction component requires pairwise comparison to calculate edges weights. Let $n$ be the number of P2P network flow clusters subject to analysis and $m$ be the maximum number of distinct contacts of a P2P network flow cluster. We implemented the comparison between each pair of hosts parallelly to handle the growth of $n$. If we denote $k$ as the number of threads running in less edges in the MCG, which could reduce the execution time of community detection, we used $\Theta_{mcr} = 0.1$ as our system parameter.

D. Evaluation on Botnet Detection

We evaluated the botnet detection component with different parameter settings. We applied this component on the remaining network flows (100 EDs) of the previous component (with $\Theta_{dd}=30$ and $\Theta_{mcr}=0.1$). We assumed that all the host in the background trace ($D^b$ and $D_{pp2}^b$) were not malicious, and would be reported as false positives if being detected.
Before P2P detection | P2P detection | Community detection | Botnet detection
--- | --- | --- | ---
# of hosts | 10,000 | 97 | 97 | 37

Table XI: The number of hosts identified by each component

| Processing Time | P2P Network Flow Detection | MCG Extraction | Community Detection | Bot Detection | Total |
|---|---|---|---|---|---|
| 15 minutes | 5 minutes | 5 seconds | 10 seconds | 20 minutes |

Table XII: Enhanced PeerHunter Execution Time

![Figure 4](image_url) Fig. 4: Processing time with different data size and $\Theta_{dd}$.

parallelly, the time complexity of MCG extraction is $O(n^2m)$. For a given ISP network, $m$ grows over time. Since our system uses a fixed time window (24 hours), for a given ISP network, $m$ tends to be stable and would not cause a scalability issue. Besides, since the percentage of P2P hosts of an ISP network is relatively small (i.e., 3% [2]), an ISP network usually has less than 65,536 (/16 subnet) hosts, and most P2P hosts generate less than 150 P2P network flow clusters (our empirical study), $n$ would be negligible compared with $m$. Moreover, since the waiting stage bots always act stealthily and only make necessary communications, $m$ also will not be large. We also tested our system using different sizes (i.e., different number of internal hosts) of EDs. For each size, we generated 10 EDs, and recorded the average processing time of our system with different $\Theta_{dd}$. As shown in Fig. 4 compared with the size of datasets, $\Theta_{dd}$ has more influence on the system scalability. Because in our P2P network flow detection component, $\Theta_{dd}$ has an impact on $n$ (the number of P2P network flow clusters subject to analysis), and larger $\Theta_{dd}$ leads to smaller $n$, thus less processing time. For instance, when $\Theta_{dd} = 10$ or 30, the increase of processing time, caused by increasing the size of data, was much less than when $\Theta_{dd} = 2$. Therefore, our system is very scalable on different sizes of data with an appropriate $\Theta_{dd}$ (e.g., 10 or 30). Also, by tuning $\Theta_{dd}$, our system has the potential to deal with different size of datasets in a reasonable time. To summarize, Enhanced PeerHunter is scalable to handle the real world network data.

3) Analyzing the Effectiveness of System Parameters:

Although we had analyzed the effectiveness of $\Theta_{dd}$, $\Theta_{avgddr}$ and $\Theta_{avgmcr}$ within the corresponding components, the effectiveness of combinations among different values of $\Theta_{dd}$, $\Theta_{avgddr}$ and $\Theta_{avgmcr}$ has not been studied. As shown in Fig. 5 we used precision, recall and false positives to evaluate the effectiveness of different parameter combinations. As discussed in Section V-B $\Theta_{dd}$ is used to detect P2P network flow clusters. Larger $\Theta_{dd}$ tends to result in more false negatives (lower recall), and smaller $\Theta_{dd}$ tends to result in more false positives (lower precision). For instance, changing $\Theta_{dd}$ from 30 or 50 to 10 resulted in 47 or 42 more false positives ($\Theta_{avgddr} = 0.15$) as shown in Fig. 5c and Fig. 5f respectively. When $\Theta_{dd} \in \{30, 50\}$, $\Theta_{avgddr} \in [0.15, 0.35]$ and $\Theta_{avgddr} \in [0.2, 0.6]$, our system yielded 100% detection rate with zero false positive. Even when $\Theta_{dd} = 10$, our system can still work effectively with $\Theta_{avgddr} \in [0.25, 0.35]$ and $\Theta_{avgddr} \in [0.2, 0.6]$. This demonstrated our system can work effectively over several different parameter combinations.

