Mechanisms for Intrusion Detection in Peer-to-Peer Networks

THESIS

Submitted in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

PRATIK NARANG
ID. No. 2011PHXF414H

Under the Supervision of

Prof. Chittaranjan Hota

Co-supervision of

Prof. V. N. Venkatakrishnan

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

2015

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the thesis entitled "**Mechanisms for Intrusion Detection in Peer-to-Peer Networks**" and submitted by **Pratik Narang** ID.No **2011PHXF414H** for the award of Ph.D. degree of the Institute embodies original work done by him under my supervision.

_____ _____

DR. CHITTARANJAN HOTA
Professor,
Department of Computer Science
and Information Systems,
BITS Pilani, Hyderabad Campus
Hyderabad, Telangana – 500078

DR. V. N. VENKATAKRISHNAN
Professor,
Department of Computer Science,
University of Illinois at Chicago

Date :_____ Date :_____

Dedicated to
my family

Acknowledgements

The journey of my doctoral degree involved the guidance, support, and well-wishes of several personalities. I take this opportunity to recognize their contributions.

I express my deep gratitude to my supervisor Prof. Chittaranjan Hota for his valuable guidance, encouragement and support throughout the course of my PhD. His directions helped me go deeper into my research work, and his flair for innovative thinking helped me grow as an independent researcher. I am extremely fortunate to have him as my supervisor.

My sincere thanks are due to my co-supervisor Prof. V. N. Venkatakrishnan (University of Illinois at Chicago) who took out his valuable time to discuss about my work over Skype, provide his inputs, and polish my work and ideas.

Thanks are also due to the members of my Doctoral Advisory Committee (DAC), Prof. N. L. Bhanumurthy and Prof. R. Gururaj, for their valuable inputs and their encouragement throughout the course of my work. I also thank Prof. Nasir Memon and Prof H. Taha Sencar at the New York University (Abu Dhabi campus) for allowing me to spend a summer with their research group.

I am also indebted to Mr. Abhishek Thakur, Mr. D. Jagan Mohan Reddy and other members of the NetClique research group for spending their time in discussing about my research work and giving their

inputs. I had a great opportunity to work with some brilliant undergraduate students who contributed towards the implementation aspects of my research work: G. Sharath Chandra, Subhajit Ray, Kunal Mehta, Vansh Khurana, Prasenjeet Biswal, and Robin Kumar Nayak.

I express my gratitude towards Prof. V. S. Rao (Director and Senior Professor at BITS Pilani Hyderabad campus and Acting Vice-chancellor of BITS Pilani) and Prof. M. B. Srinivas (Dean, Administration). I gratefully acknowledge the support of all administrators at BITS Pilani for providing a suitable working atmosphere. I thank all my fellow research scholars and other friends for their well-wishes and support.

Even though others have contributed to my thesis work, I am responsible for the content, conclusions, errors, or omissions in my thesis.

Abstract

Peer-to-peer overlay networks brought together end-users from different parts of the world and enabled them to share and mobilize resources. Peer-to-peer networks have seen widespread deployment in applications related to file-sharing, sharing of computing resources, music streaming, etc. Due to their decentralized and distributed architecture, peer-to-peer overlays involve many challenges of security. The distributed and decentralized peer-to-peer infrastructure has offered a lucrative alternative to bot-masters to build botnets which are not prone to any single point-of-failure. Recent botnets utilize the peer-to-peer architecture for their command-and-control. Such botnets have demonstrated high resilience towards break-down and take-down attempts.

A significant portion of this thesis focuses on the problem of detection of peer-to-peer botnets in the presence of traffic from benign peer-to-peer applications. Our approaches leverage on the behavioral differences between peer-to-peer botnets and benign peer-to-peer applications. The first approach combines the benefits of flow-based and conversation-based mechanisms with a two-tier architecture, and addresses the limitations of the respective mechanisms. By extracting statistical features from the network traces of peer-to-peer applications and botnets, we built supervised machine learning models which could accurately differentiate between benign peer-to-peer applications and peer-to-peer botnets, and could also detect *unknown* peer-to-peer

botnet traffic with high accuracy. The second approach further enhances conversation-based mechanisms by leveraging on the timing and data patterns in peer-to-peer botnets. The resultant approach utilizes Fourier transform and information entropy to extract features from the traces of peer-to-peer botnets and benign peer-to-peer applications. We built detection models with multiple supervised machine learning algorithms, and demonstrated that our detection approach is resilient towards evasive P2P botnets which may try to evade detection by deliberate injection of arbitrary noise in their communication patterns. With our approach, we could detect peer-to-peer botnet traffic in the presence of injected noise with True Positive rate as high as 90%. The results from peer-to-peer botnet identification modules are further used by a 'firewall' module to generate a dynamic rule-set. An Iptables-based firewall was setup on our testbed's Gateway machine. For the prototype implementation, rules were implemented to rate-limit the traffic from benign peer-to-peer applications, drop traffic from peer-to-peer bots, etc.

The third approach studies the problem of strategic positioning of intrusion detection systems in a peer-to-peer environment involving a number of peers and super-peers (who have higher responsibilities in the network). A malicious entity may become part of the peer-to-peer network by joining from any part of the network. It can attack a super-peer and thus disrupt the functioning of the peer-to-peer network. We use game theory to model the interactions between the adversary and the peers. Peers may try to secure the network by running intrusion detection systems at certain strategically-chosen locations in the network. But, a deterministic schedule of deploying the intrusion detection systems can be observed and thwarted by an adversary. In our work,

we explore the problem of strategically positioning intrusion detection systems in a peer-to-peer network with a randomized, game-theoretic approach. This approach distributes the responsibility of running the intrusion detection systems between the peers in a randomized fashion and minimizes the probability of a successful attack.

Lastly, we propose a scalable and distributed, Hadoop-based framework for the detection of peer-to-peer botnets. Our work uses the Hadoop-ecosystem to adopt a 'host-aggregation based' approach which aggregates behavioral metrics for each peer-to-peer host seen in network communications, and uses them to distinguish between benign peer-to-peer hosts and hosts infected by peer-to-peer botnets. Further, we propose a distributed data-collection architecture where data collectors sit closer to the nodes in the network. It can monitor *inside-to-inside* LAN traffic, as opposed to relying solely on the NetFlow information available at a backbone router. Our approach is scalable and distributed by design, and is especially beneficial for the detection of smart peer-to-peer bots *inside* the perimeter of a network – which talk to each other and send upgrades to themselves on LAN in a 'peer-to-peer' fashion, and limit communication to the outside world via one or two peers only.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations/Symbols

Term	Definition
P2P	Peer-to-Peer
ISP	Internet Service Provider
C&C	Command and Control
DoS	Denial of Service
DDoS	Distributed Denial of Service
TCP	Transmission Control protocol
UDP	User Datagram Protocol
IP	Internet Protocol
LAN	Local Area Network
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
SETI	Search for Extraterrestrial Intelligence
DHT	Distributed Hash Tables
DPI	Deep Packet Inspection
DFT	Discrete Fourier Transform
IRC	Internet Relay Chat

Chapter 1

Introduction

1.1 P2P networks

Computer networking has undergone a paradigm change over the past decade, with the number of users, applications, and computing devices going through an explosive growth. The past decade saw 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% [Ipoque 2008]. As an increasing number of users got access to powerful processors, large storage spaces, and increasing bandwidth, P2P networks presented a great opportunity to share and mobilize resources. The runaway 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, eMule, Gnutella, etc.), or sharing of computing resources (SETI @ home project). Apart from these, the P2P architecture has also been widely used for music streaming (Spotify), IPTV (LiveStation) and Voice-over-IP based services (Skype[1]). The 2014

[1]Skype has now moved to a cloud-based architecture [Gillet 2013].

global Internet phenomena report by Sandvine [Sandvine 2014] points to the overall percentage of P2P traffic to be 27% in the Asia-Pacific, with BitTorrent being the dominant application of P2P.

[Androutsellis-Theotokis & Spinellis 2004] provide a comprehensive definition for P2P computing systems: *"Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority"*. The client-server based architectures are characterized by an asymmetric relationship between client and server, wherein the client queries and the server responds. In contrast to that are the distributed P2P systems where every node acts as both a server and a client. The construction of P2P networks is in the form of an 'overlay' on top of the IP layer, typically with a decentralized protocol allowing 'peers' to share resources. A P2P overlay network is a logical network at the application layer providing connectivity, routing and messaging amongst addressable end-points of the communication. The 'peers' form a set of interconnections to share and mobilize resources such that peers have symmetric roles in the overlay for both message routing and resource sharing [Buford *et al.* 2008]. Unlike the client-server model, a P2P network connects several peers directly. The architecture of a P2P network is determined by the characteristics of its overlay network, placement and scope of data, and the protocols used for communication. The choice of the architecture influences how the network can be used for various tasks like searching and downloading.

