Detecting P2P botnets by discovering flow dependency in C&C traffic

Hongling Jiang · Xiuli Shao

Abstract Botnets are widely used by attackers and they have evolved from centralized structures to distributed structures. Most of the modern P2P bots launch attacks in a stealthy way and the detection approaches based on the malicious traffic of bots are inefficient. In this paper, an approach that aims to detect Peer-to-Peer (P2P) botnets is proposed. Unlike previous works, the approach is independent of any malicious traffic generated by bots and does not require bots' information provided by external systems. It detects P2P bots by focusing on the instinct characteristics of their Command and Control (C&C) communications, which are identified by discovering flow dependencies in C&C traffic. After discovering the flow dependencies, our approach distinguishes P2P bots and normal hosts by clustering technique. Experimental results on real-world network traces merged with synthetic P2P botnet traces indicate that 1) flow dependency can be used to detect P2P botnets, and 2) the proposed approach can detect P2P botnets with a high detection rate and a low false positive rate.

Keywords P2P botnet · Detection · Command and control · Flow dependency · Cluster · Security

1 Introduction

Botnets have emerged as one of the threats to the Internet. Attackers use botnets to serve as the infrastructures for a variety of cyber-crimes, such as distributed denial-of-service (DDoS) attacks, spamming, click fraud, phishing, keylogging

H. Jiang (✉) · X. Shao
College of Information Technical Science, Nankai University,
Tianjin, China
e-mail: hellojhl@163.com

and so on [1, 2]. As the cloud services become popular [3], future attackers may use cloud services to establish botnet and launch attacks. Clark et al. [4] have discussed the threat of cloud-based botnets and showed that attacks could be launched from botclouds.

Compare to other malwares, the distinguishing characteristic of botnet is the Command and Control (C&C) channel, through which botmaster can send commands to bots and receive information from them. The traditional C&C channel is based on IRC protocol. Most flexible botnets are based on HTTP protocol. Both IRC and HTTP botnets are centralized structures which provide botnets with the efficient communication. However, centralized structures are vulnerable to a single point of failure [5] and C&C servers are easy to be detected due to their heavy traffic [6].

Most recent botnets have evolved to Peer-to-Peer (P2P) structures. In a P2P botnet, there is no centralized point for C&C server. Each bot acts as both client and server. Even some nodes are offline, P2P botnet can continue to operate [7]. When losing some bots, a P2P botnet won't be disrupted because it is resilient to dynamic churn [8, 9]. Storm [7, 10–12], Nugache [10, 13] and Waledac [14–16] are some of the most popular P2P botnets. This paper focuses on the detection of P2P botnets.

Unfortunately, there are many challenges to detect P2P botnets: (1) the traffic of P2P botnet is similar to legitimate P2P network and it is hidden in normal traffic; (2) many P2P botnets such as Nugache, Storm, Waledac and Conficker employ encryption mechanisms that making approaches based on packet contents ineffective; (3) P2P bots launch malicious activities stealthily; (4) there is no central server in P2P botnets; (5) bots communicate with each other through random ports [7, 12, 13].

A number of approaches have been proposed to detect P2P botnets. Some approaches rely on the malicious

activities of botnets. However, bots stay idle most of the time until they receive commands [17, 18]. Besides, many botnets launch attacks in a stealthy way. Some other approaches require external systems which provide a list of known bots to detect P2P botnets. They could not detect botnets by themselves.

This paper presents a P2P botnet detection approach which focuses on C&C traffic of P2P bots regardless of how they perform malicious activities. The approach first discovers flow dependencies in C&C communications and then detects P2P bots using true flow dependencies. Compare with other approaches, our approach does not rely on the malicious activities of bots and focuses on the intrinsic characteristics of C&C traffic. In addition, it requires no external system and no a priori knowledge of botnets, such as traffic signatures [19].

The contributions of this paper are as follows. First, we found an intrinsic characteristic of C&C communications in P2P bots, which we called flow dependency. Second, we proposed a flow dependency extracting algorithm which is based on time information and a large number of samples. Third, we proposed a technique which utilizes flow dependency to detect P2P botnet.

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 presents the assumptions, architecture and implementation of our detection approach. The evaluation of our approach is shown in Section 4. Section 5 makes a discussion and Section 6 concludes the paper.

2 Related work

Since P2P botnets emerged on the Internet, many studies focus on analyzing the behaviors and structures of some popular P2P botnets, such as Storm [7, 10–12], Nugache [10, 13] and Waledac [14–16]. These studies provide us a good understanding of P2P botnets.

So far, many approaches have been proposed to detect centralized botnets. However, the techniques to detect P2P botnets are still ongoing. BotHunter [20] can detect centralized or P2P botnets by identifying a series of malicious behaviors, such as scanning, binary download and control channel establishment. BotMinner [21] is able to detect botnets regardless the C&C structures and network protocols. It assumes that bots in the same botnet communicate with the same C&C servers/peers and perform similar malicious activities. BotMinner finds clusters of hosts which share similar communication traffic and similar malicious traffic and considers such hosts as bots.

BotGrep [22] is an algorithm to detect P2P botnets by analyzing the communication graphs of the large network. Nodes in the graph represent hosts and edges represent the communications between hosts. BotGrep first identifies group of P2P hosts then differentiates P2P botnets from legitimate P2P networks based on the information of external system. BotTrack [23] detects P2P botnets using linkage analysis algorithm PageRank. It first creates a dependency graph between hosts and then runs the PageRank to extract nodes which are strongly linked to each other and outputs the hub rank and authority rank of each node. Given the authority and hub values, a density-based clustering algorithm is applied to find nodes with similar roles within the network. To identify bot groups, the data from honeypot is leveraged. The clusters which contain known bots are considered to be bot clusters. Coskun et al. [24] proposed a method which can identify potential members of an unstructured P2P botnet starting from a known peer. The paper is based on the hypothesis that in an unstructured P2P botnet, any given pair of P2P bots communicate with at least one common external bot during a given time window. The approach first constructs a mutual contacts graph then applies an iterative algorithm to identify other members of the botnet by computing a level of confidence to each host on the graph. The hosts with high level of confidence are declared as members of the same P2P botnet as the seed-bot.

Contrast to previous work, our approach does not rely on malicious activities of botnets. We focus on the C&C communications of botnet and detect the intrinsic characteristic of C&C communications. Besides, our approach does not require external tools to get bots' information, such as honeypots and intrusion detection systems. Our approach can detect P2P botnets by discovering the flow dependencies in C&C traffic.