4) Analyzing the “True” False Positives when $\Theta_{dd} = 10$:

In this section, we discuss about some interesting findings about the false positives resulted from setting $\Theta_{dd} = 10$. As discussed in Section III-C2 $\Theta_{avgddr}$ is used to capture the “P2P behavior” of network flows, and $\Theta_{avgmcr}$ is used to capture the “botnet behavior” of network flows. Hence, if we use a larger $\Theta_{avgddr}$ (i.e., 0.6) and a smaller $\Theta_{avgmcr}$ (i.e., 0.0), most of the false positives should be legitimate P2P host. For instance, in Fig. 5f when $\Theta_{dd} = 10$, $\Theta_{avgddr} = 0.6$ and $\Theta_{avgmcr} = 0.0$, 115 out of 118 false positives were P2P hosts (60 from $D_p^{24p}$ and 55 from $D_p^{24p}$). On the other hand, we assume that if we use a smaller $\Theta_{avgddr}$ (i.e., 0.2) and a larger $\Theta_{avgmcr}$ (i.e., 0.15), some of the false positives might come from the other types of botnets. As shown in Fig. 5e when $\Theta_{dd} = 10$, $\Theta_{avgddr} = 0.2$ and $\Theta_{avgmcr} = 0.15$, 9 out of 47 false positives were not our known legitimate P2P hosts. We investigated these false positives, with their anonymized and payload-free network traces. It turned out that, 4 out of the 9 false positives (i.e., “180.217.2.181”, “180.217.2.246”, “180.217.2.248” and “180.217.2.177”) were listed in the Barracuda Reputation Block List (BRBL) [32], a highly accurate list of the IP addresses known to send spam. Hence, we are convinced that those false positives were infected with virus or botnets. These interesting “true” false positives findings demonstrated that our system has the potential to detect other unknown botnets.

F. Mimicking Legitimate P2P Application Attacks (MMKL)

Our work is focusing on detecting P2P botnets from legitimate P2P applications. If the adversaries (e.g., botmasters) know our techniques in advance, they might attempt to evade our system via instructing P2P bots to mimic the behavior of legitimate P2P applications. Inspired by [2], in this section, we propose two evasion attacks. All the parameters used in experiments of this Section were the same as in Section V-E.
PeerHunter \[5\] and Enhanced PeerHunter under this attack, detecting P2P bots while the bots traffic and the legitimate flows coming from different P2P applications, it is capable of mutual contacts graph, which could differentiate the network since our detection algorithm is based on network-flow level require the botnets to generate more or new types of network legitimate P2P traffic will dominate the host level behavior. As such, the botnet traffic will be overlapped with the legitimate P2P applications running on the same machine at the same time. As such, the botnet traffic can instruct the bots to passively generate network traffic that communicates with another of n peers. We inserted certain network flows communicating with an extra set of randomly selected peers. As shown in Fig. 6, our community detection commu-
component is robust to AMMKL, since both BAI and BLSI were unchanged and only BSI dropped a little bit when $\gamma$ increased. When combining both attacks, both BSI and BAIP dropped a lot, and BLSI dropped from 1.0 to around 0.88, as $\gamma$ increasing from 0.0 to 3.0. This is because when combining both attacks, as $\gamma$ increasing, the community detection component tends to cluster different types of bots into the same community and separate the same type of bots into different communities. The good news is, it can still well separate bots and legitimate P2P hosts into different communities. In summary, even though combining both attacks makes it harder for our method to separate different or aggregate the same type of bots, Enhanced PeerHunter is still robust in separating P2P bots from other hosts in the community detection process.