In a P2P network, messaging and search APIs are used by the peer to communicate with other peers and search for contents from others. Routing and forwarding allows each peer to maintain connection state to neighboring peers, which in turn

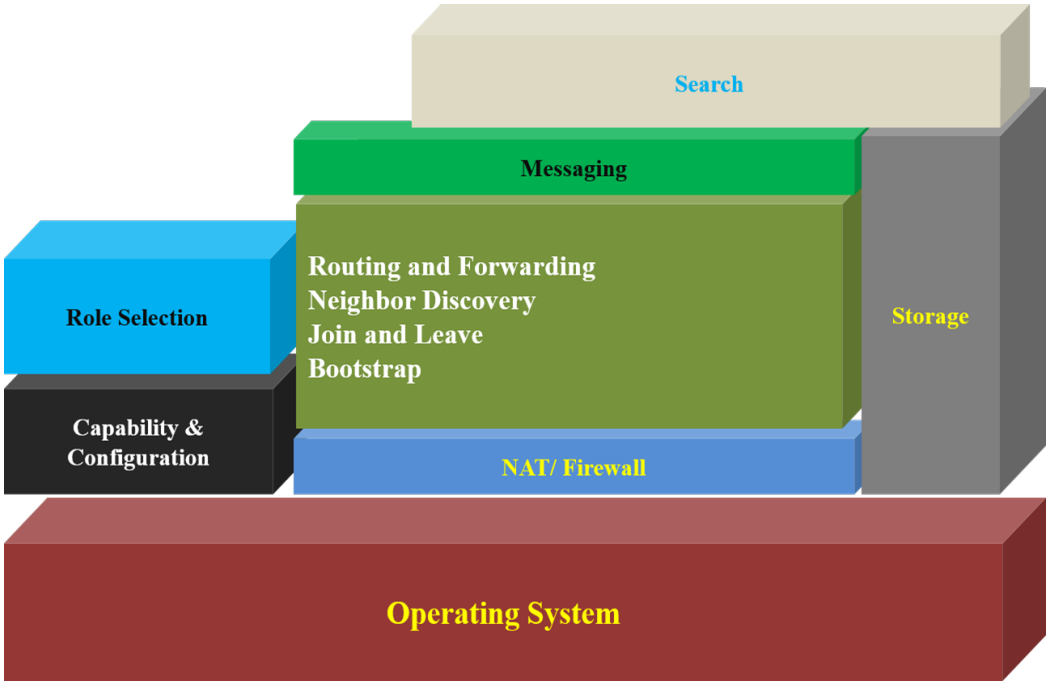


Figure 1.1: Generic peer architecture

could include neighbor discovery and state maintenance within the overlay. The generic peer architecture is given in Figure 1.1 (adopted from [Buford *et al.* 2008]). Using bootstrapping peers, peers could join and leave the overlay network. The content storage functionality at a peer should facilitate access to the stored object locally as well as by other peers using improved search indices and a query interface. Peers should also self-configure and assess their capabilities based on resource availabilities and stability.

1.1.1 Categorization of P2P networks

Many different designs of P2P networks have led to researchers proposing various kinds of categorization. Based on the topology, P2P networks are generally classified as being 'Unstructured' and 'Structured' [Lua *et al.* 2005].

Peers in an unstructured P2P network are organized in a random graph. The links between nodes are established arbitrarily and hence there is no correlation

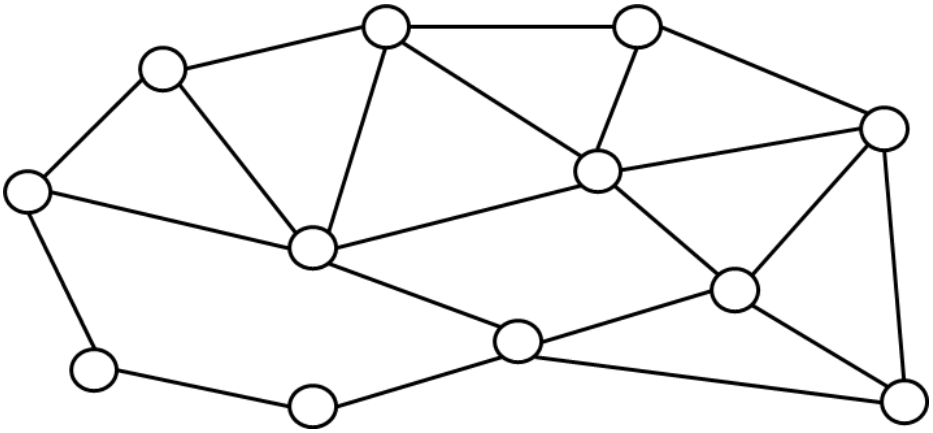


Figure 1.2: Unstructured P2P network

between a peer and the content being managed by it. A random graph is formed for a set of n isolated vertices by adding successive edges between pair of vertices uniformly at random [Buford *et al.* 2008]. Random graphs are known to be robust against partition [Vishnumurthy & Francis 2004] and provide connectivity and expansion even for very small degrees [Wormald 1999]. Random networks have demonstrated resilience against failures [Kim & Médard 2004]. An example of unstructured P2P topology is given in Figure 1.2. Unstructured P2P networks use flooding, random walks or expanding Time-to-Live (TTL) search on the graph to query content stored by the participating peers [Lua *et al.* 2005]. If a peer wants to find some piece of information in the network, the query has to be flooded through the network in order to find as many peers as possible sharing that information. In such a system, the network is easy to construct. Unstructured P2P networks such as Freenet [Clarke *et al.* 2001], Gnutella, Direct Connect, etc. offer decentralization and simplicity, but may require $O(N)$ hops (worst case complexity) to search a file when the network is made of N nodes.

In contrast to the unstructured P2P networks, structured P2P overlay networks tightly control both the network topology and the placement of content. The content in such systems is not placed at random peers but rather at a specified location. Every data item in the overlay network is assigned a key, and peers are

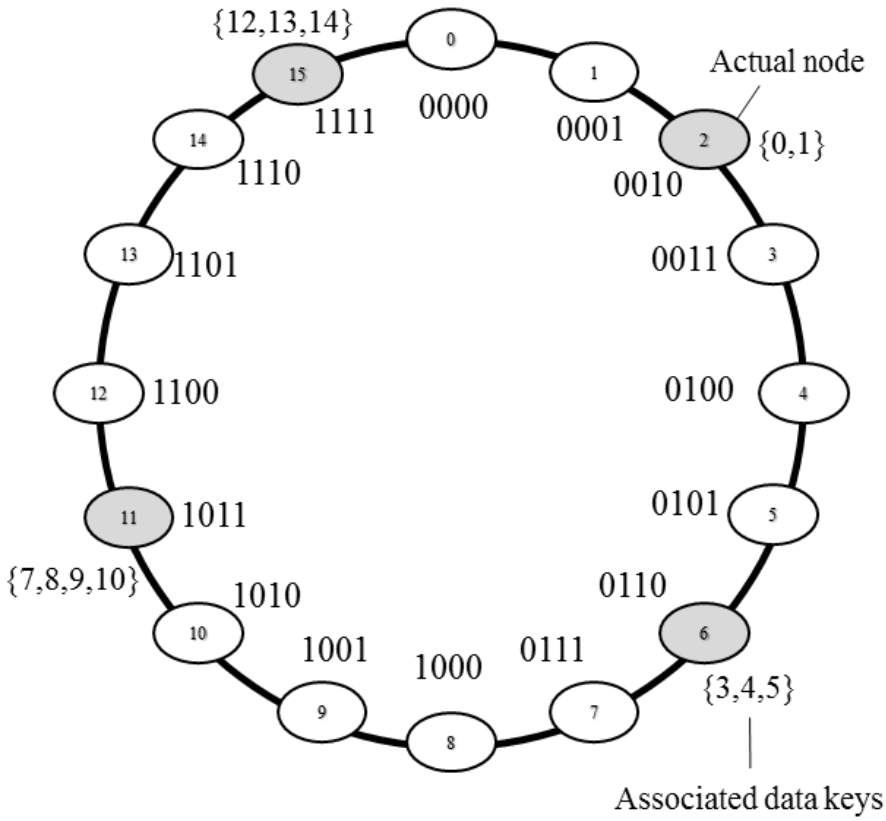


Figure 1.3: Structured P2P network

organized into a graph that maps each data-key to a peer. This enables efficient discovery of data items using the key of a data element [Sit & Morris 2002]. Specifically, the content is stored at specified locations based on distributed hash tables (DHTs), which are decentralized and distributed systems providing a lookup service similar to a hash table. An example is given in Figure 1.3. With the DHT data structure and algorithm, peers can easily map data-keys to nodes. This facilitates faster lookup for any data object in a small number of overlay hops. These networks provide a stable and robust mechanism for storing and retrieving content. Content Addressable Network (CAN) [Ratnasamy *et al.* 2001], Chord [Stoica *et al.* 2001], and Pastry [Rowstron & Druschel 2001] are some examples of structured P2P overlays.