3 Detection architecture and implementation

In this section, we first present the problem statement and motivations of the paper then propose the architecture and implement of our detection approach.

3.1 Problem statement and motivations

Botnets perform both malicious activities and C&C communication activities. However, bots does not perform malicious activities all the time. Before receiving commands, bots may stay idle. What's more, some bots perform malicious activities in a stealthy way. For example, bots send spam stealthily. It's hard to detect the malicious traffic [25]. On the other hand, the C&C communication activities are constant throughout the life-cycle of bots. That's because bots need to maintain connections with other bots to receive commands or updates. Therefore, we only focus on the C&C communication traffic to detect P2P botnet.

Compared with centralized botnets, the C&C communications of P2P botnets are relatively complex. To detect P2P botnets, we make use of the following observations: 1) there

322

Peer-to-Peer Netw. Appl. (2014) 7:320–331

are flow dependencies in the C&C traffic of P2P bots, and 2) although different bots connect different peers, it is probable that any given pair of P2P bots connect with at least one common external bot during a given time period [24, 25]. We now explain the two observations in detail.

In the first observation, why C&C traffic of P2P bots has flow dependencies? Note that many P2P botnets apply the publishing/subscribing mechanism [26], namely "pull" mechanism. In this mechanism, botmasters publish the commands over the P2P botnet. The commands are associated with special keys. In order to gain commands, bots search their neighbor peer lists for specific keys frequently/periodically [8]. In addition, bots frequently communicate with peers in their neighbor peer lists to send keep-alive messages [21]. Take the Storm botnet for an example, it utilizes pull mechanism. Every day 32 different keys are generated by an algorithm, which takes the current day and a random number between 0 and 31 as input [8].

According to the previous analysis, we assume that each bot in P2P botnets maintains a peer list to exchange keep-alive messages and receive commands or updates. It communicates with the same hosts repeatedly [27]. What's more, bots connect the peers of their lists in the same the order. This botnet model fits with the most current P2P botnets and other botnet models are outside the focus of our paper. We consider that flows between bots and different hosts in their lists manifest dependency relationship. Normal hosts behave randomly since human may think when they connect the Internet, while bots run programs and behave regularly. Thereby, the flow dependencies of bots are clearer than normal hosts.

Example of flow dependency in C&C traffic of P2P bot is shown in Fig. 1. Suppose that bot H is a P2P bot and four hosts are in its peer list. Bot H frequently/periodically communicates with the peers in its list. Every time bot H communicates with the peers in the same order. *flow A*, *flow B*, *flow C*, *flow D* are flows between bot H and peers in its list. The following flow always arrives after the previous flow within a predefined time window θ. Each random or fixed time Δ, bot H connects the peers in its list. In our flow dependency model, we assume previous flows trigger the following ones. We present a flow dependency graph which visualizes the dependency relationship between flows. Each node in the flow dependency graph represents a flow. A directed dotted line from *flow A* to *flow B* indicates that *flow B* is dependent on *flow A*, that is, *flow B* always appears shortly after *flow A*, denoted as *flow A*→*flow B*.
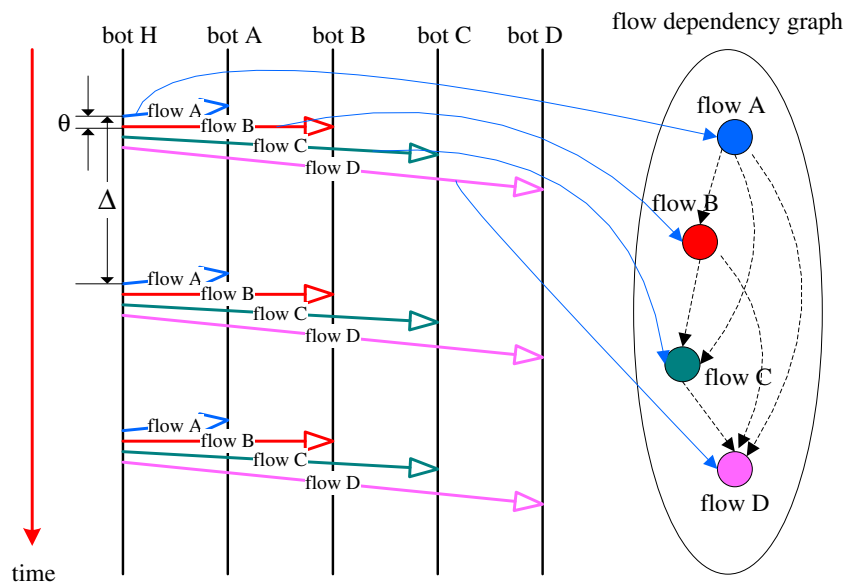
The second observation is obvious. Bots frequently connect peers in their lists to receive commands or updates. For a P2P botnet with an unstructured topology, peers in the lists of different bots have overlap. Thus, different bots in the same P2P botnet may connect common peers. However, aside from P2P bots, legitimate hosts may connect common peers. For example, there are popular servers that most hosts connect, such as Google and Yahoo etc. But they may not show flow dependencies. So we make use both of the two observations.

In summary, to detect hosts within a monitored network which are infected by P2P bots, we focus on C&C communications of P2P bots. We first discover flow dependencies in C&C traffic and then identify bots according to the common hosts in their flow dependencies.

3.2 Architecture and implementation of detection approach

In this section, we present the architecture and the implementation of our detection approach. As shown in Fig. 2, our approach consists of three components: flow capture, flow dependency extractor and bots detector.

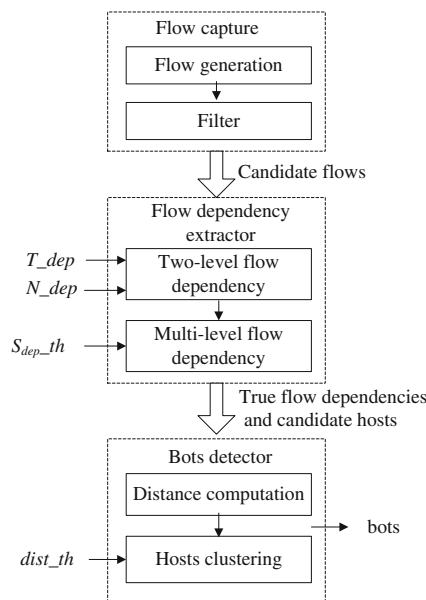Fig. 1 Example of flow dependency

Flow capture

Flow generation

↓

Filter

⇓ Candidate flows

Flow dependency extractor

T_dep →
N_dep →

Two-level flow dependency

↓

S_{dep}_th →

Multi-level flow dependency

⇓ True flow dependencies and candidate hosts

Bots detector

Distance computation

↓ → bots

$dist\_th$ →

Hosts clustering

Fig. 2 Architecture of detection approach

Flow capture is to obtain the candidate flows. It contains two modules: flow generation and filter. Flow generation translates the raw packets to TCP/UDP flows based on packet headers. Filter is to exclude the traffic which is unlikely related to P2P bots.

