EURASIP Journal on Information Security
a SpringerOpen Journal

RESEARCH Open Access

PeerShark: flow-clustering and conversation-generation for malicious peer-to-peer traffic identification

Pratik Narang[1*], Chittaranjan Hota[1] and VN Venkatakrishnan[2]

Abstract

The distributed and decentralized nature of peer-to-peer (P2P) networks has offered a lucrative alternative to bot-masters to build botnets. P2P botnets are not prone to any single point of failure and have been proven to be highly resilient against takedown attempts. Moreover, smarter bots are stealthy in their communication patterns and elude the standard discovery techniques which look for anomalous network or communication behavior. In this paper, we present a methodology to detect P2P botnet traffic and differentiate it from benign P2P traffic in a network. Our approach neither assumes the availability of any 'seed' information of bots nor relies on deep packet inspection. It aims to detect the *stealthy* behavior of P2P botnets. That is, we aim to detect P2P botnets when they lie dormant (to evade detection by intrusion detection systems) or while they perform malicious activities (spamming, password stealing, etc.) in a manner which is not observable to a network administrator.

Our approach `PeerShark` combines the benefits of flow-based and conversation-based approaches with a two-tier architecture, and addresses the limitations of these approaches. By extracting statistical features from the network traces of P2P applications and botnets, we build supervised machine learning models which can accurately differentiate between benign P2P applications and P2P botnets. `PeerShark` could also detect *unknown* P2P botnet traffic with high accuracy.

Keywords: Botnets; Peer-to-peer; Machine learning; Security

1 Introduction

The past decade has seen the immense rise of the peer-to-peer (P2P) computing paradigm. In the beginning of the twenty-first century, the P2P architecture attracted a lot of attention of developers and end-users alike, with the share of P2P over the Internet in different continents being reported to be in the range of 45% to 70% [1]. As an increasing number of users got access to powerful processors, large storage spaces, and increasing bandwidths, P2P networks presented a great opportunity to share and mobilize resources.

Peer-to-peer overlay networks are distributed systems consisting of interconnected nodes which self-organize into network topologies. They are built with specific purposes of sharing resources such as content, CPU cycles, storage, and bandwidth. P2P networks have the ability to accommodate a transient population of nodes while maintaining acceptable connectivity and performance. They also operate without requiring the intermediation or support of a global centralized server or authority [2]. The construction of P2P networks is on the top of IP layer, typically with a decentralized protocol allowing 'peers' to share resources. The immense success of P2P applications is primarily attributed to the ease of resource sharing provided by them - be it in the form of music, videos, files (BitTorrent), or sharing of computing resources (SETI @ home project). Apart from these, P2P paradigm has also been widely deployed for IPTV (LiveStation) and voice over IP-based services (Skype[a]).

However, the P2P paradigm has been plagued with issues of privacy, security, and piracy to name a few [3-5]. Such issues, coupled with the advent of other popular content-sharing platforms (like YouTube and Netflix)

*Correspondence: p2011414@hyderabad.bits-pilani.ac.in
[1]BITS-Pilani, Hyderabad Campus, Hyderabad, Telangana 500078, India
Full list of author information is available at the end of the article

have led to decline in the share of P2P applications over the Internet to a mere 10% [6].

As P2P networks are inherently modeled without any centralized server, they lack a single point of failure [7]. This resilience offered by P2P networks has also attracted the attention of adversaries in the form of bot-masters (a.k.a. bot-herders). A 'bot' is a computer program which enables the operator to remotely control the infected system where it is installed. A network of such compromised end-hosts under the remote command of a master (i.e., the bot-master) is called a 'Botnet'. The ability to remotely command such bots coupled with the sheer size of botnets (numbering to tens of thousands of bots) gives the bot-masters immense power to perform nefarious activities. Botnets are employed for spamming, Bitcoin mining, click-fraud scams, distributed denial of service (DDoS) attacks, etc. on a massive scale, and generate millions of dollars per year in revenue for the bot-master [8]. Botnets are being touted as the largest threat to modern networks [9].

Botnets can either adopt a centralized or a distributed architecture for their command-and-control (C&C) communications. Earlier botnets were known to be centralized (e.g., Spybot, R-bot, Gaobot, etc.), and commonly used IRC or HTTP to receive commands from a single bot-master. But they suffer from a single point-of-failure since bringing down the bot-master effectively brought down the entire botnet. The distributed and decentralized P2P infrastructure has offered a lucrative alternative to bot-masters to build botnets which are not prone to any single point-of-failure. They have also proven to be highly resilient against takedown attempts [10].

Detection of P2P botnets by analysis of their network behavior has frequently utilized 'flow-based' approaches. Owing to certain limitations of these approaches in identifying modern P2P applications (discussed in Section 2.2), alternatives have been proposed in the form of super-flow-based and conversation-based approaches. However, these approaches are not yet mature and suffer from several drawbacks.

To this end, we present `PeerShark`, with a 'best of both worlds' approach utilizing flow-based approaches as well as conversation-based approaches in a two-tier architecture. `PeerShark` can differentiate between benign P2P traffic and malicious (botnet) P2P traffic, and also detect *unknown* P2P botnets with high accuracy. We envision `PeerShark` as a 'P2P-aware' assistant to network administrators wanting to segregate unwanted P2P traffic and detect P2P botnets.

`PeerShark` does not assume the availability of any 'seed' information of bots through blacklist of IPs. It does not rely on deep packet inspection (DPI) or signature-based mechanisms which are rendered useless by botnets/applications using encryption. It aims to detect the stealthy behavior of P2P botnets, that is, when they lie dormant in their rally or waiting stages (to evade intrusion detection systems which look for anomalous communication patterns) or while they perform malicious activities (spamming, password stealing, etc.) in a manner which is not observable to a network administrator.

`PeerShark` begins with the *de facto* standard 5-tuple flow-based approach and clusters flows into different categories based on their behavior. Within each cluster, we create 2-tuple 'conversations' from flows. Conversations are oblivious to the underlying flow definition (i.e., they are port- and protocol-oblivious) and essentially capture the idea of *who is talking to whom*. For all conversations, statistical features are extracted which quantify the inherent 'P2P' behavior of different applications, such as the duration of the conversation, the inter-arrival time of packets, the amount of data exchanged, etc. Further, these features are used to build supervised machine learning models which can accurately differentiate between benign P2P applications and P2P botnets.

To summarize our contributions:

- A 'best of both worlds' for P2P botnet detection which combines the advantages of flow-based and conversation-based approaches, as well as overcomes their limitations.
- Our approach relies only on the information obtained from the TCP/UDP/IP headers. Thus, it does not require DPI and cannot be evaded by payload-encryption mechanisms.
- Our approach can effectively detect activity of *stealthy* P2P botnets even in the presence of benign P2P applications in the network traffic.
- We extensively evaluate our system `PeerShark` with real-world P2P botnet traces. `PeerShark` could also detect *unknown* P2P botnets (i.e., those not used during the training phase) with high accuracy.