P2P overlays are also categorized based on the degree of centralization. In the true sense of the term, P2P overlays are expected to be fully decentralized. This is

however not true in practice.

Purely decentralized P2P networks involve all peers with more or less the same role. All participating peers enjoy the same share of resources in the network and have similar responsibilities. Such networks exhibit robustness to peer-churn (joining and leaving of peers) and node failures. However, peers have limited knowledge about the P2P network. Therefore, purely decentralized overlays usually do not provide guarantee on search efficiency and content availability. Early versions of Gnutella (0.4), FreeHaven, Freenet, Chord, Pastry and Tapestry are examples of this architecture.

Partially decentralized P2P architectures introduce the notion of 'super-peers' or 'ultra-peers'. The majority of overlay responsibilities (related to routing, indexing, etc.) are assigned to a small subset of these more powerful nodes. A super-peer may be chosen based on the participation/contribution of the peer in the network, its uptime, bandwidth, publicly visible IP address, etc. Super-peers are connected among themselves in a pure decentralized architecture. Introduction of super-peers does not bring the problem of single point of failure since they are chosen dynamically. If a super-peer fails or leaves the network, local peers automatically connect to another super peer and the network will chose a replacement for the failed super-peer. Gnutella2, Kazaa and eMule are some examples of this architecture. Skype also had a super-peer based architecture.

Hybrid decentralized architecture uses a centralized facility to allow interactions between peers. The central server, in general, maintains the metadata, file indices, etc. If the central server fails, peers can no longer reach out to each other. Napster and Direct Connect are examples of this architecture.

1.1.2 Issues of security and privacy

P2P networks and applications have drawn a significant attention from the community of researchers. P2P file-sharing is notorious for being a source of information leakage, piracy, spread of malware etc., and it has known to become a serious concern for Internet Service Providers (ISPs), Governments and other public and private organizations.

To detect and prevent information leakage from an internal node or user's machine, network administrators must deploy appliances that can handle these types of applications in an intelligent manner while allowing the users to not worry about the security and privacy concerns that could arise out of such usage. P2P file sharing applications such as BitTorrent allow people to share files amongst themselves. Sharing files on one's computer with unknown users ('peers') on a public Internet by creating logical identities brings in natural concerns related to security and privacy. In order to access and share files on one's computer within a P2P network, one must open a specific TCP/UDP port through the firewall and allow the P2P application to communicate. In effect, once firewall has opened the port, one can no longer be protected from malicious traffic coming through it. Another major security concern while using P2P applications is to determine the authenticity of the content that is downloaded or accessed. In other words, how do you ensure that you have not received malicious content in the form of spyware or a bot? The self-configurable nature of a peer and the dependence of peers with each other allows malicious peers to abuse the trust. Since internals are exposed to fellow peers in the name of sharing or distributing the workload, adversaries can leverage this to compromise the P2P network and create havoc for other users.

Furthermore, P2P traffic has many characteristics that overlap with malicious traffic. For example: multiple persistent high-throughput flows (similar to spyware),

Table 1.1: Well-known port numbers used by several P2P applications

Application	TCP Port	UDP Port
Direct Connect	411, 412, 1025–32000	1025–32000
eDonkey	3306, 4242, 4500, 4501, 4661–4674, 4677, 4678, 4711, 4712, 7778	4665, 4672
Gnutella	6346, 6347	6346, 6347
Kazaa	1214	1214
Limewire	6346	6347
BearShare	6346	6346
eMule	4662	4672
BitTorrent	6881–6889	6881–6889
winMx	6699	6257

communication with centralized P2P trackers (also seen in botnets), large number of simultaneous peer connection requests, many of which are unsuccessful due to peer-churn (similar to self-propagating malware infections and port-scan attacks), communication on uncharacteristic ports, receiving requests from a peer and forwarding those requests to neighbors immediately, trying to connect using both TCP and UDP ports, etc. Consequently, in addition to consuming considerable network resources, P2P traffic also creates several issues for network security devices such as Intrusion Detection Systems (IDS), firewalls, etc.

With the proliferation of P2P systems, it is critical to consider the impact of these systems on the security of an Internet environment that is already struggling from several security issues. The usage of P2P applications has not been regulated much by policy-makers in most of the countries. Present day IDS/IPS solutions generally block P2P traffic at network as a part of their policy enforcement mechanism. This is not an optimal solution as end-user flexibility is lost and rule enforcements are only at the organizations boundary. Commercial security appliances like Cyberoam, FortiGate (from Fortinet), etc. detect and alert policy violations for use of P2P applications using techniques like port-based analysis and protocol-analysis. However, majority of P2P applications today randomize

their port numbers and also allow their users to manually change the default port numbers (see Table 1.1), making it difficult for perimeter security appliances to effectively use port-based analysis. Protocol-analysis may also fail if encryption or tunneling is used to hide the traffic. It is also frequently seen that user-owned devices (like smart-phones and tablets) connect to torrent-based websites on the Internet directly over cellular networks (3G, 4G, etc.). In this way, they bypass organizational enforcements and the IDS/firewall solutions placed on the periphery of the network. If such devices get infected by a malware while being directly connected to the Internet, they pose a security threat to their organization when they connect to the organizational network again. Hence, there is a great need of efficient mechanisms to detect and classify malicious traffic in a P2P environment.

1.2 Background on P2P botnets

As P2P networks are inherently modeled without any centralized server, they lack a single point of failure [Buford *et al.* 2008]. 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 [Kanich *et al.* 2011]. Botnets are being touted as the largest threat to modern networks [Microsoft 2010].

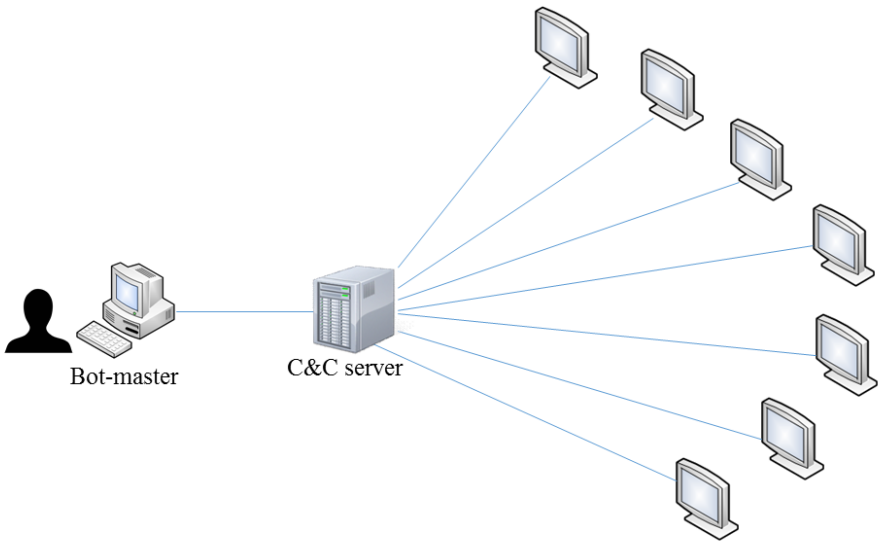


Figure 1.4: A centralized botnet

The command-and-control (C&C) communication channel is the key aspect of any botnet. Traditional botnets had a centralized architecture (e.g., Gaobot, Spybot, R-bot, etc.), wherein the bot-master commanded the bots using one or few C&C servers (see Figure 1.4). Internet Relay Chat (IRC) had been the most commonly used communication channel. A centralized architecture suffered from an obvious drawback that if the C&C server is identified and taken down, the bot-master loses control over his/her bots. 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. Recent botnets utilize the P2P architecture for their C&C communications (see Figure 1.5). The bots create an 'overlay network' amongst themselves and use P2P channels to exchange commands, pass-on stolen information, etc. Although setting up P2P communication channels can be more costly and will suffer from higher latency (compared to a centralized communication channel), it offers a great advantage to the bot-master in terms of high resilience towards network break-down and take-down attempts [Rossow *et al.* 2013, Andriesse *et al.* 2013]. Even if a few bots in the network are identified or taken down, the botnet does not break-down.