Flow dependency extractor discovers flow dependencies and considers the hosts which have flow dependencies as candidate hosts. Given a maximum time interval T_dep and difference of occur times N_dep between two dependency flows, flow dependency extractor first discovers two-level flow dependencies and then finds multi-level flow dependencies based on the two-level flow dependencies. The flow dependencies with score larger than S_{dep}_th are considered as true flow dependencies.

Bots detector is to identify P2P bots. It first computes the distance between any given pair of candidate hosts. The distance between two hosts relies on true flow dependencies and the common external hosts the two hosts connect with. Then bot detector applies a hierarchical clustering algorithm over hosts according to the distance computed above to identify P2P bots. The parameter $dist\_th$ is used to separate bot and normal clusters. We describe each of the components in detail in the following sections.

3.2.1 Flow capture

(1) Flow generation

To identify the flow dependencies, we translate raw packets into flows. It is nontrivial because, firstly, multiple packets belong to a flow and analyzing packets can not convey explicit dependency information and may introduce much redundancy. Secondly, flow records need much smaller storage and computation costs than packets.

A flow is defined as a sequence of packets between a source and a destination within a connection. A flow is generated in terms of IP addresses, ports and protocol. Packets with the same five tuple (*localIP*, *localPort*, *remoteIP*, *remotePort*, *protocol*) compose a flow. *localIP* and *localPort* are the IP address and port of host inside the monitored network. *remoteIP* and *remotePort* are the IP address and port of host outside the monitored network. The flow is bi-directional. The source IP address/port and the destination IP address/port are swappable. The time interval between any two consecutive packets in a flow must be less than the time-out threshold.

Due to most P2P botnets use UDP or TCP protocol for communication, we analyze UDP and TCP flows. TCP 3-way handshake (SYN, SYN-ACK, ACK) packets between a client and a server indicate the start of a TCP flow. TCP 4-way handshake (FIN, ACK, FIN, ACK) or RST packets indicate the termination of a TCP flow. While processing TCP flow records, we only analyze the successful TCP flows and ignore flows that do not have a completed handshake. UDP packets with the same five tuple (*localIP*, *localPort*, *remoteIP*, *remotePort*, *protocol*) are aggregated into a UDP flow and the time interval between any two consecutive packets in a flow is less than the timeout threshold.

To discover flow dependencies, we use TCP/UDP packet headers and time information. Such information is easy to obtain and independent of packet payload. The flow information is recorded as follows: (1) five-tuple; (2) start time, that is the arrival time of the first packet within a flow; (3) end time, that is the arrival time of the last packet within a flow; (4) the duration time of a flow, that is the difference between end time and start time; (5) the number of packets in a flow; (6) the bytes of a flow, that is the sum of bytes of the packets in a flow.

(2) Filter

To reduce the volume and noise in network traces, we filter out all the traffic unlikely related to P2P botnets. We note that the filter is not critical for our detection approach. However, it is useful to reduce the traffic and make the detection more efficient. The filter is down in three steps. In the first step, we filter out the flows with small bytes. The C&C flows of botnets generally consist of small packets. The total bytes of C&C flows are usually small. On the other hand, the total bytes of many legitimate flows may large, such as the media flows.

In the second step, we filter out long flows. Another characteristic of C&C flows is that the duration time of C&C flow is usually short. The long running flows are common in network, such as long SSH sessions and remote desktop connections, which could be kept alive for hours. We ignore all flows that are active over a long period of time.

324

Peer-to-Peer Netw. Appl. (2014) 7:320–331

In the third step, we filter out the flows rarely happen. Due to P2P bots frequently connect to the peers in their peer lists, the flows between bots and the peers in their lists frequently happen. Those flows between two hosts rarely happen may not be related to botnets. Due to P2P bots usually use random port to communicate, such as Nugache [13], we ignore the port number and only consider the IP addresses and protocol of a flow.

3.2.2 Flow dependencies extractor

Flow dependencies extractor is to discover flow dependencies and identify candidate hosts likely to be bots. Hosts having flow dependencies are considered as candidate hosts. Flow dependencies extractor first discovers two-level flow dependencies and then discovers multi-level flow dependencies.

(1)   Two-level flow dependency

How to extract two-level flow dependencies in a given trace? Our key insight is that if a pair of flows consistently occurs together, they may have dependency relationship. We discover flow dependencies by looking for the time correlation of flows. For example, if *flow B* is observed shortly after *flow A*, we can assume *flow B* is dependent on *flow A*. While time correlation may not always indicate a true dependency, we rely on a large number of samples to reduce the false positives. In other words, our idea is to extract a pair of flows which occur together frequently and repeat consistently over time.

However, there may be too many flows within a few hours. Examining every pair of flows is expensive and not scalable and efficient. To solve the issue, our approach applies two heuristics. Given a current flow, firstly, we only analyze the flows that start after a short time of the current flow because a flow is usually dependent on the flow which occurs recently. To capture all the possible flow dependencies, the time window should be set larger. Secondly, we only analyze the flows which occur with nearly number of times. If two flows always occur together, the number of occurrences of these flows are almost equal to each other during the same period.

Given the filtered flows obtained in previous step, all *m* flows with the same protocol (TCP/UDP) that share the same local host *h*, are aggregated into a group $G_h = \{f_i\}_{i=1,2...m}$, where each f_i is a single TCP/UDP flow. In a given period of monitor time, we compute the number of occurrences of each flow. For each group G_h, we sort flows $\{f_i\}_{i=1,2...m}$ in the order of their start time. Metric *T* denotes the number of times two flows occur together. On observing a flow record, we look forward to all the following flows that start within the predefine timeframe *T_dep*. If such flows exist, we further check whether the number of occurrences of these flows is almost equal to the current flow. *N_dep* denotes the difference of

occur times between two flows. If such flows are found, these flows may be dependent on the current flow. Each pair of flows is considered as a candidate flow dependency. We increase the metric *T* of these flow dependencies by one.