In the next section, we give a brief background of P2P botnets (Section 2.1) and discuss past efforts on P2P botnet detection (Section 2.2). In Section 3, we discuss the system design of `PeerShark`. Section 4 gives the details of design choices and implementation of `PeerShark`, followed by its evaluation in Section 5. In Section 6, we discuss about the limitations and possible evasions of `PeerShark`, and briefly mention about multi-class classification. We conclude in Section 7.

2 Background and related work

2.1 Background

A number of P2P-based botnets have been seen over the past decade, and a few of them have been taken down only recently with the combined effort of multiple nations. The massive Citadel botnet (a variant of the Zeus (or

'Gameover') P2P botnet) is believed to have stolen more than US $500 million from bank accounts over 18 months. It was reported in the past year that the 88% of the botnet has been taken down by the combined efforts of Microsoft and several security agencies and authorities of more than 80 countries [11]. However, recent reports claim that the botnet is on the rise again with a tweaked version being used to target a small number of European banks [12]. A variant of the Zeus P2P botnet also targeted Nokia phones using Symbian OS [13]. The botnet operated by installing a malware on the smart phone (via drive-by download from infected websites), which was used to steal the username-password credentials of the victim's online bank account transactions. The stolen details were forwarded to the bot-master.

Storm, a state-of-the-art botnet of its time, was known to comprise of at least a few million 'bots' when at its peak. It was involved in massive spamming activities in early 2007. Even the anti-spamming websites which targeted Storm came under a DDoS attack by the botnet [14]. Researchers have confirmed that the Waledac botnet is an improved version of the Storm botnet [15]. Waledac was capable of sending about 1.5 billion spam messages a day. It also had the capabilities to download and execute binaries and mine the infected systems for sensitive data. It was taken down in the year 2010.

A P2P bot's life cycle consists of the following stages:

- Infection stage, during which the bot spreads (this might happen through drive-by downloads, a malicious software being installed by the end-user, infected USB sticks, etc.)
- Rally stage, where the bot connects with a peer list in order to join the P2P network
- Waiting stage, where the bot waits for the bot-master's command (and does not exhibit much activity otherwise)
- Execution stage, in which it actually carries out a command, such as a denial of service (DoS) attack, generate spam emails, etc.

To evade detection by intrusion detection systems (IDSs) and firewalls, botnets tend to keep their communication patterns (with the bot-master or other bots) quite stealthy. IDSs and Firewalls, which rely on anomalous communication patterns to detect malicious behavior of a host, are not very successful in detecting such botnets. Generating little traffic, such bots 'lie low' and thus pass under the radars of IDSs/firewalls.

With the advent of the Internet of Everything, the possibility of malware taking control of 'smart' appliances such as television, air-conditioners, refrigerators, etc. will not be limited to theory. In fact, there have been recent reports of 'smart' refrigerators and cars being hacked, and wi-fi-enabled LED bulbs having security weaknesses [16]. As the creators of botnets continue to adopt innovative means in creating botnets, detection of stealthy botnets continues to be a challenging paradigm.

2.2 Related work

Most prior work has either focused on P2P traffic classification from the perspective of a more general problem of Internet traffic classification [17-19], or has given special attention to detection of botnets (centralized or distributed) in Internet traffic [20-22]. The challenging context of detection of stealthy P2P botnets in the presence of benign P2P traffic has not received much attention.

Initial work on detection of P2P botnets involved signature-based and port-based approaches [23]. Solutions such as BotMiner [20] rely on DPI which can easily defeated by bots using encryption. Some of the recent work has used supervised [24,25] and unsupervised [26,27] machine learning approaches and other statistical measures [28]. PeerRush [24] created 'application profile' from the network traces of multiple P2P applications. Their work utilized payload sizes and inter-packet delays to categorize the exact P2P application running on a host. The approach of Zhang et al. [26,27] used 'control flows' of P2P applications to extract statistical fingerprints. P2P bots were identified based on certain features like fingerprint similarity, number of overlapping contacts, persistent communication, etc. However, their work can detect P2P bots inside a network only when there are multiple infected nodes belonging to the same botnet. Yen and Reiter [28] attempt to segregate P2P bots from benign P2P apps based on metrics like the volume of data exchanged and number of peers contacted. Unfortunately, their features are not sufficient to correctly differentiate P2P bots and apps. Furthermore, their approach fails to detect bots when bots and apps run on the same machine.

Most of the past works have employed the classical 5-tuple categorization of network flows. Packets were classified as 'flows' based on the 5-tuple of source IP, source port, destination IP, destination port, and transport layer protocol. Flows have bidirectional behavior, and the direction of the flow is decided based on the direction in which the first packet is seen. This traditional definition of flows has been greatly employed and has seen huge success in the problems of Internet traffic classification [29] and even in the early days of P2P traffic classification [30]. This definition relies on port number and transport layer protocol. The latest P2P applications as well advanced P2P bots are known to randomize their communication port(s) and operate over TCP as well as UDP. Such applications will not be well-identified by these traditional approaches. Since such a behavior is characteristic of only the latest variants of P2P applications (benign or malicious), it is obvious that past research did not touch upon this aspect.

In response to this, a recent work [22] has used the 2-tuple 'super-flows' based approach with a graph-clustering technique to detect P2P botnet traffic. Although authors in [22] presented interesting insight and obtained good accuracy in detecting the traffic of two P2P botnets, their approach has certain limitations. Their work evaluates the detection of P2P botnets only with regular web traffic (which was not analyzed for the presence or absence of regular P2P traffic). This is a serious limitation because P2P botnet traffic (quite obviously) exhibits many similarities to benign P2P traffic. Furthermore, graph-based approaches work on a 'snapshot' of the network. P2P networks have high 'churn-rate' (joining and leaving of peers). Since the network is changing fast, any solution suggested for a 'snapshot' of the network would quickly become obsolete. Thus, their approach would fail in the presence of benign P2P traffic. Distinguishing between hosts using regular P2P applications and hosts infected by a P2P botnet would be of great relevance to network administrators protecting their network.

Another recent work [31] has seen the use of 'conversation-based' approach in the P2P domain, but for a different problem, namely, the detection of overlapping P2P communities in Internet backbone. Their work does not focus on identification of any specific P2P application—whether malicious or benign.

A preliminary version of our work [32] adopted conversation-based approach for the detection of P2P botnets. In [32], we looked for high-duration and low-volume conversations in order to separate P2P bots from apps and used their timing patterns as a distinguishing feature for categorization of different P2P apps and bots. None of the past works employing super-flow or conversation-based approaches ([22,31,32]) address an inherent drawback of these approaches: they fail to detect botnet activity if P2P bots and apps are running on the same machine (which might be a rare scenario, but cannot be ruled out nonetheless). This is because conversations (or super-flows) try to give a bird's eye view of the communications happening in the network and thus miss certain finer details.