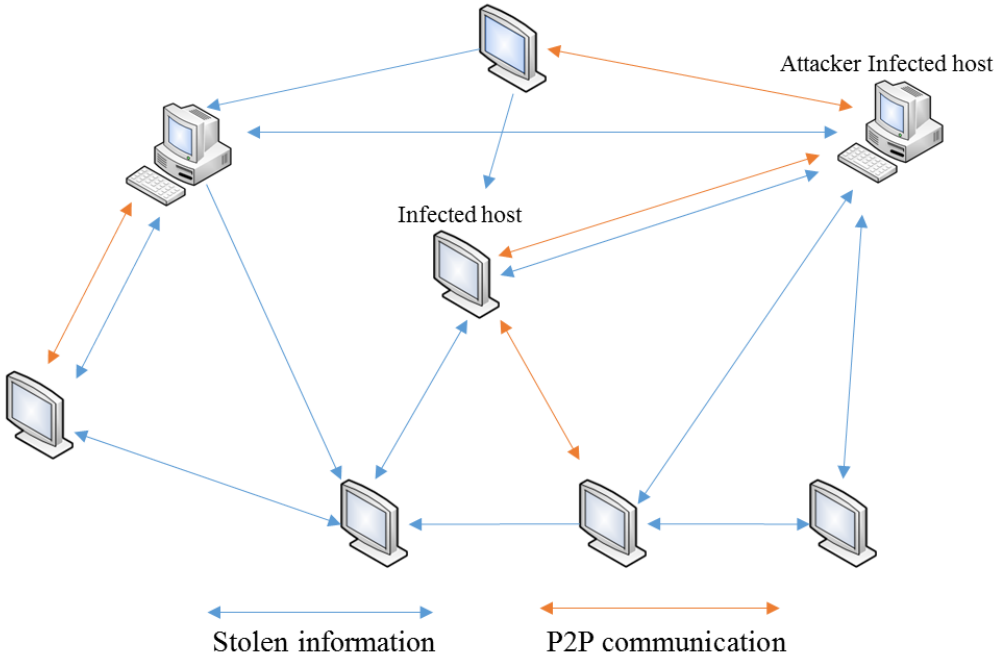


Figure 1.5: A P2P botnet

A number of P2P-based botnets have been seen over the past decade, and a few of them have been taken down recently with the combined effort of multiple nations. Some notable example of P2P botnets include the Zeus botnet, the Storm botnet, its improved version as the Waledac botnet, etc.

The P2P variant of Zeus botnet, also known as 'Gameover Zeus' [Drinkwater 2014], has been a popular toolkit amongst bot-creators which has been tweaked and improvised to create newer botnets. The recent banking trojan 'Dridex' also belongs to the Gameover Zeus family [Mimoso 2015]. The massive Citadel botnet [Segura 2012] is also known to be a variant of Zeus. Citadel is believed to have stolen more than 500 million USD from bank accounts over 18 months. It was reported in 2013 that 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 [Fisher 2013]. However, reports in 2014 claim that the botnet is on the rise again, with a tweaked version being used to target a small number of European banks [Drinkwater 2014]. A variant of the Zeus P2P botnet also targeted Nokia phones

using Symbian OS [Greene 2010]. 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 [Stewart 2007]. Researchers have confirmed that the Waledac botnet is an improved version of the Storm botnet [Lelli 2011]. 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.

Perhaps the most notable example in this regard is of the highly acclaimed and sophisticated Stuxnet botnet. Although Stuxnet targeted SCADA systems, and detection of such botnets cannot be discussed in the same vein as that of Internet botnets, it is worth mentioning that Stuxnet also used the P2P architecture for communication between its peers over LAN [Murchu 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 IDS and firewalls, botnets tend to keep their communication patterns (with the bot-master or other bots) quite stealthy. IDS 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 IDS/firewalls.

With the advent of the Internet of Things (IoT), the possibility of malware taking control of 'smart' appliances such as television, air-conditioners, refrigerators, etc. will not be limited to imagination. In fact, there have been recent reports of 'smart' refrigerators and cars being hacked, and Wi-Fi enabled LED bulbs having security weaknesses [Leyden 2014]. Since P2P computing systems are expected to be an integral part of IoT [Bedrosian 2013], we can expect that the evolution of P2P botnets will continue and the detection of such botnets will continue to be an important research paradigm.

1.3 Firewalls, IDS and IPS

Many different types of devices and mechanisms are used to provide security at the network perimeter of an enterprise. Firewalls and Intrusion Detection/Prevention Systems are two of the most significant as well as foundational tools in this regard. In this section, we will present a brief background about these techniques.

A firewall is a network security system that enforces an access control policy on a network. It can be software-based or hardware-based, and it controls incoming and outgoing network traffic based on a set of rules. Firewalls can be thought of as a pair of mechanisms: one which exists to block traffic, and the other which exists to permit traffic. Policies are typically based on protocol type, source address, destination address, source port, and/or destination port. Packets that do not

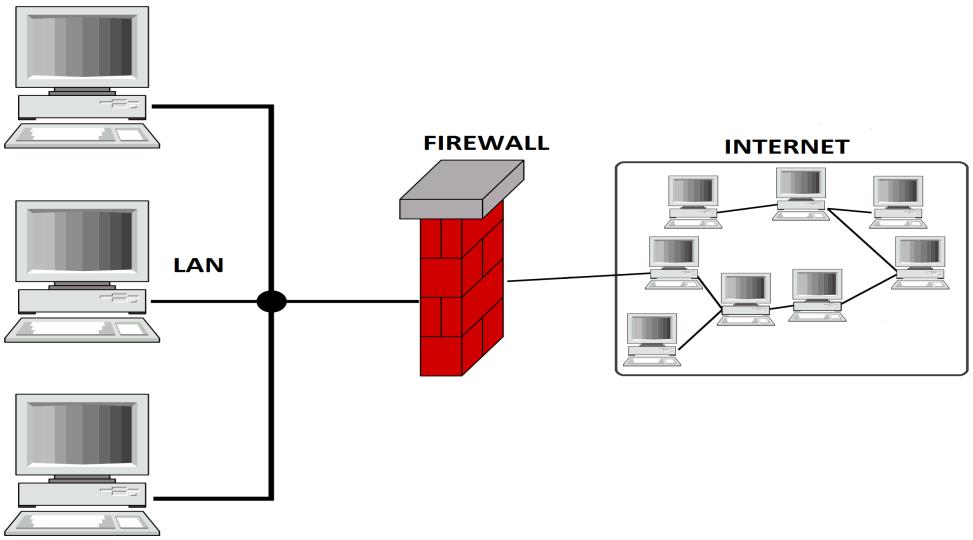


Figure 1.6: A generic firewall

match the policy are rejected. A firewall establishes a barrier between a trusted and secure 'home' network and another network (e.g., the Internet) that is assumed not to be secure and trusted. A generic example of a firewall is given in Figure 1.6, wherein the trusted 'home' network is on the left, and the untrusted network (Internet) is on the right.

Intrusion detection is the process of monitoring the events occurring in a computer system or network, and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices [Scarfone & Mell 2007]. An IDS is an application or a device which monitors the network and/or system activities for such policy violations, threats, malicious activities etc., and generates reports for the administrator.