The identified candidate flow dependencies may have false positives due to coincidental flows that occur together. We rely on a large number of samples to reduce the false positives. A flow dependency is considered as true, if the score of the flow dependency is greater than S_{dep}_th. Give a candidate two-level flow dependency $f_i \rightarrow f_j$, we use the following metric to score it:

$$S_{dep}(f_i \rightarrow f_j) = \sqrt{\frac{T_{ij}^2}{N_i * N_j}} \qquad (1)$$

N_i is the number of occurrences of flow f_i. T_{ij} is the number of times flow f_j happens shortly after flow f_i. The larger the score is, the more likely the two of flows have dependent relationship. If the score of a flow dependency is larger than S_{dep}_th, we consider the flow dependency as true. Figure 3 shows the algorithm of extracting two-level flow dependencies.

(2)   Multi-level flow dependency

So far, two-level flow dependencies have been discovered. However, two-level flow dependencies also exist in normal traffic. For example, a flow to a web server is often dependent on a flow to a DNS server. But flow dependencies of P2P bot usually involve more than two flows, because the number of peers in the peer list of P2P bot is usually more than two. To identify P2P bots, multi-level flow dependencies should be extracted.

Multi-level flow dependencies are extracted by combining lower level flow dependencies. For example, two-level flow dependency $f_1 \rightarrow f_2$ and $f_2 \rightarrow f_3$ will infer a three-level flow dependency $f_1 \rightarrow f_2 \rightarrow f_3$. Further, if $f_3 \rightarrow f_4$ is a true flow

---

Algorithm 1 Extracting two-level flow dependencies
Input: F-the set of candidate flows during an epoch
Output: Two-level flow dependencies
1: Compute the number of occurrences
    for each flow f_i in F, denoted as N_i
2: **foreach** local host *h* in F **do**
3:    Find the set of flows G_h of *h*
4:    Sort the flows in G_h in the order of start time
5:    **foreach** flow f_i in G_h **do**
6:      Find flows F_h start after f_i within T_dep
7:      **foreach** flow f_j in F_h **do**
8:        **if** $|N_i\text{-}N_j| < N\_dep$ **then**
9:          **if** $f_i \rightarrow f_j \in$ candidate flow dependencies set *D* **then**
10:            $T_{ij} = T_{ij} + 1$
11:          **else**
12:            Insert $f_i \rightarrow f_j$ to *D*
13: **foreach** $f_i \rightarrow f_j$ in *D* **do**
14:    Compute $S_{dep}(f_i \rightarrow f_j)$
15:    **if** $S_{dep}(f_i \rightarrow f_j) > S_{dep}\_th$ **then**
16:      Label $f_i \rightarrow f_j$ as true flow dependency

Fig. 3 Algorithm of extracting two-level flow dependencies

dependency, it could deduce a four-level flow dependency $f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4$. The score of a k-level flow dependency is calculated as follows:

$$S(f_1 \rightarrow f_2 \rightarrow ... \rightarrow f_k) = \prod_{i=1}^{k-1} S(f_i \rightarrow f_{i+1}) \qquad (2)$$

When discovering the multi-level flow dependencies, we only consider the true flow dependencies whose scores are larger than S_{dep_th}. The local hosts which have two-level or multi-level flow dependencies are more likely to be P2P bots and they are considered as candidate hosts.

3.2.3 Bots detector

(1) Distance computation

$$Sim^{(k)}(H_a, H_b) = \begin{cases} \dfrac{\left| DEP_a^{(k)} \cap DEP_b^{(k)} \right|}{\left| DEP_a^{(k)} \cup DEP_b^{(k)} \right|} & if \left| DEP_a^{(k)} \cap DEP_b^{(k)} \right| \neq 0, \left| DEP_a^{(k)} \cup DEP_b^{(k)} \right| \neq 0 \\ \\ 0 & otherwise \end{cases} \qquad (3)$$

$DEP_a^{(k)}$ and $DEP_b^{(k)}$ are respectively the sets of remote hosts in the k-level flow dependencies of host H_a and H_b. Then we define distance between two hosts H_a and H_b as:

$$Dist(H_a, H_b) = \sum_{k=2}^{n} w_k * \left(1 - Sim^{(k)}(H_a, H_b) \right), w_k$$
$$= \frac{k}{\sum\limits_{k=2}^{n} k} \qquad (4)$$

w_k is the kth weight of distance and w_k is not the same when k changes. As we see previously, normal hosts may have flow dependencies, but their levels are usually lower than P2P bots. To identify P2P bots, we increase w_k when k increases.

(2) Hosts clustering

After computing the distance, a distance matrix $D = \{d_{ab}\}_{a,b=1...n}$ consisting of distance $d_{ab} = dist(H_a, H_b)$ between each pair of hosts is generated. n is the number of candidate hosts. To identify P2P bots clusters, a single-linkage hierarchical clustering algorithm [28, 29] is applied. The hierarchical clustering algorithm takes D as input and outputs a dendrogram which is a tree-like graph (see Fig. 7), in which the leaves represent the original hosts and the lengths of the edges represent the distances between clusters [28]. The dendrogram shows the relationship among hosts. The smaller the distance between two hosts is, the shorter

edge the hosts are connected in the dendrogram. To group the hosts into clusters, a parameter $dist\_th$ is used to cut the dendrogram at a certain height. $dist\_th$ is set according to the distribution of distances among hosts. If some hosts have small distances to each other and form a dense cluster and other hosts have larger distances, we set $dist\_th$ to the middle of the small distances and large distances. Too large $dist\_th$ may lead to false positives, normal hosts may be clustered to botnet clusters, while too small $dist\_th$ may lead to false negatives, and some bots may not be detected. Thus, $dist\_th$ should be set according to the distances distribution. We cut the dendrogram at $dist\_th$ and classify the clusters below $dist\_th$ as bot clusters.

4 Evaluation

In this section, we evaluate the effectiveness of our P2P botnet detection approach. We have tested its performance on real world traffic merged with synthetic P2P botnet traffic.