3 System design

3.1 System overview

P2P botnets engage in C&C using custom or well-known P2P protocols. As a result, their traffic can blend with benign P2P traffic flowing in a network and thus pass undetected through IDSs or firewalls.

`PeerShark` uses a two-tier approach to differentiate P2P botnets from benign P2P applications. The first phase clusters P2P traffic-flows based on the differing behavior of different applications. In the second phase, conversations are created from flows within each cluster. Several statistical features are extracted from each conversation and are used to build supervised machine learning models for the detection of P2P botnets. In Section 5, we will evaluate the effectiveness of our detection scheme with traces of known and *unknown* (i.e., not used in the training phase) P2P botnets.

Figure 1 gives the architecture of `PeerShark`. The system design of `PeerShark` is explained here:

3.2 Flow-clustering phase

Flow-based analysis has been the *de facto* standard for Internet traffic classification and has yielded great success in the past [29,30]. Typically, 'flows' are constructed based on the 5-tuple: `<Src. IP, Dest. IP, Src. port, Dest. port, Proto>` (where 'Src.' stands for 'source', 'Dest.' stands for 'destination', and 'Proto' implies the transport layer protocol (TCP or UDP)). The direction of the flow is determined by the direction in which the first packet is seen. An important difference between flows of P2P applications and traditional client-server applications is that P2P traffic is inherently bi-directional in nature.

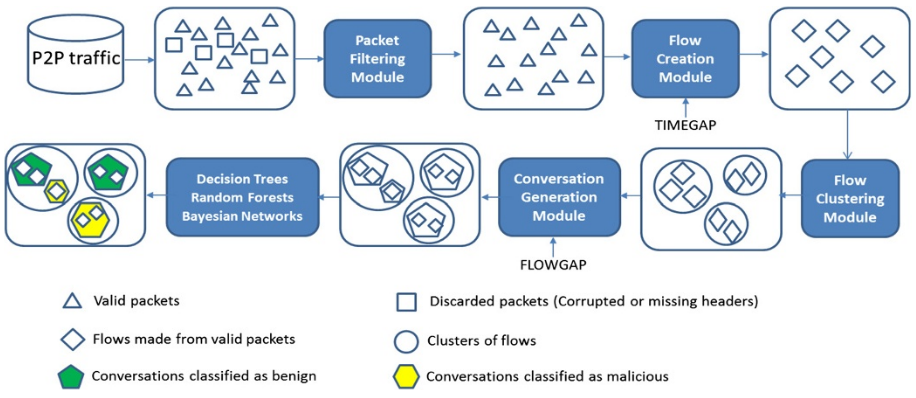
P2P traffic
Packet Filtering Module
Flow Creation Module
TIMEGAP
Flow Clustering Module
Conversation Generation Module
FLOWGAP
Decision Trees Random Forests Bayesian Networks
Valid packets
Discarded packets (Corrupted or missing headers)
Flows made from valid packets
Clusters of flows
Conversations classified as benign
Conversations classified as malicious

Figure 1 PeerShark: architecture.

This differentiating factor has been leveraged by some recent works [24,25] as well.

We leverage the bidirectional behavior of P2P traffic to segregate flows into different clusters based on their differing behavior. The correct classification (in terms of benign or malicious application) is not a concern at this point. At this stage, we want to separate flows into different clusters based on their behavior. As an example, we observed a peculiar behavior in the network traces of the Zeus botnet. Flows between two hosts switched between TCP and UDP. However, the communication over TCP was always fast but short-lived (a few hundred packets exchanged within a matter of seconds), while communication over UDP was stealthy and long-lived (two or three packets exchanged in half-an-hour duration). At this stage of flow clustering, these differing flows of Zeus are expected to get separated into different clusters.

For the purpose of clustering, we extract a five-feature vector for every flow: `Protocol`, `Packets per second (f/w)`, `Packets per second (b/w)`, `Avg. Payload size (f/w)`, and `Avg. Payload size (b/w)`, with 'f/w' and 'b/w' signifying the forward and the backward direction of the flow, respectively. The primary motivation behind the choice of these features is to exploit the bidirectional nature of P2P traffic and separate flows based on their 'behavior' in terms of the transport layer protocol used and packets and payload exchanged. A more detailed discussion on the choice of features and clustering algorithm will follow in Section 4.

3.3 Conversation-generation phase

Once a bot-master infects a particular machine, it is in the prime interest of the bot-master to not lose connectivity with his bots. The bot-peers near to each other in the P2P overlay network maintain regular communication amongst themselves to check for updates, to exchange commands, and/or to check if the peer is alive or not. If such messages are exchanged very frequently, the bots are at a risk of getting detected by IDS/firewalls monitoring the network. Hence, the communication between the bot-master and his bots, or that of bots amongst themselves, is expected to be low in volume (note here that this usually corresponds to the rally and waiting stages; 'execution' stage can be aggressive or stealthy depending upon the activity for which the bots are used; DDoS attack can be quite aggressive, while password stealing may remain stealthy).

Since certain botnets (and even benign P2P applications) are known to randomize their port numbers over which they operate, the classical 'flow' definition will not be able to give a clear picture of the activity a host is engaged in. The traditional 'flow' definition will create multiple flows out of what is actually a single conversation happening between two such peers (although happening on different ports) and thus give a false view of the communications happening in the network. To get a bird's eye view of the *conversations* happening between the P2P, hosts can be beneficial for a network administrator to hunt for malicious conversations between the bots.

As has been explained before, the present works utilizing conversation-based (or super-flow based) approaches [22,32] did not have the capabilities to detect a system infected by a P2P bot if it is running benign P2P applications as well. The main reason behind this flaw was that conversations attempt to provide a bird's eye view of the network activity to the network administrator but miss out certain finer details: since all flows between two IPs are aggregated into a single conversation, this approach will create a single conversation for two IPs having malicious as well as benign flows between them and thus fail to detect the malicious traffic. To combat this drawback, we use flow-clustering in the first phase which separates flows into different clusters based on their differing behavior. By this, we attempt to perform a coarse separation of P2P apps and bots based on their differing behavior.

In the second phase, we create conversations from flows *within each cluster*. Note that in our previous work and in other works, conversations were created by aggregating all flows/conversations between `IP1` and `IP2` into a single 'conversation'. Here, we limit conversation creation to the flows within each cluster. Since flows within the same cluster have similar behavior, we are creating conversations out of only those flows which show similar behavior. Thus, the drawback of aggregating *all* flows/conversations between two IPs into a single conversation (and thus missing out finer details) is addressed.