Both IDS and firewalls relate to securing a network. However, firewalls primarily work on the aspect of looking for intrusions from outside the network and stop them from happening. Firewalls assume the 'home' network to be secure and trusted. Firewalls limit access between the 'home' network and the untrusted network to prevent intrusions, and do not signal an attack from inside the net-

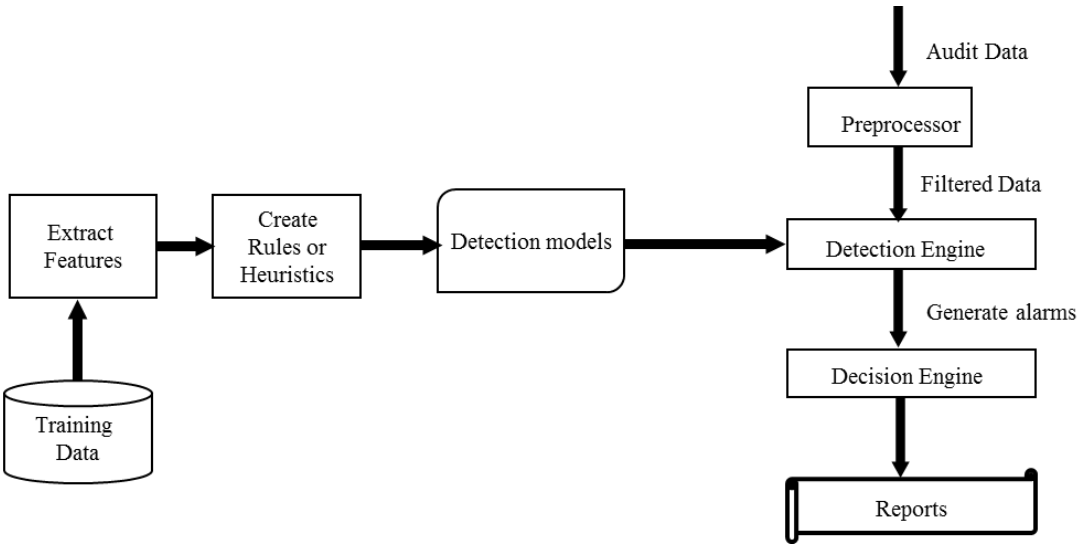


Figure 1.7: An IDS with its different components

work. An IDS evaluates traffic to detect policy violations or potentially malicious activities, and raises an alarm if such activities are detected. An IDS can also monitor for attacks generating from within the 'home' network. It may achieve this by using signature-based detection mechanisms, examining communication patterns, use of other heuristics, etc. As its name suggests, an Intrusion Detection System is usually limited to the task of 'detecting' an intrusion. After an intrusion is detected, an alarm may be raised for the network administrators. IDS usually does not react or take live action against an intrusion. This may be because the IDS module is not located 'inline' with the network and thus cannot take any live action, or because the responsibility of taking live action lies with a separate firewall module.

An Intrusion Prevention System, on the other hand, is a software that contains all capabilities of an IDS, and is also coupled with the capability of blocking the intrusions. Typically, if an IDS module sits 'inline' with the network it is monitoring, and is coupled with a basic 'firewalling' mechanism, it can function as an IPS. An IDS with 'prevention' functionality is usually referred by the single term 'Intrusion Detection/Prevention System' or IDPS. A generic diagram for an IDS,

demonstrating its various components, is given in Figure 1.7.

In general, IDS can be Host-based or Network-based. A Host Intrusion Detection System (HIDS) runs on individual hosts inside the network. It can monitor all inbound/outbound packets from that particular host, and it will alert the user or administrator if some suspicious activity is detected. One generic way in which a HIDS functions is by taking a snapshot of existing system files and matching it to the previous snapshot. If certain critical system files have been modified or deleted, an alert is generated.

A Network Intrusion Detection System (NIDS) is placed at a strategic point(s) within the network to monitor traffic to and from all devices on the network. It analyses the incoming and outgoing traffic to identify policy violations, known threats, etc. In order to identify malicious activities, it may utilize signature-based matching of the content of traffic payload, or behavior-based profiling of the traffic. When an attack or abnormal activity is identified, alert(s) can be issued to the administrator.

1.4 Background study

A significant portion of this thesis utilizes known machine learning approaches in the realm of supervised and unsupervised learning techniques, feature selection techniques, etc. Although a complete tutorial on these topics is out-of-scope of this thesis, we provide a brief introduction to some of these terminologies which find a frequent mention in our work.

A portion of thesis also relies on game theoretic techniques and strategies. We also present a very brief background of game theory specific terminologies related to our work. This section serves to be a quick reference for the reader, and is not intended to be exhaustive.

1.4.1 Machine learning

Machine learning is the science of getting computers to act without being explicitly programmed. A more formal definition is provided by Tom Mitchell in [Mitchell 1997]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

1.4.1.1 Supervised learning

Supervised learning is the machine learning task of inferring a function from labeled training data [Mohri *et al.* 2012]. The 'training data' is defined as a set of labeled examples, with each example consisting of an input vector and a desired output value. This output value is the 'class label' field for that particular input, which is the target value of that particular instance of the training set. Supervised learning algorithms take the labeled training data as the input. The training data is then analyzed to produce an inference function. This inference function can now be used to map new examples, and is expected to be able to predict their output class label. Accurate prediction necessitates that the algorithm is able to generalize the prediction from the training data to unseen examples in a "reasonable way".

C4.5 Decision trees, Random forests, Naïve Bayes classifier, etc. are some popular supervised machine learning algorithms that have been used in this work.

1.4.1.2 Unsupervised learning

Unsupervised machine learning contrasts itself from supervised machine learning in the end-goal. Unsupervised learning involves finding the hidden structure in data which is unlabeled. Here, the goal is not to maximize a utility function, but simply to find similarities in the training data. Unsupervised learning algorithms

often rely on some kind of *clustering* of input data. The motivating principle is that the clusters discovered are often equivalent to an *intuitive* classification on the training data. The new instances can be mapped into one of the clusters thus generated.

Unsupervised learning techniques which utilize clustering algorithms include k-means clustering, hierarchical clustering, mixture models, etc. In neural networks, self-organizing maps (SOM) are considered as an unsupervised learning technique.

The reader is requested to note that there are other categories of 'learning' apart from supervised and unsupervised techniques. However, other techniques are not relevant to this thesis, and we omit any discussion on them for the sake of brevity.

1.4.1.3 Feature extraction

Feature extraction is a very broad term which includes many routines such as feature construction, space dimensionality reduction, sparse representations, and feature selection. These techniques are commonly used as preprocessing steps to machine learning and statistics tasks of prediction, including pattern recognition and regression [Guyon & Elisseeff 2006].

The first step of feature extraction may be defined as conversion of raw data into a set of useful features. This often necessitates human expertise and domain knowledge. The process of converting raw data into useful features is specific to a domain. These methods will be discussed in detail during the course of this thesis. This thesis also involves the use of feature selection techniques, and we discuss about them in the next section. We ignore discussion on other topics related to feature extraction since they do not hold much relevance to this thesis.

1.4.1.4 Feature selection techniques

Feature selection is the process of reducing the number of features, with the aim of removing those features from the learning algorithm which have low impact on the "classification problem". Primary motivation behind feature selection is that the training data contains many features which are either *redundant* to the classification problem (i.e., they provide no further information than the currently selected features), or are totally *irrelevant* to the classification problem itself. Consider the simple example of features that may be extracted from network flows for the purpose of classification of network traffic. Many features that can be extracted for this purpose – such as average packet size, duration of the flow, number of unique ports used, etc. But certain features are not relevant to the task of classifying network traffic. For example, every TCP data packet receives an acknowledgment in response. A count for the number of ACK packets is not a good feature for classifying network traffic because it will be present in packets of every application, and cannot help in distinguishing between different applications.

Reducing the number of features provides direct benefit in terms of lesser training time for the learning model. Such 'feature selection' is also known to reduce the problem of 'over-fitting' or the variance error [Hall 1999], and is also helpful in overcoming the class imbalance problem [Van Der Putten & Van Someren 2004]. It should be noted that although the final accuracy of the learning algorithm will depend on the learning technique used, the suitability of the original features (i.e., those obtained prior to use of feature selection) etc., feature selection techniques are effective in optimizing the performance of the classifier since the number of features used for classification are reduced.

Given below is an overview of two well-known feature selection techniques which have been used for our work in this thesis.