4.1 Dataset and experiment setup

We collected real world traffic by a sensor at the campus network of Tianjin Polytechnic University. The sensor runs tcpdump which is configured to collect TCP/UDP packets between internal and external networks and write to a pcap

Table 1 Statistics of background traffic

Description	Value
Duration of trace	5 h
Total number of packets	2,376,163
Number of flows	227,863
Number of active local hosts	181
Number of remote hosts	12,735
Number of (local IP, remote IP) pairs	468,606

file every hour. The data was captured through port mirroring on the core switch. The traffic rate is 150Mbps–200Mbps at daytime. The number of the hosts in the monitored subnetwork is about 200. Because we consider the flows with the same protocol, in our experiment we analyze the UDP traffic only. TCP traffic is analogous. The data we used is 5 h long from eight o'clock on April 4, 2011. The statistics of background traffic are shown in Table 1.

Given the 5 h background traffic, we counted the number of flows for each (local IP, remote IP) pair. As shown is Fig. 4, a lot of pairs have less than five flows. For example, 400697 (local IP, remote IP) pairs have only one flow. 37172 pairs have two flows. 10047 pairs have three flows. But bots (local IP, remote IP) pairs may have much more flows because they need to frequently connect the peers in their lists to get commands or updates. In our experiment, we filtered the traffic of (local IP, remote IP) pairs which have less than four flows.

Ideally, we should capture botnet traces in the Internet and add them to traces of non-botnet. Unfortunately, real bots traces are very difficult to obtain, due to the reluctance of ISPs to share their internal traffic. Besides, it is hard to get ground truth on which hosts are bots. An accurately labeled data set should be used to evaluate our approach. Inspired by former work, such as [22, 23, 30], which applied their approaches to synthetic traces. We also synthesize botnet traces and inject them into the real-world traces of non-botnet.
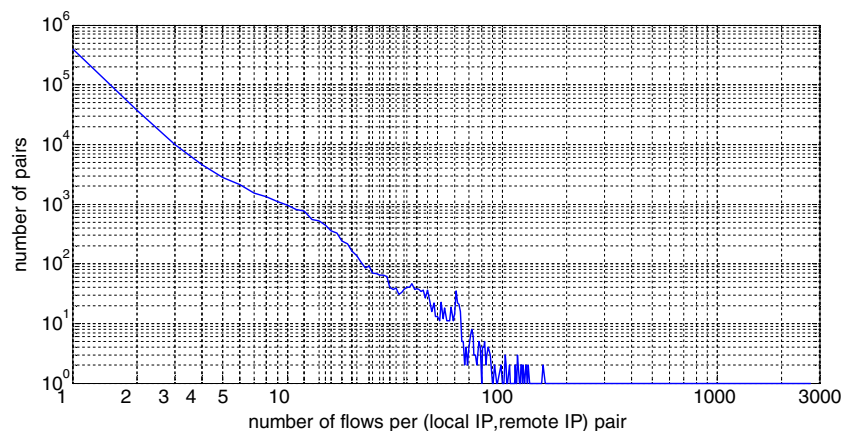
We added synthetic P2P botnet flows by taking into account the local IP address, remote IP address, start time, duration and protocol since our approach only relies on them. Each bot maintains a peer list. Every random time Δ ($\Delta \in [55_S, 65_S]$) bots sequentially connect the peers in their lists. The internal time interval θ is between 0.01 s and 0.1 s. In the same botnet, peers in lists of different bots may overlap. We define p as the percentage of common peers in the lists of different bots and change p to simulate different botnet models. To simulate the realistic scenario, we randomly select different IP addresses from the normal traffic and replace the IP addresses of bots. That is, the infected hosts show both normal and botnet-related behavior.

In our experiment, parameter $T\_dep=1$ s. Since the time intervals between most pairs of normal flows which do not have dependency are beyond 1 s. In our botnet model, the time interval between the previous flow and the following flow θ is in [0.01 s, 0.1 s] and less than 1 s. So we set T_dep to 1 s.

Parameter $N\_dep=1$. Using the 5 h data, we have examined the number of dependent flow pairs when N_dep increases from 1 to 8. The results are shown in Table 2:

Table 2 shows that as N_dep increases, the number of true botnet dependent flow pairs does not change and all the botnet flows have been discovered, while the number of true normal and total dependent flow pairs increase. Our goal is to discover botnet flow dependency, so the number of normal dependent flows is the small the better. So we set N_dep to 1.

4.2 Experiments results

We made a statistic of host percentage for different flow dependency level. We set common peers percentage $p=60$ % and flow dependency score threshold $S_{dep}\_th=0.5$. As shown in Fig. 5, the percentages of hosts which have two-level or three-level flow dependencies are higher. All the bots have two-level and three-level flow dependencies. Some normal hosts have two-level and three-level dependencies. But there is no normal host having flow dependencies with level

Fig. 4 The number of flows per (local IP, remote IP) pair

Table 2 The number of dependent flow pairs for different N_dep

N_dep	1	2	3	4	5	6	7	8
Number of true botnet dependent flow pairs	834	834	834	834	834	834	834	834
Number of true normal dependent flow pairs	815	1,105	1,300	1,473	1,561	1,611	1,727	1,797
Number of total dependent flow pairs	31,652	47,517	61,192	73,280	83,588	92,636	100,758	107,568

larger than three while bots still have flow dependencies with higher level. As the level of flow dependency increases, the percentages of bots decrease. It confirms the fact that bots have flow dependencies and their levels are higher. Although normal hosts have flow dependencies, their levels are lower and no more than three.

Taking the two-level flow dependencies as example, we examine the influence of S_{dep_th} on the true flow dependency percentage. It is easy to see from Fig. 6, as S_{dep_th} increases, the percentages of botnet true flow dependencies do not change and remain 100 %, while the percentages of normal true flow dependencies decrease. When S_{dep_th} increases from 0 to 0.1, the percentage of normal true flow dependencies decreases sharply, it is deduced that the scores of most normal flow dependencies are less than 0.1. This is due to normal pair of flows happen together coincidently and their number of samples is small. On the other hand, botnet flow dependencies have a large number of samples so their scores are large than S_{dep_th} regardless of what the value of S_{dep_th}. Our goal is to extract botnet true flow dependencies, so the percentage of normal true flow dependencies should be small. Large S_{dep_th} will eliminate more normal true flow dependencies. When S_{dep_th} is larger than 0.4, the percentage of normal true flow dependencies is very small. We could set S_{dep_th} in (0.4, 0.9].