Furthermore, all P2P applications—whether malicious or benign—operate with their 'app-specific' *control messages* which are used by peers to connect to the P2P network, make file searches, leave the network, etc. Since each application has its own specific control messages, we exploit the timing patterns of these control messages to differentiate between P2P applications by considering the median value of the *inter-arrival time* of packets for each P2P application. Moreover, as has been explained before, bot traffic tends to be stealthy. Hence, bot conversations are expected to have higher inter-arrival time of packets than benign P2P conversations.

In summary, after creating conversations from flows, we extract four statistical features from each conversation:

1. The duration of the conversation
2. The number of packets exchanged in the conversation
3. The volume of data exchanged in the conversation
4. The median value of the inter-arrival time of packets in that conversation

These features are then used to build supervised machine learning models to differentiate between benign and malicious P2P traffic. More details will follow in the next section.

4 Design choices and implementation details

In this section, we present the implementation aspects and design choices of `PeerShark` in detail:

4.1 Data

This work uses data of benign P2P applications and P2P botnets obtained from two different sources. The data of four benign P2P applications, namely uTorrent, eMule, Vuze, and Frostwire, and the data of three P2P botnets, namely Storm, Waledac, and Zeus, was obtained from the University of Georgia [24]. The data for P2P applications was generated by the authors in [24] by running those applications in their lab environment for a number of days, using AutoIt scripts to simulate human-user activity on the P2P hosts. The data of P2P botnets corresponds to real-world traces obtained by them from third parties. The dataset of P2P botnet Nugache also corresponds to real-world traces obtained from authors of [33]. Altogether, we used four bots and four apps for this work.

As explained in the previous section, network traces of each application were parsed to create flows and further generate conversations (more details will follow in this section). The conversations thus obtained were labeled to create a 'labeled dataset' for training and testing purposes. For all conversations corresponding to P2P applications, we use the label 'benign'.

In the network traces of each of the P2P botnets, there are certain 'known malicious hosts' (Storm had 13, Waledac had 3, Zeus had 1, and Nugache had 4 'known malicious hosts'). However, it is not known whether the other IP addresses seen in the network traces are benign or malicious[b]. Hence, in order to create a 'ground truth' for our evaluation, we treat a conversation as 'malicious' if either of the IPs (either source or destination) is known to be 'malicious'. If none of the IPs in the conversation are known to be malicious, we treat that conversation as benign. Full details of this dataset are given in Table 1. Our training/testing datasets are representative of the real-world where the majority of traffic flowing in any network is benign. Our datasets contain more than 90% benign traffic.

4.2 Packet filtering module

`PeerShark` operates on a dump of network traces. The first module takes network logs in the form of raw packet data (`pcap` files) as input and parses them using the `Libpcap` library. The module reads each packet and isolates those which have a valid IPv4 header. For the purpose of data sanitization, all packets without a valid IPv4 header are deemed invalid and discarded. The packets are further filtered to retain only those packets which have a valid TCP or UDP header and a non-zero payload. From each packet, the Timestamp, Source IP, Source port, Destination IP, Destination port, and Payload size are extracted and stored for future use. In addition to these, we also extract the TCP flags (`SYN, ACK, RST, FIN` etc.) for all TCP packets.

This module is algorithmically explained in Algorithm 1.

Table 1 Dataset details

Name	Number of flows	Number of conversations	Purpose
eMule	413,995	293,704	Train/Test
uTorrent	1,409,291	458,624	Train/Test
Vuze	1,207,963	603,145	Train/Test
Frostwire	890,300	234,335	Train/Test
Storm	95,316	59,157	Train/Test
Waledac	81,778	5,765	Train/Test
Zeus	43,593	2,751	Unseen test
Nugache	51,428	49	Unseen test

Algorithm 1 Packet filtering module

```
 1: procedure FILTERPACKETS(packetCapture)
 2:     ArrayList < ModifiedPkt > filteredPkts;
 3:     for Packet p in packetCapture do
 4:         time ← p.getTimestamp();
 5:         if p has IPHeader then
 6:             ip ← p.getIPHeader();
 7:             IP1 ← ip.getSourceIP();
 8:             IP1_port ← ip.getSourcePort();
 9:             IP2 ← ip.getDestIP();
10:             IP2_port ← ip.getDestPort();
11:             if p has TCPheader or UDPheader then
12:                 header ← p.getTransportHeader();
13:                 pSize ← header.getPayloadSize();
14:                 if payloadSize not null or zero then
15:                     nextPkt ← ModifiedPkt(IP1, IP1_port,
16:                                  IP2, IP2_port, pSize, time);
17:                     filteredPkts.add(nextPkt);
18:                 end if
19:             end if
20:         end if
21:     end for
22:     return filteredPackets;
23: end procedure
```

4.3 Flow creation module

The output of the packet filtering module is fed into this module to generate bidirectional flows. Each flow is identified by `<Src. IP, Dest. IP, Src. port,`

`Dest. port, Proto>`, as explained before. All packets corresponding to a 5-tuple are gathered and sorted based on timestamp. Flows are created based on the 5-tuple and a `TIMEGAP` value. `TIMEGAP` is defined as the maximum permissible inter-arrival time between two consecutive packets in a flow, beyond which we mark the latter packet as the beginning of a new flow. For TCP flows, in addition to `TIMEGAP` criteria, we initiate a flow only after the regular TCP three-way handshake has been established. The termination criteria for a TCP flow is met either by `TIMEGAP` or the TCP close connection sequence (in terms of `FIN` packets or `RST` packets), whichever is encountered first. In case of UDP's (virtual) flows, only the `TIMEGAP` can be employed. The module is explained algorithmically in Algorithm 2 (TCP connection establishment or termination sequences are skipped from the algorithm for the sake of brevity).

Note that `TIMEGAP` is a 'tunable' parameter, which must be decided by a network administrator based on his understanding of his network. From our experiments, we observed that a high `TIMEGAP` value was more suitable since many bots exchanged very few packets after long intervals of time. A low `TIMEGAP` value would imply just one or two packets per flow, which will be useless to extract any useful statistical metrics. We used a `TIMEGAP` value of 2,000 seconds.

Algorithm 2 Flow creation module

```
1:  procedure CreateFlows(filteredPackets)
2:      ArrayList < Flow > initFlowList;
3:      ArrayList < PacketGroup > pgList;
4:      pgList ← filteredPkts.groupPktsBy5tuple();
5:      for PacketGroup pg in pgList do
6:          sort packets in pg by timestamp;
7:          nextFlow ← Flow(NULL);
8:          for Packet p in pg do
9:              if p.timestamp between
10:                 (nextConv.start - TIMEGAP) &&
11:                 (nextConv.end + TIMEGAP) then
12:                     nextFlow.addPacket(p);
13:             else
14:                     nextFlow ← Flow(p);
15:                     initFlowList.add(nextFlow);
16:             end if
17:         end for
18:     end for
19:     return initFlowList;
20: end procedure
```

For every flow, a number of statistical features are extracted, such as:

1. Transport layer protocol
2. Avg. payload (forward and backward)
3. Total payload exchanged
4. Packets per second (forward and backward)
5. Bytes per second (forward and backward)
6. Total number of packets exchanged
7. Duration of the flow
8. Median of inter-arrival time of packets

Some of these are utilized for the flow clustering module, while some are retained for later use in the conversation generation module.