Correlation-based feature selection: Correlation-based Feature Selection (CFS) algorithm is based on a simple filter method. Given a full set, it finds an optimal subset that contains features that are highly correlated with class label and uncorrelated with each other. CFS evaluates correlation of the feature subset on the basis of this hypothesis – "A good feature subset contains features highly correlated with (predictive of) the classification, yet uncorrelated with (not predictive of) each other" [Hall 1999]. This hypothesis relies on two metrics – one is 'feature-class' correlation, and other is 'inter-correlation' amongst features. The 'feature-class' correlation indicates how much the feature is correlated with its class, while inter-correlation amongst features tells of correlation between any two features.

$$M_s = \frac{k\bar{r}_{cf}}{\sqrt{k + k(k-1)\bar{r}_{ii}}} \tag{1.1}$$

The above equation calculates the merit values ('M_s') for each subset of k features. For each value of k, all possible subsets are chosen and merit values are computed. The subset giving the highest merit is the output of CFS. In Equation 1.1, $\bar{r}_{cf}$ is the average correlation between feature and class, and $\bar{r}_{ii}$ is the average inter-correlation between two features. The heuristic metrics $\bar{r}_{cf}$ and $\bar{r}_{ii}$ are calculated as the Symmetrical Uncertainty (SU) [Hall 1999] given in Equation 1.2.

$$SU = 2.0 \times \left[\frac{H(X) + H(Y) - H(X,Y)}{H(X) + H(Y)}\right] \tag{1.2}$$

where H(X) is defined as entropy, given by:

$$H(X) = -\sum_{x \epsilon X} p(x) log_2(p(x)) \tag{1.3}$$

Here $p(x)$ is the probability of occurrence of x.

Consistency-based subset evaluation: The consistency-based Subset Evaluation (CSE) search algorithm [Dash & Liu 2003] evaluates the feature subsets and finds an optimal subset of relevant features that are consistent to each other. To determine the consistency of a subset, the combination of feature values representing a class are given a pattern label. All instances of a given pattern should thus represent the same class. A pattern is *inconsistent* if there exist at least two instances such that their patterns are same but they differ in their class labels. The overall inconsistency of a pattern p is calculated by Inconsistency Count (IC):

$$IC(p) = \quad n_p - c_p \tag{1.4}$$

where n_p is the number of instances of the pattern p, and c_p is the number of instances of the majority class of the n_p instances.

The overall consistency of a subset S is calculated using Inconsistency Ratio (IR). IR is the sum of all inconsistency counts over all the patterns of the feature subsets divided by the total number of instances in the data set:

$$IR(S) = \frac{\sum\limits_{p} IC(p)}{\sum\limits_{p} n_p} \tag{1.5}$$

1.4.1.5 Performance metrics for classification

Many different metrics have been proposed to measure the performance of a classification algorithm. In this thesis, we use established metrics such as Precision, Recall and False Positive rate for evaluation of the approaches proposed by us. We briefly define these metrics 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 $\frac{TP}{TP+FP}$.

- **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 $\frac{TP}{TP+FN}$.
- **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 $\frac{FP}{FP+TN}$.

TP stands for True Positive, TN stands for True Negative, FP stands for False Positive and FN stands for False Negative.

1.4.2 Game theory

Game theory is the study of mathematical models of conflict and cooperation between intelligent and rational decision-makers [Myerson 2013]. It provides mathematical rules and concepts for analyzing situations of conflict, competition or cooperation among two or more individuals, where their decisions influence each other's welfare. Such situations are analyzed in form of strategic 'games'. These are games of strategy (such as chess) but not of chance (such as rolling a dice). The strategies chosen by all 'players' of the game jointly determine the final outcome of the game.

1.4.2.1 Nash equilibrium

John Nash formally defined an equilibrium of a non-cooperative game to be a profile of strategies, one for each player in the game, such that each player's strategy maximizes his/her expected utility payoff against the given strategies of the other players [Myerson 1999]. It implies that if each player of the game has chosen his/her equilibrium strategy, no player in the game can benefit just by changing his/her strategy while the other players keep their strategies unchanged. Each

player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only his/her own strategy [Osborne & Rubinstein 1994].

1.4.2.2 Mixed strategy Nash equilibrium

A mixed strategy of a player is defined as a randomization over its given pure strategies. This implies that the players assign a probability distribution over their pure strategies and the payoff to each player becomes the expected payoff derived from that probability distribution [Nash *et al.* 1950]. In such a case, the mixed strategies of players are said to be in a Nash Equilibrium when each player's mixed strategy is a 'best response' to every other player in the game.

1.4.2.3 Two person zero-sum games

A zero-sum game is a situation where the loss incurred by one player by playing any set of strategies is exactly equal to the gain received by the other player by playing the same set. Therefore, the sum of payoffs for any choice of strategies of the players is zero. The matrix form in such games is usually defined by writing the payoff of only the 'row' player; the payoff to 'column' player is the negative of that for each strategy.

1.4.2.4 Von Neumann's Minimax principle

It is defined for a two-person zero-sum game. The payoffs are treated as the value (utility) paid by the 'column' player to the 'row' player. The principle states that the greatest expected payoff that a row player can derive by playing a mixed strategy is equal to the smallest expected payoff that the column player will derive using a mixed strategy (For further details, we refer the reader to [Motwani &

Raghavan 2010]). If *a* is such a mixed strategy probability distribution for the row player, and *b* for the column player, then there exists value of the game *V* such that *(i) a* guarantees an expected payoff of *at least V* to the row player, no matter what the column player does; and *(ii) b* guarantees an average negative payoff of *at most V* to column player, no matter what the row player does. In case of a zero-sum game, the Minimax strategy is also a Nash Equilibrium.

1.4.2.5 Stackelberg security games

Stackelberg games are leader-follower games. The game is derived from the Stackelberg leadership model in economics in which the leader firm moves first and then the follower firm(s) moves sequentially. The leader knows *ex ante* that the follower observes his action. Stackelberg security games are a subset of Stackelberg games which focus on the attacker-defender model. In general, they involve scheduling of scarce defense resources to cover a subset of the potential targets [Letchford 2013].

1.5 Scope and organization of the thesis

The P2P paradigm has moved beyond its boundaries of file sharing, and has seen deployment for several applications such as the SETI @ home project, Spotify, LiveStation, etc. It is clear that the P2P architecture is here to stay, and hence there is need for the current security and intrusion detection mechanisms to be more 'P2P-aware'. This need arises from the fact that P2P networks behave differently from the traditional systems in several aspects, such as the lack the traditional client-server architecture, peer-churn (the joining and leaving of peers), security issues in a distributed and decentralized environment, etc.

The objective of this thesis is to build effective intrusion detection mechanisms

which are 'P2P aware'. The thesis presents mechanisms for intrusion detection in P2P networks that: *(i)* can segregate malicious (botnet) P2P traffic from benign P2P traffic based on their network behavior, *(ii)* are resilient to evasive attacks where an adversary may try to evade detection by deliberate injection of noise in his communication patterns, *(iii)* propose strategic deployment of IDS in a P2P network with a randomized, game-theoretic approach, and *(iv)* propose a scalable and distributed, Hadoop-based framework for the detection of P2P botnets.

Given their widespread use, security in P2P networks has naturally attracted a lot of attention from researchers. In Chapter 2, we discuss related work and past research efforts on detection of attacks and security threats in P2P networks.

In Chapters 3 and 4, we present our proposed approaches for the detection of P2P botnet traffic in the presence of benign P2P traffic at a network perimeter. Our approaches do not assume the availability of any 'seed' information of bots through blacklist of IPs. They do not rely on Deep Packet Inspection (DPI) or signature-based mechanisms which are rendered useless by botnets/applications using encryption. They aim to detect the *stealthy* behavior of P2P botnets on the basis of their 'P2P' behavior and C&C communication with other bots, when they lie dormant (to evade IDS 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.

Chapter 5 considers the problem of securing a P2P network from an adversary who may become part of the P2P network by joining from any part of the network. We explore the problem of strategically positioning IDS in a P2P network with a game theoretic approach. Our approach distributes the responsibility of running the IDS between the peers in a randomized fashion and minimizes the probability of a successful attack.

Chapter 6 presents a scalable, Hadoop-based framework for the detection of P2P

botnets which extracts statistical features *per host* for all P2P hosts involved in network communication, and uses them to train supervised machine learning models which can differentiate P2P botnets from P2P applications. We propose a distributed data collection architecture wherein data collectors are distributed at multiple locations inside an enterprise network. This allows *inside-to-inside* communication view, which can be vital for detecting smart P2P bots inside a network which communicate to each other over LAN.

We conclude the thesis and present future work in Chapter 7.