When detecting bots, we applied hierarchical clustering on the distance matrix. Parameter $dist\_th$ is used to identify dense clusters and find the botnet clusters. Figure 7 shows an example of hierarchical clustering dendrogram. We set common peers percentage $p=60$ % and flow dependency score threshold $S_{dep\_th}=0.9$. P2P bots have common peers

in their lists, thus the remote hosts of flow dependencies overlap and result in small distances and dense clusters. As shown in Fig. 7, when we cut the tree at $dist\_th=0.6$ to identify dense clusters, distances between any pair of P2P bots are smaller than $dist\_th$. P2P bots form a dense cluster, while normal hosts have larger distances to each other and can not form a dense cluster.

The performance of our approach is measured by *detection rate* (DR) and *false positive rate* (FPR). DR is the rate of bots correctly identified. It is defined by the formula:

$$DR = \frac{TP}{TP + FN} \tag{5}$$

TP is the number of true positives (bots classified as bots) and FN is the number of false negatives (bots classified as non-bots). FPR is defined as the rate of normal hosts mistakenly classified as bots. It is defined as follows:

$$FPR = \frac{FP}{FP + TN} \tag{6}$$

FP is the number of false positives (non-bots classified as bots) and TN is the number of true negatives (non-bots classified as non-bots).

Figure 8 shows Receiver Operating Characteristic (ROC) curves to display the relationship of DR and FPR. The horizontal axis represents the FPR and the vertical axis represents the DR. The chains are established by varying $dist\_th$ from 0.1 to 0.9. We can see that the best results have been achieved which attain the 100 % DR and 0 % FPR. It's observed that as $dist\_th$ increases, DR increases. But when $dist\_th$ increases to a certain value, DR remains 1. When

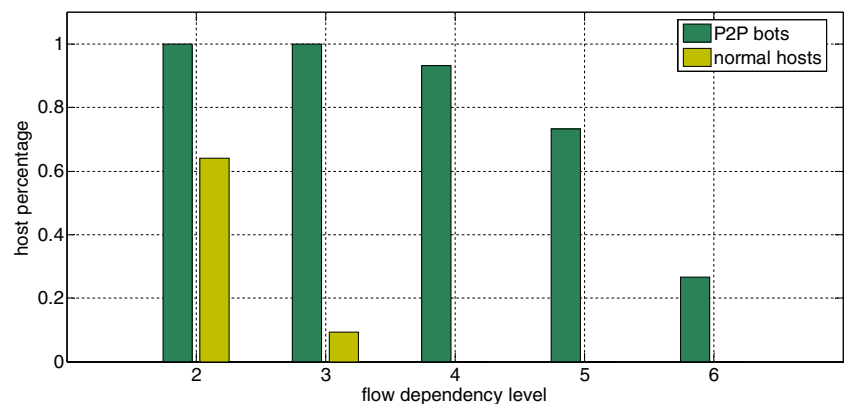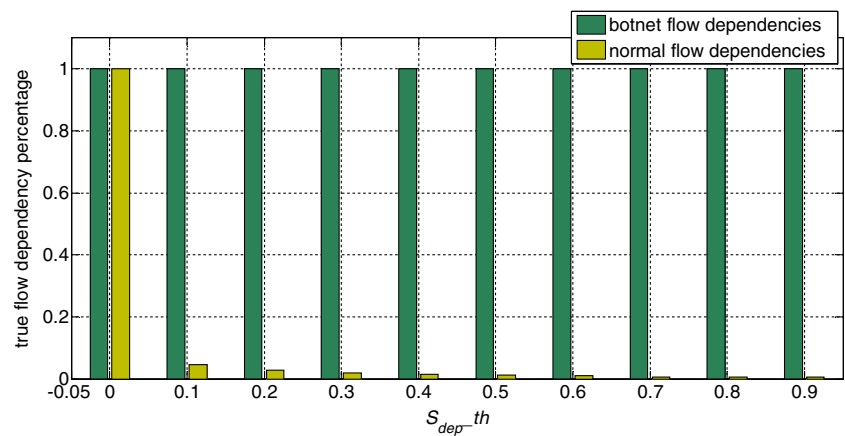**Fig. 5** Host percentage for different flow dependency level

Fig. 6 True flow dependency percentage for different S_{dep_th}

dist_th<=0.7, FPR is equal or nearly equal to 0, while dist_th>0.7, FPR increases sharply. This is due to normal host are different from each other and their distances are larger than dist_th when dist_th<=0.7. It's obvious that the larger the p is, the larger the DR. This is because larger p will lead to more common peers that different bots connect, thus distances among bots will smaller than dist_th and bots could be detected.

We have estimated the detection performance when there is no P2P botnet or there are two P2P botnets in the monitored network. It is possible that no host is compromised by P2P bot in the monitored network. In this case, we are concerned with the FPR. We applied our approach on the background traffic, where we set dist_th=0.7. No host is detected as bot and FPR=0 %. It is also possible that two P2P botnets compromise the hosts in the monitored network. We simulated the C&C traffic of two different botnets and the peers of the two botnets are different from each other. We applied our approach

on the data and set dist_th=0.7. The two botnets are detected. DR=100 % and FPR=0 %. Figure 9 shows the hierarchical clustering dendrogram of the two cases. Figure 9(a) shows that normal hosts could not form a dense cluster and Fig. 9(b) shows that there are two dense clusters which are respectively the two different P2P botnets.

5 Discussion

Our approach detects P2P botnets by discovering flow dependencies. The dependency flows have to happen together frequently. If these flows rarely happen, our approach may have difficulty to discover the flow dependency. The limitation is common in dependency discovery techniques based on network. The limitation can be mitigated by extending the time period of the collected data.