4.4 Flow clustering module

The flow clustering module aims to separate flows into different clusters based on their differing behavior. In order to keep our approach suitable for large networks, the choice of a fast clustering algorithm was necessary. At the same time, we would want the number of clusters to be chosen automatically as per the behavior seen in the data. To this end, we use the X-means clustering algorithm [34]. X-means algorithm is a variant of K-means which scales better and does not require number of clusters to be supplied by the user.

Clustering is an unsupervised learning approach. But since we had a labeled dataset available to us, we adopted the route of classes-to-clusters evaluation. In classes-to-clusters evaluation, the class label is initially ignored and the clusters are generated. Then, in a test phase, class labels are assigned to clusters based on the majority value of the class attribute in that cluster. Further, the classification error is computed, which gives the percentage of data points belonging to the wrong cluster. This classification error gives us a rough idea of how close the clusters are to the actual class labels of the instances.

Since the transport layer protocol naturally distinguishes between TCP and UDP flows, it was a natural choice for a feature to be used for clustering. In order to choose the rest of the features, we began with a superset S_n of n pair of features which represent bidirectional behavior of flows, such as: bytes per second (forward) & bytes per second (backward), packets per second (forward) & packets per second (backward), avg. payload (forward) & avg. payload (backward), etc. We computed the classification error obtained from classes-to-cluster evaluation with S_n and recomputed it for all S_{n-2} sets by removing one pair of features at a time (note that all features occur in pairs of 'forward' and 'backward'). The classification errors for S_n and all sets of S_{n-2} were compared. The set with the lowest classification error was chosen. If that set was S_n, the computation terminated. Else, the set S_{n-2} with lowest classification error was chosen, and the process was repeated until the classification error did not drop further.

The final set of features thus obtained were: `protocol, packets per second (f/w), packets per second (b/w), avg. payload size (f/w), and avg. payload size (b/w).`

The classification error obtained with these features was 50.1163%. Irrespective of the number of features used, the X-means algorithm always created four clusters from the training data. Since our dataset is a representative dataset with the 4 P2P apps having the majority of instances, an outcome of four clusters was quite expected.

4.5 Conversation generation module

Within each cluster, the flows created previously are aggregated into conversations. Conversations are generated for a `FLOWGAP` value as desired by a network administrator. Flows between two IPs are aggregated into a single conversation if the last packet of flow 1 and first packet of flow 2 occur within the `FLOWGAP` time. Here, the network administrator is given the flexibility to mine data for the time period desired by him, say 2 h, 24 h, etc., thus giving him visibility into the network logs as required. Such flexibility is especially valuable for bots which are *extremely stealthy* in their communication patterns and exchange as low as a few packets every few hours. For this evaluation, the value being used is 1 h. This module is explained algorithmically in Algorithm 3.

Algorithm 3 Conversation generation module

```
1:  procedure CONVOGENERATION(initFlowList, FLOWGAP)
2:      ArrayList < Conversation > ConvoList;
3:      ArrayList < ConversationGroup > cgList;
4:      cgList ← initFlowList.groupByIPpair();
5:      for ConversationGroup cg in cgList do
6:          sort IPpairs in cg by timestamp;
7:          nextConvo ← Conversation(NULL);
8:          if cg.timestamp between
9:              (nextConvo.start - FLOWGAP) &&
10:             (nextConvo.end + FLOWGAP) then
11:                 nextConvo.addConvo(cg);
12:         else
13:                 nextConvo ← Conversation(cg);
14:                 ConvoList.add(nextConvo);
15:         end if
16:     end for
17:     return ConvoList;
18: end procedure
```

Using the features extracted from every flow, we extract fresh features for every conversation: number of packets, conversation volume (summation of payload sizes), conversation duration, and the median value of inter-arrival time of packets in the conversation. The reasons behind choosing these features have already been explained in the previous section.

The median of inter-arrival time of packets was observed to be a better metric than the mean because `PeerShark` aggregates several flows into a single conversation as per the `FLOWGAP` value supplied. In such a scenario, it is quite possible that flow 1 and flow 2 get merged into a single conversation while the last packet of flow 1 and first packet of flow 2 occur several minutes (or even hours) apart. This will skew the mean value, and the use of median value was found to be more suitable from our experiments.

5 Results and evaluation

5.1 Training and testing datasets

The labeled data of all four P2P apps along with Storm and Waledac was used for training and testing purposes. Altogether, the dataset contained 1,654,730 conversations (1,589,808 benign and 64,922 malicious). This dataset was split into training and testing datasets in a 2:1 ratio. The training dataset had 1,092,122 conversations (1,049,242 benign and 42,880 malicious), and the test split contained 558,348 conversations (540,566 benign and 22,042 malicious). The training as well as test splits contain more than 90% benign data. Although such class imbalance makes the task of detecting P2P botnets more challenging, this ratio is representative of the real-world scenario where majority of traffic flowing in a network is benign.

After building the models on the training set and testing them with the test set, we evaluate our models against *unseen* botnet datasets (i.e., not used in training) of Zeus and Nugache. Since the network traces of Zeus contain only one ‘known malicious host’, they are not adequate to train detection models. Similarly, although traces of Nugache contain data of four malicious hosts, the dataset is very small (see Table 1) and thus not suitable to build detection models. Nevertheless, they can be used to evaluate `PeerShark`’s capability on profiling unknown P2P bots.

5.2 Classifiers

The training and testing of our models was performed using the Weka machine learning suite [35].

Our training and testing dataset contains a high ‘class imbalance’ towards the benign class. This imbalance was kept on-purpose in order to have a dataset representative of real Internet traffic. Hence, we need to utilize learning algorithms which can handle class imbalance. Moreover, the classifiers must be fast to train. We use J48 decision trees, which are simple to train and fast classifiers and can handle class imbalance problems well.

Second, we use random forests. Random forests create an ensemble of decision trees and output the final class that is the mode of the classes output by individual trees. It randomly chooses a set of features for classification for each data point and uses averaging to select the most important features. It can effectively handle overfitting of data and run efficiently on large datasets.

Along with tree-based classifiers, we use a stochastic learning algorithm—Bayesian network. Bayesian networks are probabilistic graphical models that can handle class imbalance, missing data and outliers quite well. They can also identify relationships amongst variables of interest.

Ten-fold stratified cross-validation was used over the training dataset to build detection models with these classifiers. The models were tested with the test dataset. These results are presented in Table 2.