As shown in Fig. 7c both scenarios (i.e., AMMKL and combining both attacks) did not introduce new false positives (i.e., precisions equals to 1.0). Compared with conducting AMMKL, combining both attacks has more influences on the dropping of detection rate. Fig. 7a and Fig. 7b illustrate the detection rate of each botnet under two different scenarios, where the detection rate of different botnets started to drop around different $\gamma$. Table XV shows the analysis of all 5 botnets. Take Storm for instance, to affect the detection of Storm, each P2P network flow cluster of Storm needs to communicate with at least an extra 40% of its current peers, and in order to completely evade our system, $\gamma$ needs to be increased to 80%. Consider the fact that each Storm host generates an average of 67 P2P network flow clusters in 24 hours, and each network flow cluster communicates to an average of 740 peers. As such, to completely evade our system, each Storm host must communicate with at least an extra of $67 \times 740 \times 80\% \approx 39,664$ peers. In this case, it makes the P2P botnet less stealthy, less efficient and more exposed to trigger anomaly-based P2P botnet detection approaches [33]. In conclusion, although our system could not completely mitigate AMMKL, conducting AMMKL makes the botnets less stealthy, less efficient and more exposed, which still shows a winning of our system against P2P botnets.

G. Comparison to Zhang et al. [2]

We compared our system to one of the state of the art P2P botnet detection system Zhang et al. [2]. They proposed a scalable botnet detection system capable of detecting stealthy P2P botnets (i.e., in the waiting stage), where no knowledge of existing malicious behavior is required in advance. The system first applies a two-step flow clustering approach to create the fingerprints of hosts that have engaged in P2P activities. Afterwards, it applies two layers of filtering to detect potential P2P bots: a coarse-grained filtering to detect “persistent” P2P hosts that have longer active time of P2P behaviors, and a fine-grained filtering that applies hierarchical clustering to group pairs of P2P hosts that have less distance between their fingerprints. Our system shares many similarities with Zhang et al. [2]. For instance, both systems are (a) using network flow-based approach, (b) using unsupervised approach (i.e., no knowledge of existing malicious behaviors are required and have the potential to detect unknown botnets), (c) claiming to work while the botnet traffic are overlapped with the legitimate P2P traffic on the same set of hosts, (d) designed to have the built-in scalability, and (e) deployed at the network boundary (e.g., gateway), thus could be evaluated on the same datasets.

The main differences between our system and Zhang et al. [2] are listed as follows. First, two systems are using different network flow features. Zhang et al. [2] uses the number of bytes and packets of each flow; Enhanced PeerHunter uses the bytes-per-packet rate of each flow. Second, two systems are using different approach to cluster network flows (i.e., at different granularity). Zhang et al. [2] uses a two-step distance-based clustering (i.e., k-means, BIRCH) to cluster network flows of similar feature values; Enhanced PeerHunter clusters the network flows that have exactly the same feature values. Third, two systems apply the botnet detection step at different levels (i.e., host-level or network-flow-level). Zhang et al. [2] uses the distance between each pair of hosts to detect bots; Enhanced PeerHunter uses the distance between each pair of network flows to detect botnet network flow communities and then further identify the corresponding bots. Last but not least, two systems are using different heuristics to detect botnets. Zhang et al. [2] uses an threshold on the height of the hierarchical clustering dendrogram to detect bot clusters, which is very sensitive to the experimental datasets (as shown in Table XIV); Enhanced PeerHunter uses network-flow level community behavior analysis (i.e., AVGDDR and AVGMCR) to identify botnet (network flow) communities, which is more robust to the proposed attacks and can also be extended to other/new community behaviors.