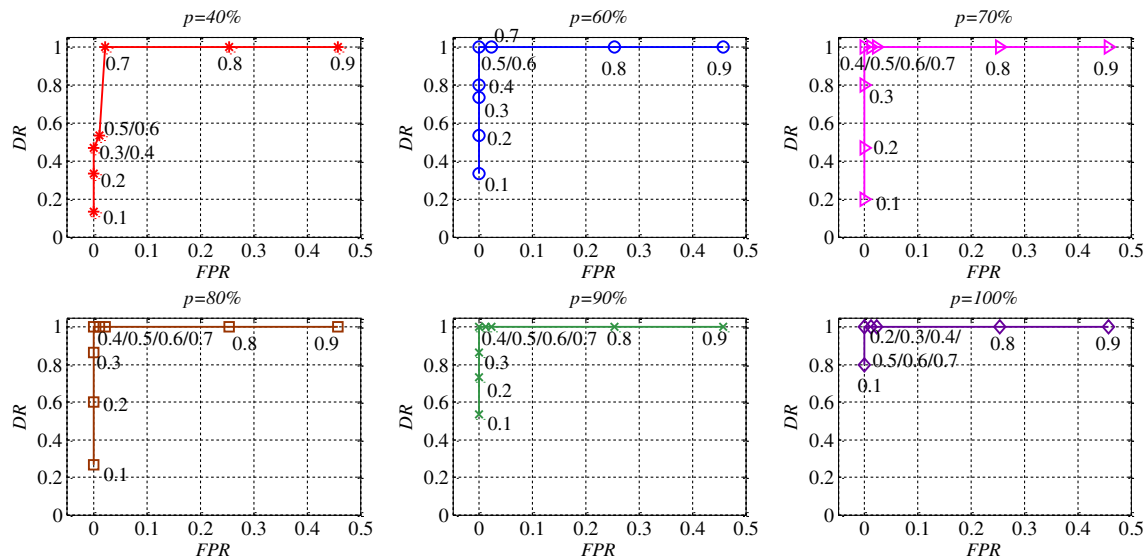**Fig. 7** Example of hierarchical clustering dendrogram
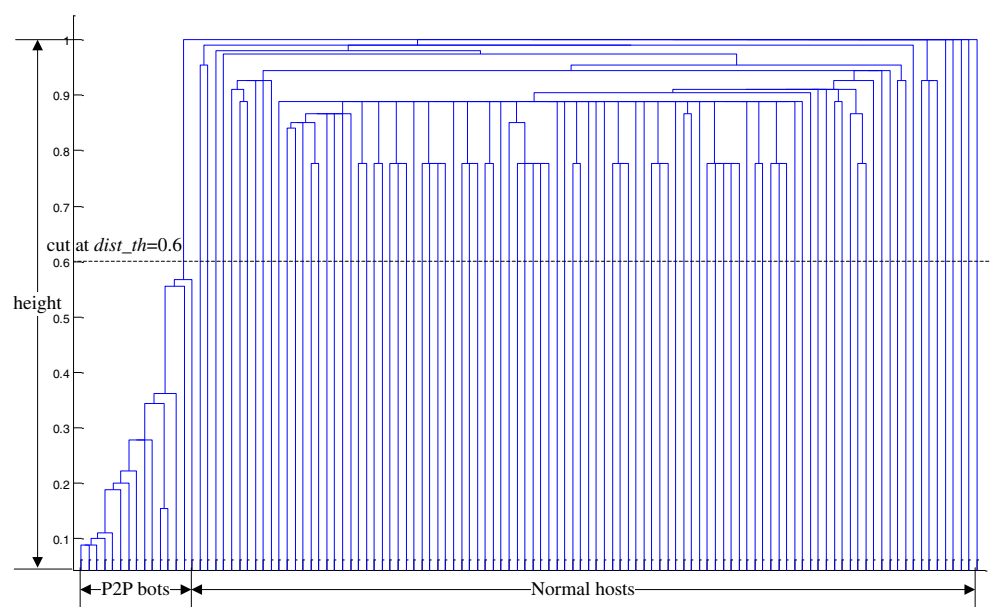
Fig. 8 ROC curves for different distance threshold *dist_th* and different common peer percentage *p* (the number aside the point is the value of *dist_th*)

The legitimate P2P hosts may frequently connect the peers in their neighbor lists to search for files or maintain the P2P network. They may have flow dependencies too. In our approach, two means can classify the legitimate P2P hosts and P2P bots. First, since the active time of the legitimate P2P hosts may be short, it is usually determined by users. For example, users may shut down the P2P application when they have downloaded the files. While the active time of P2P bots is long since they run as long as the underling system. Thus, the number of samples of legitimate P2P hosts' flow dependencies may be less than P2P bots' and the metric T_{ij} of legitimate flows may be smaller. Second, the cluster process can exclude legitimate P2P hosts. Since different legitimate P2P hosts tend to search for different files and the peers they connect have small possibility to overlay, so the distance of legitimate P2P hosts are large and they could not form a dense cluster.
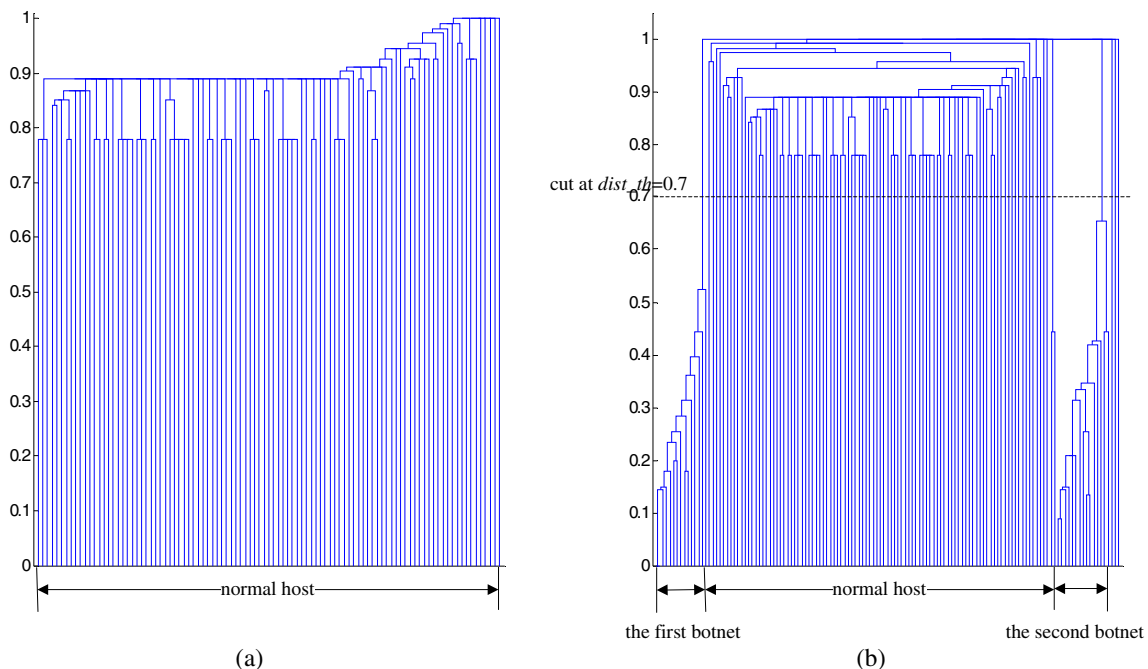

Fig. 9 Hierarchical clustering dendrogram of no botnet and two botnets

On the other hand, P2P bots in the same botnet usually connect common peers and their distances are small, thus they could form a dense cluster and can be detected.