5.3 Evaluation metrics

We use established metrics such as precision, recall, and false positive rate for evaluation of our approach. We briefly define them here:

- *Precision* is the ratio of the number of relevant records retrieved to the total number of relevant and irrelevant records retrieved. It is given by $\frac{\mathrm{TP}}{\mathrm{TP+FP}}$.
- *Recall* (or true positive rate) is the ratio of the number of relevant records retrieved to the total number of relevant records in the complete dataset. It is given by $\frac{\mathrm{TP}}{\mathrm{TP+FN}}$.
- *False positive rate* is given by the total number of false positives over the total number of true negatives and false positives. It can be expressed mathematically as $\frac{\mathrm{FP}}{\mathrm{FP+TN}}$.

TP stands for true positive, TN stands for true negative, FP stands for false positive, and FN stands for false negative.

As the results in Table 2 show, `PeerShark` could consistently detect P2P bots with high accuracy and very low false positives. We emphasize that these results are over the test set and not the training data. All three classifiers achieved high precision and recall. Since the train and test datasets have higher number of 'benign' instances, benign traffic is naturally classified with much higher accuracy. However, even in the presence of more than 90% benign traffic, false positive rate for the 'malicious' class (i.e., benign conversations incorrectly classified as malicious) was quite low. This is important for any malicious traffic classifier since it must not create false alarms by classifying benign traffic as malicious.

5.4 Testing on unseen data

To further evaluate the effectiveness of `PeerShark` on profiling new and unseen P2P botnet traffic, we use the three models trained above and test them against the conversations of Zeus and Nugache which were not used in training the models. It is evident from the results presented in Table 3 that the approach adopted by `PeerShark` is effective and generic enough to detect unseen P2P botnets with high accuracy.

An ardent reader may note that the detection accuracy achieved for the validation set of Nugache and Zeus is higher than that of the test set composed of Storm and Waledac. We would like to make two points in this regard. Firstly, our training and testing datasets were highly variegated, being composed of four benign P2P applications and two P2P botnets, with a huge number of flows/conversations of each and the proportion of benign traffic being more than 90%. With such variety, the results presented by us are indicative of what one might expect in a real-world scenario. But we did not have the same luxury with the validation datasets. Had there been more variety in the network traces of Nugache and Zeus, it is quite possible that the detection rate would have been slightly lower.

Secondly, we made an interesting observation in the network traces of Storm and Nugache. Nugache and Storm have been hailed as cousins [36]. Nugache which became well known amongst analysts has a 'TCP port 8 botnet' [37] since it used the unassigned port 8 over TCP for several communications. However, while examining the network traces of Storm, we observed some activity over TCP on port 8. This is not a typical behavior of Storm. We suspect that these hosts believed to have been infected by Storm also had Nugache infection on them (although we do not have the facility to verify this). This could possibly explain high detection rate for Nugache.

6 Discussion

6.1 Possible evasions

`PeerShark` clusters flows based on their behavior and then forms conversations from the flows within a cluster. Since it employs both approaches, `PeerShark` overcomes many limitations of past efforts. P2P bots which

Table 2 Performance of classifiers on test data

	Decision trees			Random forests			Bayesian network		
Class	Precision	Recall	FP rate	Precision	Recall	FP rate	Precision	Recall	FP rate
Malicious	95.3%	93.4%	0.2%	95.3%	94.9%	0.2%	91.9%	88.4%	0.3%
Benign	99.7%	99.8%	6.6%	99.8%	99.8%	5.1%	99.5%	99.7%	11.6%

Table 3 Performance of classifiers on unseen P2P botnets

	Decision trees			Random forests			Bayesian network		
	Classified malicious	Classified benign	Accuracy (%)	Classified malicious	Classified benign	Accuracy (%)	Classified malicious	Classified benign	Accuracy (%)
Zeus	2,696	55	98%	2,717	34	98.76%	2,660	91	96.69%
Nugache	42	7	85.71%	43	6	87.76%	48	1	97.96%

randomize port numbers and switch between TCP/UDP distort the network administrator's view of the actual communication happening between two hosts. Flow-based techniques are insufficient for such cases. The conversation-generation scheme adopted by `PeerShark` can effectively address this issue by aggregating flows of the same cluster into a single conversation. Previous works employing conversation-based approaches could not separate P2P bots and apps on the same machine. By segregating flows into different clusters based on their behavior, the approach adopted by `PeerShark` can effectively separate P2P bot and app traffic running on the same machine.

However, in order to differentiate between benign and malicious P2P traffic, `PeerShark` relies on 'behavioral' differences in the flows/conversations of P2P bots and apps. If two bot-peers mimic a benign P2P application, our system may fail in detecting them accurately. To elaborate more on this, consider the following scenario: a bot-master could configure his bots to engage in occasional file-sharing activity with each other on a regular P2P network (like eMule, uTorrent, etc.). Seeing such benign-like activity on a host, `PeerShark` is likely to mis-classify the flows/conversations between them as 'benign'. But, since occasional file-sharing by bots involves network bandwidth usage (and, say, accompanying monetary charges), such an activity has the likelihood of getting noticed by the owner of the system or a network administrator and is thus fraught with risks for the bot-master. Nonetheless, we admit that it is possible for bot-masters to design smarter bots which mimic benign-like behavior and/or add noise (or randomness) to their communication patterns and thus evade the present detection mechanism of `PeerShark`. Authors in [38] argue on a similar case by building a botnet with Skype and validate their assertions with simulations.

Furthermore, assume the case of a peer A which is engaged in P2P file sharing with a benign peer B, but is also covertly a part of a botnet and is engaged in exchanging command-and-control with a malicious peer C. PeerShark will see these as two conversations, namely A to B and A to C. Since PeerShark regards a conversations as 'malicious' even if either of the IPs (source or destination) is malicious, A to C is identified as 'malicious' without hesitation. But since the conversation between A and B also involves one malicious peer (namely A), this conversation will also be tagged as 'malicious'. Although it is a limitation on the part of PeerShark to regard that peer B is engaged in a malicious conversation, it is not a serious shortcoming. Since peer B is, as a matter of fact, conversing with a peer which has been compromised, it runs high risk of being infected in the future. Thus, raising an alarm for conversations between A and B (apart from those between A and C) is not completely unwarranted.

Finally, as described in the packet filtering module (Section 4), `PeerShark` discards all packets having a zero payload. This was necessary to remove corrupted packets and sanitize the network traces obtained from authors in [24]. However, such an approach has an inherent drawback of dropping all legitimate packets with zero payload, such as TCP connection establishment (`SYN`) packets. It can be exploited by an active adversary who may use zero payload TCP packets (`SYN` or `ACK` packets) to exchange simple commands between bots.