We implemented a prototype system of Zhang et al. [2] did not have a publicly available implementation. Most of our implementations followed the description as in [2], other than the system parallelization, which has no impact on the system effectiveness evaluation. The experimental datasets used in both works are also differ-

|                  | No Attack | PMMKL | No Attack | PMMKL | No Attack | PMMKL | No Attack | PMMKL |
|------------------|-----------|-------|-----------|-------|-----------|-------|-----------|-------|
| PeerHunter [2]   | 100%      | 0%    | 100%      | 0%    | 100%      | 0%    | 100%      | 0%    |
| Enhanced PeerHunter | 100%     | 0%    | 100%      | 0%    | 100%      | 0%    | 100%      | 0%    |
| Zhang et al. [2] (i.e. $\theta_{bot} = 0.6$) | 100% | 0% | 100% | 0% | 100% | 0% | 100% | 0% |
| Zhang et al. [2] (i.e. $\theta_{bot} = 0.8$) | 100% | 0% | 100% | 0% | 100% | 0% | 100% | 0% |

TABLE XIV: Comparison of the botnet detection results under no attack and PMMKL attack. (* detection rate)
For instance, we evaluated our system on 100 synthetic experimental datasets (different background traffic and different topology, as described in Section V-A3) and took the average results; Zhang et al. [2] was evaluated on single customized dataset. Furthermore, even though both datasets use the same 24 hours time window, our datasets have much more internal hosts (i.e., 10,000 vs. 953), higher legitimate P2P hosts to P2P bots ratio (i.e., 727:37 vs. 8:16), and more types of botnets (i.e., 5 vs. 2). To summarize, our experimental datasets are more challenging and comprehensive.

We applied our implemented Zhang et al. [2] on the same experimental datasets as Enhanced PeerHunter under two circumstances (i.e., No Attack and PMMKL). We followed the same settings for most of the system parameters as described in [2], such as $\Theta_{BGP} = 50$, $\Theta_{p2p} = 0.5$, $K = 4,000$, $\lambda = 0.5$. Since the default value of $\Theta_{bot} (i.e., 0.95)$ used by the original paper, did not perform well on our dataset, we evaluated Zhang et al. [2] using two other different well selected values of $\Theta_{bot}$ (i.e., 0.6 and 0.8) that shows better results.

From the experimental results (Table XIV), we achieved several observations as follows. First, Zhang et al. [2] is more sensitive to the experimental dataset. For instance, Zhang et al. [2] was reported to achieve 100% detection rate and 0.2% false positive rate on their own datasets (using $\Theta_{bot} = 0.95$), while could not achieve similar results on our datasets using either the default parameter ($\Theta_{bot} = 0.95$) or the well selected parameter ($\Theta_{bot} = 0.6$ or $\Theta_{bot} = 0.8$). Second, as discussed in Section V-E our system is more stable and effective over a large range of system parameters ($\Theta_{avgdr}$ and $\Theta_{avgncr}$), while Zhang et al. [2] is more sensitive to its system parameter ($\Theta_{bot}$). For instance, Zhang et al. [2] had higher precision (lower false positives) and lower recall (higher false negatives) while using $\Theta_{bot} = 0.6$ comparing with using $\Theta_{bot} = 0.8$. Third, our system outperforms Zhang et al. [2] in terms of the detection rate of different botnets, the overall precision, recall and false positives. For instance, our system achieved 100% detection rate with zero false positives under different circumstances, while Zhang et al. [2] failed to detect all the
bots under both well selected parameters. At last, our system is more robust to PMMKL attack. For instance, PMMKL attack had no impact on the effectiveness of our system, while decreasing the F-score of Zhang et al. \cite{2} from 97.3% to 75% ($\Theta_{\text{bot}} = 0.6$) or from 74.6% to 72.7% ($\Theta_{\text{bot}} = 0.8$).