Chapter 2

Literature survey

We begin this chapter by discussing the security threats and attacks which plagued P2P networks in their early days (Section 2.1). These attacks have been greatly studied by a number of researchers over the past decade, and many proposed solutions and improved implementations of P2P applications have resolved these security threats to a considerable extent. Next, in Section 2.2, we expound prior efforts on the detection of P2P botnets and present their shortcomings. Further, in Section 2.3, we look at past research in P2P networks from a perspective of game theory, and discuss past work on incentives for sharing, modeling of malicious peers, security games and other related topics.

2.1 Security threats and attacks – early challenges

Any decentralized and distributed computing environment naturally involves many challenges of security, and P2P overlay networks are no exception. In this section, we will review some of the prominent attacks and security threats which were prevalent in P2P networks in their early days (the first decade of the 21st century). However, this list is not exhaustive. For a taxonomy of attacks on P2P

overlays, we refer the reader to [Yue *et al.* 2009] and [Trifa & Khemakhem 2012].

2.1.1 Identity assignment attacks

Peers participate in P2P networks by assuming virtual identities. As a matter of fairness, it is assumed that one physical entity shall own one random virtual identity in the P2P network by which it will participate in network activity. However, a malicious peer may attack this identity assignment principle by creating multiple identities referred to as 'sybil' identities [Douceur 2002]. By positioning its 'sybil' nodes at strategic positions within the network, an attacking peer can try to gain illegitimate control over the network or its part. Sybils can cause multiple damages. By monitoring the traffic flowing through the sybil nodes, the attacker can observe the traffic patterns of other peers whose traffic is flowing through them, and can also attempt to misuse the communication protocol in other ways. Sybil nodes may also be used to forward routing or search requests to other malicious/sybil nodes, thereby disrupting the normal functioning of the network. The attacker can also gain control over certain files shared in the network and can choose to deny access to them or corrupt them.

In DHT-based P2P networks, the nodes having the ID closest to the ID of a particular file are responsible for maintaining information about that file. Hence, an attacker could possibly control the availability of a certain resource if he maps his IDs very close to that resource. [Locher *et al.* 2010] demonstrated the possibility of corrupting the information in the DHT-based Kad network by spreading polluted information through a 'Node insertion' or 'ID mapping' attack. The attacker may carry out this attack to pollute information about keywords, comments or list of sources. [Singh *et al.* 2006] implemented an attack for keywords where the attacker manipulates the ID assignment mechanism of Kad and generates an ID of his own, which lies very close to the targeted file-keyword. Hence, when a search

is carried out for that particular keyword, it does not yield correct results. Rather, the bogus information planted by the attacker is returned.

A number of approaches have been suggested to defend P2P systems against sybil attacks. One of the early work in this regard is the use of computational puzzles [Borisov 2006]. [Haribabu *et al.* 2010] evaluate the use of CAPTCHAs in detecting sybils. An approach for detecting sybils using 'psychometric tests' is proposed in [Haribabu *et al.* 2012]. Authors in [Yu *et al.* 2006] have argued that a malicious user can create many identities, but only a few relationships of trust. They use this fact to locate disproportionately-small 'cuts' in the graph between the sybil nodes and the honest nodes, and attempt to bound the number of identities a malicious user may create.

2.1.2 Routing attacks

2.1.2.1 Route table poisoning attacks

Due to high rate of churn (joining and leaving of peers) in P2P networks, peers have to regularly update their routing tables in order to perform correct routing lookup. Peers create and update their routing tables by consulting other peers. If a malicious node sends false information in its routing table update, it will corrupt the routing table entries of benign nodes, leading to queries being directed to inappropriate nodes or non-existent nodes.

Different solutions have been developed to counter route table poisoning by imposing certain requirements on the participating peers. In the Pastry network [Rowstron & Druschel 2001], each entry in routing tables is preceded by a correct prefix, which cannot be reproduced by a malicious entity. The CAN network [Ratnasamy *et al.* 2001] considers the round-trip-time in order to favor lower latency paths in routing tables. [Condie *et al.* 2006] propose 'induced' churn as a counter

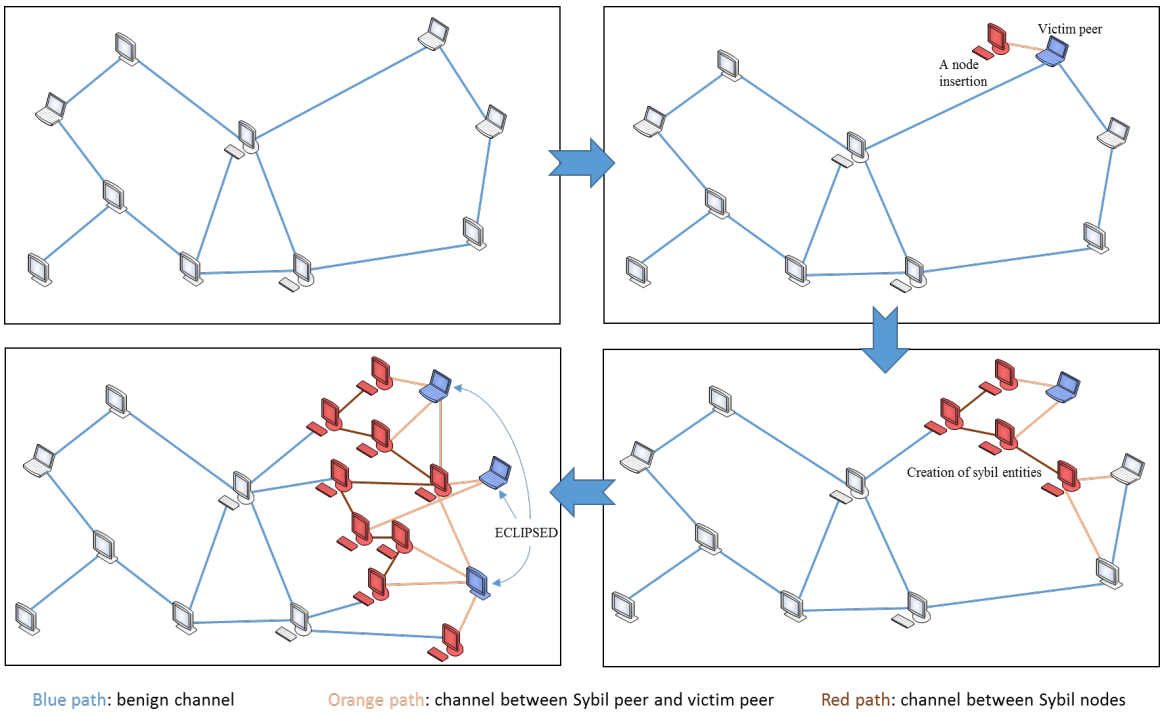


Figure 2.1: Node insertion attack and sybil attack escalated to an eclipse attack

against such attacks.

2.1.2.2 Eclipse attacks

Each node in the P2P network maintains overlay links to a set of neighbor nodes, and each node uses these links to perform a lookup from its neighbors. Thus, an attacker may be able to control a significant part of overlay network by controlling a large part of the neighbors of legitimate nodes. This is known as an eclipse attack. Eclipse attacks are *escalated* forms of identity assignment attacks or route table poisoning attacks described above. If an attacker is able to generate a large number of fake identities and place those identities in the overlay network, he could mediate most overlay traffic, and thus *eclipse* legitimate nodes from each other. A pictorial representation of node insertion attacks and sybil attacks being escalated into an eclipse attack is given in Figure 2.1

[Steiner *et al.* 2007] describe the eclipsing of search keywords in Kad P2P network.

Their experiment explains that an eclipse attack and a sybil attack can be performed quite similarly, except that the Kad ID space covered for an eclipse attack is much smaller. For eclipsing a certain keyword k in a DHT-based network such as Kad, the authors in [Steiner *et al.* 2007] choose to position a certain number of sybils as close as possible to the keyword k. Then, the sybil nodes are announced to the benign peers in the network which has the effect of 'poisoning' the regular peers' routing tables for k and attract all the route requests for k to the sybil nodes. [Steiner *et al.* 2007] claim through their experiments that as few as eight sybil peers were sufficient to ensure that any request for k terminates on one of the sybil nodes which were strategically planted in the network to lie as close as possible to k. Since all requests for k go via the sybil nodes, the attacker has effectively *eclipsed* the keyword k from all peers in the Kad network.