6.2 A note on multi-class classification

In the preliminary version of our work [32], we had attempted a multi-class classification approach which could categorize the exact P2P application running on a host. Initially, we attempted multi-class classification for this work as well. The detection accuracy and false positive rate for P2P botnets was nearly the same as that of binary classification approach. However, *within* the benign P2P applications, we saw a false positive rate of 2% to 10%. Although in terms of percentage, the false positive rate is not high, it comes up to thousands of conversations in terms of the actual number of conversations. In particular, we saw many misclassified conversations between Vuze and Frostwire. Such misclassification may be attributed the fact that the majority of our benign P2P data consists of 'torrent' based applications. uTorrent, Vuze, and Frostwire are all 'torrent' based (while eMule is not). Thus, it is quite natural that conversations of one torrent-based app were misclassified as that of another. However this distribution is representative of the real world where the share of P2P in Internet traffic is dying, and BitTorrent is the only aspect of P2P which continues to dominate [6].

New P2P botnets continue to be seen every year. Many of these are just variants or 'tweaks' of older ones. For example, Citadel used a tweaked variant of Zeus [12]. A multi-class approach will only be able to correctly classify those botnets for which it has been trained. It will either miss new variants or call them as 'unknown' (as in [24]). Rather than calling a variant of an old botnet as 'unknown', we find a binary classification approach more suitable. Further, since a multi-class or binary-class approach had little impact on the detection accuracy of P2P botnets, we decided to go in favor of a simpler and intuitive binary approach.

However, in a specific case where a network administrator needs to profile the exact P2P application running on a host, multi-class classification is the only solution. For the interested reader, we briefly share our experimental findings in this regard with respect to Gaussian mixture models (GMMs) [39]. In the flow-clustering phase elaborated previously, we explained the use X-means algorithm, which is a variant of K-means for the purpose of clustering flows. X-means reported four clusters in the data, which corresponded to the four benign P2P applications. As noted earlier, this was quite natural since more than 90% of the flows belong to P2P applications. Moreover, we also got large false positives amongst the benign P2P applications, indicating that their data points lie in overlapping clusters. GMMs are a natural choice in such cases since they are well known for clustering problems involving overlapping clusters. We repeated the flow-clustering experiments with the entire dataset (all eight applications) using GMMs with the optimization approach of expectation-maximization (EM) [40]. The flow-clustering phase with GMMs generated seven clusters for the eight applications, with each application except Storm having a cluster where it was dominantly present (clusters of Storm and Waledac overlapped). However, we observed that GMMs with EM is an *extremely slow* clustering approach. X-means outperforms GMMs by hundreds of times. X-means did not require number of clusters as an input, whereas GMMs with EM does. Thus, we did not find it suitable for `PeerShark`. Since this approach is not the mainstay of `PeerShark`, we do not spend more time on it. But an interested reader can leverage from the ability of GMMs and EM to separate overlapping clusters.

7 Conclusions

In this paper, we presented our system `PeerShark`, which uses a 'best of both worlds' approach by combining flow-based and conversation-based approaches to accurately segregate P2P botnets from benign P2P applications in network traffic. `PeerShark` clusters flows based on statistical features obtained from their network behavior and then creates conversations between the flows in the same cluster. Using several statistical features extracted from each conversation, we build supervised machine learning models to separate P2P botnets from benign P2P applications. With the models built on three classifiers, `PeerShark` could consistently detect P2P botnets with a true positive rate (or recall) ranging between 88% to 95% and achieved a low false positive rate of 0.2% to 0.3%. `PeerShark` could also detect unseen and unknown P2P botnet traffic with high accuracy.

As a part of work in progress, we are extending our approach with a distributed model for data collection where data collectors sit closer to the nodes inside the network (say at wi-fi access points). This will give greater visibility of the network traffic which occurs over LAN and never touches the backbone router of an enterprise. Such insight can be very valuable for detecting P2P bots inside a network perimeter which try to evade detection by maintaining connectivity with each other over LAN in a P2P fashion and limit the conversations with the outside world via one or two designated peers only (as seen in Stuxnet[c]).

Endnotes

[a]Skype has now moved to a cloud-based architecture [41].

[b]Personal communication with Babak Rahbarinia (November 2013) [24] w.r.t Storm, Waledac and Zeus.

[c]See [42] for details.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

This work was supported by grant number 12(13)/2012-ESD for scientific research under Cyber Security area from the Department of Information Technology, Govt. of India, New Delhi, India.

Author details

[1]BITS-Pilani, Hyderabad Campus, Hyderabad, Telangana 500078, India.
[2]University of Illinois at Chicago, Chicago, IL 60607, USA.

Received: 11 August 2014 Accepted: 12 September 2014
Published online: 08 October 2014

References

1. Ipoque Internet study 2008/2009. http://www.ipoque.com/en/resources/internet-studies. Accessed 4 Jan 2014
2. S Androutsellis-Theotokis, D Spinellis, A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. **36**(4), 335–371 (2004)
3. P Kopiczko, W Mazurczyk, K Szczypiorski, Stegtorrent: a steganographic method for the p2p file sharing service, in *Proceedings of the 2013 IEEE Security and Privacy Workshops (SPW '13)* (IEEE Computer Society Washington, DC, USA, 2013), pp. 151–157
4. R-A Shang, Y-C Chen, P-C Chen, Ethical decisions about sharing music files in the p2p environment. J. Bus. Ethics. **80**(2), 349–365 (2008)
5. T Isdal, M Piatek, A Krishnamurthy, T Anderson, Privacy-preserving p2p data sharing with oneswarm, in *Proceedings of the ACM SIGCOMM 2010 Conference*, vol. 40 (ACM New York, NY, USA, 2010), pp. 111–122
6. Sandvine Global Internet Phenomena Report 2013. https://www.sandvine.com/trends/global-internet-phenomena/. Accessed 4 Jan 2014
7. J Buford, H Yu, EK Lua, *P2P Networking and Applications*. (Morgan Kaufmann Publishers Inc., San Francisco, 2008)
8. C Kanich, N Weavery, D McCoy, T Halvorson, C Kreibichy, K Levchenko, V Paxson, GM Voelker, S Savage, Show me the money: characterizing spam-advertised revenue, in *Proceedings of the 20th USENIX Conference on Security, (SEC'11)* (USENIX Association Berkeley, CA, USA, 2011), pp. 15–15