6 Conclusion

In this work, we proposed a P2P botnet detection approach which focuses on C&C communications of P2P bots. Our approach first translates raw packets to TCP/UDP flows and filters unrelated flows. Then, it discovers flow dependencies using an algorithm which relies on time information and a large number of samples. Finally, it utilizes the true flow dependencies to cluster hosts and identify P2P bots. We implemented the approach and evaluated its performance. The experimental results confirm the fact that flow dependencies could be used to detect P2P bots and our approach can detect P2P bots with high detection rate and low false positive rate. In the future, we plan to extend our approach to detect P2P botnets in real time. In addition, extending our approach to detect cloud-based botnets is also our future work.

References

1. Barford P, Yegneswaran V (2006) An inside look at botnets. Special Workshop on Malware Detection, Advances in Information Security, Springer Verlag, Part III 27: 171–191
2. Lu W, Rammidi G, Ghorbani A (2011) Clustering botnet communication traffic based on n-gram feature selection. Comput Commun 34(3):502–514
3. Erdil D (2011) Simulating peer-to-peer cloud resource scheduling. Peer-to-Peer Netw Appl 5:219–230
4. Clark K, Warnier M, Brazier F (2011) BotClouds: the future of cloud-based Botnets. In Proceedings of the 1st International Conference on Cloud Computing and Services Science
5. Zeng Y, Hu X, Shin K (2008) Detection of botnets using combined host- and network-level information. In Proceedings of International Conference on Dependable Systems & Networks
6. Huang Z, Zeng X, Liu Y (2010) Detecting and blocking P2P botnets through contact tracing chains. Int J Internet Protoc Technol 5(1/2):44–54
7. Grizzard J, Sharma V, Nunnery C, Kang B, Dagon D (2007) Peer-to-peer botnets: overview and case study. In Proceedings of the 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)
8. Wang P, Wu L, Aslam B, Zou C (2009) A systematic study on peer-to-peer botnets. In Proceedings of the 18th International Conference on Computer Communications and Networks
9. Jacobs T, Pandurangan G (2012) Stochastic analysis of a churn-tolerant structured peer-to-peer scheme. Peer-to-Peer Netw Appl, in press
10. Stover S, Dittrich D, Hernandez J, Dietrich S (2007) Analysis of the Storm and Nugache Trojans: P2P is here. USENIX Mag 32 (6):18–27
11. Porras P, Saidi H, Yegneswaran V (2007) A multi-perspective analysis of the storm (Peacomm) worm. Computer Science Laboratory, SRI International, Tech Rep
12. Holz T, Steiner M, Dahl F, Biersack E, Freiling F (2008) Measurements and mitigation of peer-to-peer-based botnets: a case study on Storm worm. In USENIX Wksh. Large-Scale Exploits and Emergent Threats
13. Dittrich D, Dietrich S (2008) P2P as botnet command and control: a deeper insight. In Proceedings of 3rd International Conference on Malicious and Unwanted Software, MALWARE
14. Jang D, Kim M, Jung H, Noh B (2009) Analysis of HTTP2P botnet: case study Waledac. In Proceedings of the 2009 IEEE 9th Malaysia International Conference on Communications
15. Stock B, Gobel J, Engelberth M, Freiling F, Holz T (2009) Walowdac—analysis of a peer-to-peer botnet. In Proceedings of European Conference on Computer Network Defense
16. Sinclair G, Nunnery C, Kang B (2009) The Waledac protocol: the how and why. In Proceedings of the 4th International Conference on Malicious and Unwanted Software, MALWARE
17. Prasad K, Reddy A, Karthik M (2011) Flooding attacks to internet threat monitors (ITM): modeling and counter measures using botnet and honeypots. In Proceedings of International Journal of Computer Science & Information Technology
18. Udhayan J, Anitha R, Hamsapriya T (2010) Lightweight C&C based botnet detection using Aho-Corasick NFA. In Proceedings of International Journal of Network Security & Its Applications
19. Raahemi B, Zhong W, Liu J (2009) Exploiting unlabeled data to improve peer-to-peer traffic classification using incremental tri-training method. Peer-to-Peer Netw Appl 2:87–97
20. Gu G, Porras P, Yegneswaran V, Fong M, Lee W (2007) BotHunter: detecting malware infection through IDSdriven dialog correlation. In USENIX Secur Symp
21. Gu G, Perdisci R, Zhang J, Lee W (2008) BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In Proceedings of the 17th conference on Security Symposium
22. Nagaraja S, Mittal P, Hong C, Caesar M, Borisov N (2010) BotGrep: finding P2P bots with structured graph analysis. In Proceedings of the 19th USENIX Conference on Security
23. Francois J, Wang S, State R, Engel T (2011) BotTrack: tracking botnets using NetFlow and PageRank. Lect Note Comput Sci, Part I LNCS 6640:1–14
24. Coskun B, Dietrich S, Memon N (2010) Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In Proceedings of Annual Computer Security Applications Conference
25. Zhang J, Perdisci R, Lee W, Sarfraz U, Luo X (2011) Detecting stealthy P2P botnets using statistical traffic fingerprints. In Proceedings of IEEE/IFIP 41st International Conference on Dependable Systems and Networks
26. Kato D, Elkhiyaoui K, Kunieda K, Yamada K, Michiardi P (2010) A scalable interest-oriented peer-to-peer pub/sub network. Peer-to-Peer Netw Appl 4:165–177
27. Yen T (2011) Detecting stealthy malware using behavioral features in network traffic. Carnegie Mellon University, Dissertation
28. Jain A, Dubes R (1998) Algorithms for clustering data. Prentice-Hall
29. Jain A, Murty M, Flynn P (1999) Data clustering: a review. ACM Comput Surv 31(3):264–323
30. Sun X, Torres R, Rao S (2010) On the feasibility of exploiting P2P systems to launch DDoS attacks. Peer-to-Peer Netw Appl 3(1):36–51

Hongling Jiang is a PhD student at College of Information Technical Science, Nankai University. Her main research interests include P2P network, network security and data mining. Email: hellojhl@163.com. Mailing address: Xiqu apartment, Building No.8(Block C) 2–402, Weijin Road No.94, Tianjin, China, Zip code: 300071

Xiuli Shao is a professor at College of Information Technical Science, Nankai University. Her current research interests include data mining, network, cloud computing and parallel algorithm.