VI. DISCUSSION

A. Evasions and Possible Solutions

To avoid being detected by Enhanced PeerHunter, the botmaster could use a combination of the following three approaches: (a) adding randomized paddings or junk packets to influence the bytes-per-packet characteristics for network flow clustering, (b) reducing the number or rate of destination diversity, or (c) reducing the number or rate of mutual contacts. To deal with the randomized spatial-communication behavior, we could adopt more time-communication features, such as packet/flow duration and inter-packet delays, or apply more generalized features, such as the distribution, mean or standard deviation of bytes-per-packet. The other two evasion approaches would be the victory of our system. On one hand, to reduce the number or rate of destination diversity, a bot has to limit its communication to the network of certain locations, which degrades the P2P botnet into a centralized fashion. On the other hand, reducing the number of mutual contacts means there will be less bots targeting on the same set of victims, and less bots playing the role as botmasters, which will jeopardize the effectiveness and the decentralized structure of a P2P botnet. Also, as shown in Section VI-F2, reducing the rate of mutual contacts while maintaining the same number of mutual contacts (i.e., by conducting AMMKL) will make the botnets less stealthy, less efficient and more exposed to the other detection systems (e.g., anomaly-based botnet detection using high volumes of network traffic).

B. The deployment of Enhanced PeerHunter

In the previous sections, we simply assumed that our system is deployed at the boundary of a single organization. In this section, we discuss about the deployment of Enhanced PeerHunter in three more realistic scenarios.

1) The number of bots within an organization is too small: It would be challenging to build the MCG of botnet communities (i.e., the number of bots belonging to the same botnet is less than 3). In this case, we can deploy multiple Enhanced PeerHunter systems at the boundaries of multiple organizations, and correlate the network flows collected by those multiple Enhanced PeerHunter systems to build an appropriate size of MCG to detect botnet communities.

2) The number of bots within an organization is too large: The mutual contacts of certain bots might be within the organization internal network, hence invisible to the single system monitoring at the network boundary. In this case, we can deploy multiple Enhanced PeerHunter systems within the organization, that divide the organization network into several appropriate size of sub-internal networks. Each system is responsible for one sub-internal network.

3) The botmaster knows the system deployment location: In this way, the botmaster could assign the location of bots or control the communications of the bots based on the knowledge of the system deployment location to evade our system. For instance, the botmaster could assign bots into different sub-internal networks, and instruct most of the bots communicate with the others within the same sub-internal network. In this case, we could use the concept and idea of Moving Target Defense (MTD) \cite{24} to develop a strategy that makes it more difficult for botmasters to learn the deployment locations of our systems, by dynamically changing the settings or deployments of our systems.

C. Extend Enhanced PeerHunter to detect other botnets

Although Enhanced PeerHunter is designed to detect P2P botnets, our idea of using mutual contacts graph has the potential to detect not only unknown botnets, but also the other types of botnets (e.g., centralized botnets, such as IRC botnets \cite{18}, mobile botnets \cite{35}). Since bots are usually controlled by machines, rather than humans, bots from the same botnets tend to communicate with a similar set of peers or attacking targets. For instance, bots from the same IRC botnets tend to contact a similar set of C&C servers, while bots from the same mobile botnets tend to contact a similar set of satellite servers. Hence, we argue that Enhanced PeerHunter could be easily extended to detect the other types of botnets.

VII. CONCLUSION

We present a novel community behavior analysis based P2P botnet detection system, Enhanced PeerHunter, which operates under several challenges: (a) botnets are in their waiting stage; (b) the C&C channel has been encrypted; (c) the botnet traffic are overlapped with legitimate P2P traffic on the same host; (d) no bot-blacklist or “seeds” are available; (e) none statistical traffic patterns known in advance; and (f) does not require to monitor individual host. We propose three types of community behaviors (i.e., flow statistical features, numerical community features and structural community features) that can be used to detect P2P botnets effectively. In the experimental evaluation, we propose a network traces sampling and mixing method to make the experiments as unbiased and challenging as possible. Experiments and analysis were conducted to show the effectiveness and scalability of our system. With the best parameter settings, our system achieved 100% detection rate with none false positives. We also propose two mimicking legitimate P2P application attacks (i.e., PMMKL and AMMKL). The experiment results showed that our system is robust to PMMKL, and will make the botnets less stealthy, less efficient and more exposed while conducting AMMKL.

REFERENCES

[1] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, “Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets,” in Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013, pp. 97–111.

[2] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, “Building a scalable system for stealthy p2p-botnet detection,” Information Forensics and Security, IEEE Transactions on, vol. 9, no. 1, pp. 27–38, 2014.