9. Microsoft Security Intelligence Report, Volume 9, January-June 2010. http://www.microsoft.com/security/sir/. Accessed 1 Feb 2014
10. C Rossow, D Andriesse, T Werner, B Stone-Gross, D Plohmann, CJ Dietrich, H Bos, Sok: P2PWNED - modeling and evaluating the resilience of peer-to-peer botnets, in *Security and Privacy (SP), 2013 IEEE Symposium On* (IEEE Computer Society Washington, DC, USA, 2013), pp. 97–111
11. D Fisher, 88 percent of Citadel botnets down. http://threatpost.com/microsoft-88-percent-of-citadel-botnets-down/101503. Accessed 9 Jan 2014
12. D Drinkwater, Gameover trojan rises from the dead. http://www.scmagazineuk.com/gameover-trojan-rises-from-the-dead/article/357964/. Accessed 20 Jul 2014
13. T Greene, ZeuS botnet has a new use: stealing bank access codes via SMS. http://www.networkworld.com/news/2010/092910-zeus-botnet-sms-banks.html. Accessed 9 Jun 2013
14. J Stewart, Storm worm DDoS attack. http://www.secureworks.com/cyber-threat-intelligence/threats/storm-worm/. Accessed 1 Feb 2014
15. A Lelli, Waledac botnet back on rise. http://www.symantec.com/connect/blogs/return-dead-waledacstorm-botnet-back-rise. Accessed 1 Feb 2014
16. J Leyden, Fridge hacked. Car hacked. Next up, your light bulbs. http://www.theregister.co.uk/2014/07/07/wifi_enabled_led_light_bulb_is_hackable_shocker/. Accessed 12 Jul 2014
17. S Sen, O Spatscheck, D Wang, Accurate, scalable in-network identification of p2p traffic using application signatures, in *Proceedings of the 13th International Conference on World Wide Web (WWW '04)* (ACM New York, NY, USA, 2004), pp. 512–521
18. J Li, S Zhang, Y Lu, J Yan, Real-time p2p traffic identification, in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008* (IEEE, USA, 2008), pp. 1–5
19. M Iliofotou, H-C Kim, M Faloutsos, M Mitzenmacher P Pappu, G Varghese, Graph-based p2p traffic classification at the internet backbone, in *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops (INFOCOM'09)* (IEEE Press Piscataway, NJ, USA, 2009), pp. 37–42
20. G Gu, R Perdisci, J Zhang, W Lee, *Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection*. (USENIX Association, Berkeley, CA, USA, 2008), pp. 139–154
21. J François, S Wang, R State, T Engel, Bottrack: tracking botnets using netflow and pagerank, in *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I NETWORKING'11* (Springer Berlin, Heidelberg, 2011), pp. 1–14
22. H Hang, X Wei, M Faloutsos, T Eliassi-Rad, Entelecheia: detecting p2p botnets in their waiting stage, in *IFIP Networking Conference, 2013* (IEEE USA, 2013), pp. 1–9
23. R Schoof, R Koning, Detecting Peer-to-peer Botnets. University of Amsterdam (2007). University of Amsterdam. Technical report
24. B Rahbarinia, R Perdisci, A Lanzi, K Li, Peerrush: mining for unwanted p2p traffic, in *Detection of Intrusions and Malware, and Vulnerability Assessment* (Springer Berlin, Heidelberg, 2013), pp. 62–82
25. P Narang, JM Reddy, C Hota, Feature selection for detection of peer-to-peer botnet traffic, in *Proceedings of the 6th ACM India Computing Convention (Compute '13)* (ACM New York, NY, USA, 2013), pp. 16:1–16:9
26. J Zhang, R Perdisci, W Lee, U Sarfraz, X Luo, Detecting stealthy p2p botnets using statistical traffic fingerprints, in *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN '11)* (IEEE Computer Society Washington, DC, USA, 2011), pp. 121–132
27. J Zhang, R Perdisci, W Lee, X Luo, U Sarfraz, Building a scalable system for stealthy p2p-botnet detection. IEEE Trans. Inf. Forensics Security. **9**(1), 27–38 (2014)
28. T-F Yen, MK Reiter, Are your hosts trading or plotting? Telling p2p file-sharing and bots apart, in *Proceedings of the 2010 30th International Conference on Distributed Computing Systems (ICDCS '10)* (IEEE Computer Society Washington, DC, USA, 2010), pp. 241–252
29. T Karagiannis, K Papagiannaki, M Faloutsos, Blinc: multilevel traffic classification in the dark, in *SIGCOMM Comput. Commun. Rev., vol. 35* (ACM New York, NY, USA, 2005), pp. 229–240
30. T Karagiannis, A Broido, M Faloutsos, K Claffy, Transport layer identification of p2p traffic, in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)* (ACM New York, NY, USA, 2004), pp. 121–134
31. L Li, S Mathur, B Coskun, Gangs of the internet: towards automatic discovery of peer-to-peer communities, in *Communications and Network Security (CNS), 2013 IEEE Conference On* (IEEE USA, 2013), pp. 64–72
32. P Narang, S Ray, C Hota, VN Venkatakrishnan, Peershark: detecting peer-to-peer botnets by tracking conversations, in *Proceedings of the 2014 IEEE Security and Privacy Workshops (SPW'14)* (IEEE Computer Society Washington, DC, USA, 2014). in press
33. MM Masud, J Gao, L Khan, J Han, B Thuraisingham, Mining concept-drifting data stream to detect peer to peer botnet traffic (2008). Univ. of Texas at Dallas Technical Report# UTDCS-05- 08
34. D Pelleg, AW Moore, X-means: extending k-means with efficient estimation of the number of clusters, in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)* (Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2000), pp. 727–734
35. M Hall, E Frank, G Holmes, B Pfahringer, P Reutemann, IH Witten, The WEKA data mining software: an update. ACM SIGKDD Explor. Newslett. **11**(1), 10–18 (2009)
36. D Fisher, Storm, Nugache lead dangerous new botnet barrage. http://searchsecurity.techtarget.com/news/1286808/Storm-Nugache-lead-dangerous-new-botnet-barrage. Accessed 20 Jul 2014
37. S Stover, D Dittrich, J Hernandez, S Dietrich, Analysis of the storm and nugache trojans: p2p is here. USENIX; login. **32**(6), 18–27 (2007)
38. A Nappa, A Fattori, M Balduzzi, M Dell'Amico, L Cavallaro, Take a deep breath: a stealthy, resilient and cost-effective botnet using skype, in *Detection of Intrusions and Malware, and Vulnerability Assessment* (Springer Berlin, Heidelberg, 2010), pp. 81–100
39. D Reynolds Gaussian mixture models, in *Encyclopedia of Biometrics*, ed. by S Li, A Jain (Springer, 2009), pp. 659–663
40. TK Moon, The expectation-maximization algorithm. IEEE Signal Processing Mag. **13**(6), 47–60 (1996)
41. M Gillett, Skype's cloud-based architecture. http://blogs.skype.com/2012/07/26/what-does-skypes-architecture-do/. Accessed 3 Jul 2014
42. LO Murchu, Stuxnet P2P component. http://www.symantec.com/connect/blogs/stuxnet-p2p-component. Accessed 12 Feb 2014

doi:10.1186/s13635-014-0015-3
Cite this article as: Narang *et al.*: PeerShark: flow-clustering and conversation-generation for malicious peer-to-peer traffic identification. *EURASIP Journal on Information Security* 2014 **2014**:15.