The developer community of eMule (a P2P application which uses the Kad network) had responded to these security threats by making suitable changes to their applications. For example, the 'change-log' for eMule's version 0.49a released in February 2008 (available at www.emule-project.net) states, "Kad will now enforce certain limits when adding new contacts to the routing table: No more than 1 KadNode per IP, 2 similar KadNodes (same bin) from a /24 network and max 10 different KadNodes from a /24 network are allowed". Some countermeasures have also been proposed in literature, such as an optimized routing table and verified routing table [Castro *et al.* 2002], and a combination of induced churn and one-hop redirection [Puttaswamy *et al.* 2009]. [Zhang *et al.* 2011b] presented a scheme for generating node IDs which requires a user to solve a computational puzzle generated by their network parameters together with time-related information. The authors claim that such an ID generation mechanism makes ID assignment and eclipse attacks computationally infeasible for an attacker, while levying only a small overhead on a regular user.

The implementation of many such countermeasures made large eclipse attacks very difficult or computationally prohibitive for an attacker. But, the emergence of P2P botnets provided a new ground for the adversaries to launch large-scale attacks without the time and effort required for an eclipse attack. P2P botnets can either be created on an existing P2P network's protocol (for example, Storm used the Overnet P2P network [Stover *et al.* 2007]) or they may use a custom P2P protocol to build their own network of infected hosts (for example, Sality [Falliere 2011]). The bot-master does not need to create multiple sybil identities or perform node insertions. A victim machine, upon infection from a malware or trojan, joins and becomes a part of the P2P network of the botnet. The decentralized architecture also helps in evading detection. We will cover P2P botnets in more detail in Section 2.2.

2.1.3 Application level attacks

2.1.3.1 Index poisoning attacks

P2P file-sharing involves the use of 'indices', which are used by peers to search and find locations of files desired by them. If an attacker inserts massive numbers of bogus records into the index for a set of targeted titles, it is called as an index poisoning attack [Liang *et al.* 2006]. When a legitimate peer searches for a targeted title, the index will return bogus results, such as bogus file identifiers, bogus IP addresses, etc.

Key defense against index poisoning is to provide a verifiable identity. This may be done through Public Key Infrastructure (PKI) cryptography, as in [Berket *et al.* 2004]. However, PKI requires the presence of a trusted certificate authority, which is not always feasible in a P2P network. Even if a certificate authority is implemented, it will face huge workload due to a large number of peers as well

as high peer-churn.

2.1.3.2 Query flooding

Flooding of search queries is basic search mechanism employed in many P2P networks. A peer looking for a resource will broadcast its query to its immediate neighbors. If a neighbor does not have the resource, the query is further forwarded to its neighbors. This goes on until the 'hop limit' is reached or the resource is found. Malicious peers can exploit this flooding mechanism to generate a query flooding attack. A malicious user may generate as many useless queries as possible to flood the network. As a result, the resources of the network will be engaged in serving these requests, and benign users may face service degradation.

A solution for preventing query flooding in Gnutella was implemented by putting an upper limit on the number of queries a node can accept from a requesting peer [Daswani & Garcia-Molina 2002]. After this number is reached, all other queries are dropped.

2.1.3.3 Pollution attacks

Since peers participate in the P2P network using virtual identities, it is easy for an attacker to spread unusable or harmful content without getting the risk of getting caught. Peers are engaged in downloading chunks of files from different sources. An attacker may host a file x with himself, but replace the original contents with some junk pieces of data or some malware. When a peer, looking for file x, downloads the content from the attacker, he receives 'polluted' content in the form of a corrupted or malicious file. Hash verification and chunk signing are some of the common approaches used against pollution of content in P2P networks [Liang *et al.* 2005, Dhungel *et al.* 2007].

2.1.3.4 Rational attacks

P2P networks are based on the principle of peers cooperating to mobilize and share resources. If peers are unwilling to cooperate, the P2P network will fail to survive. A self-interested node, however, may attempt to maximize the benefits it receives from the network while minimizing the contributions it makes. This has been described as a rational attack. A “free-rider" is a term used in P2P systems to describe a peer who utilizes the services and benefit of the network, but does not make any contributions. Since P2P networks are inherently about collaboration and sharing of resources, the services of the network face degradation if most of the users behave in a selfish way and choose to free-ride. Since modeling of ‘selfish’ free-riders has often been studied from a game theoretic perspective, we will discuss more on it in Section 2.3.

2.2 Detection of P2P botnets

2.2.1 Early attempts

The distributed and decentralized nature of P2P networks offered a lucrative alternative to bot-masters to build botnets which are not prone to any single point-of-failure. P2P botnets have also been proven to be highly resilient against take-down attempts [Rossow *et al.* 2013]. Moreover, smarter bots are stealthy in their communication patterns, and elude the standard discovery techniques which look for anomalous network or communication behavior.

Initial work on detection of P2P botnets involved signature-based and port-based approaches [Schoof & Koning 2007]. Such solutions—which rely on DPI and signatures—can easily be defeated by bots using encryption. [Schoof & Koning 2007] deployed and analyzed the Sinit and Nugache P2P bots on their testbeds.

Their analysis of Nugache reports that it uses a static list of 22 IP addresses to connect upon initialization and listens on port 8 (an unassigned port), while Sinit starts probing randomly chosen IP addresses and communicates always on port 53. Although such information is valuable to IDS and firewalls, this approach is 'signature-based' and does not guarantee to catch the variants/successors of these botnets. Moreover, botnets which use encryption will easily evade such detection mechanisms.

Most prior work has either focused on P2P traffic classification from the perspective of a more general problem of Internet traffic classification [Sen *et al.* 2004, Li *et al.* 2008, Iliofotou *et al.* 2009], or has given special attention to detection of botnets (centralized or distributed) in Internet traffic [Gu *et al.* 2008a]. The challenging context of detection of stealthy P2P botnets in the presence of benign P2P traffic has not received much attention. Furthermore, building a scalable detection framework did not receive much focus during early research, and has received very little attention even in recent research (such as in [Zhang *et al.* 2014, Singh *et al.* 2014]).

2.2.2 Statistical approaches

An early work using machine learning techniques was performed by [Livadas *et al.* 2006], who attempted to identify the 'command & control' traffic of IRC (Internet Relay Chat) botnets in particular. Their approach used Naïve Bayes classifier to first segregate the traffic into IRC and non-IRC traffic, and then classify the IRC traffic as malicious or non-malicious. Although their classifier gave good results on test data, it performed poorly in classifying the IRC botnet flows generated by authors on their testbed.

For the detection of P2P botnet traffic, some of the recent work has used su-

pervised learning approaches [Saad *et al.* 2011, Rahbarinia *et al.* 2014, Narang *et al.* 2013, Singh *et al.* 2014], unsupervised learning approaches [Zhang *et al.* 2011a, Zhang *et al.* 2014] and other statistical measures [Noh *et al.* 2009, Yen & Reiter 2010].

[Noh *et al.* 2009] used a 'multi-flow' model to detect P2P botnets. Their approach tried to detect similar flows occurring between groups of hosts in a network on a regular basis. Flows with similar behavior were labeled into groups to construct a transition model of the groups using a probability matrix. 'Likelihood ratio' metric was employed by the authors to detect bots inside the network. Their approach suffers from a drawback that it will be unable to identify bots inside the network if their number happens to be very small.

[Yen & Reiter 2010] attempted to segregate P2P bots from benign P2P apps based on metrics of a host such as the volume of data exchanged and number of peers contacted. Unfortunately, it is difficult to generalize their findings since they are based on the data of a single botnet. Also, the features used by the authors are not sufficient to correctly differentiate P2P bots and apps, and their approach fails to detect bots when bots and apps run on the same machine.

[Saad *et al.* 2011] studied the application of five commonly used machine learning algorithms for online detection of P2P botnet traffic. The algorithms used by them were – Support Vector Machines, Artificial Neural Networks, Nearest Neighbor Classifier, Gaussian-based classifier and Naïve Bayes Classifier. Their experimental results showed that traffic behavior of botnets is effective in detecting botnets during their C&C phase. The authors, however, argued that online botnet detection requires algorithms to be adaptable and have novel and early detection, but none of the studied techniques address all the requirements at once.

PeerRush [Rahbarinia *et al.* 2014] 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.* 2011a, Zhang *et al.* 2014], on the other hand, 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.

A similar work is BotSuer [Kheir & Wolley 2013], which proposes a behavioral approach to detect P2P botnets. Their work relies on P2P 'control flows' and creates clusters of those P2P malware which implement similar functionality. At a high-level, their work uses the following bidirectional features extracted from each flow – number of packets sent and received, bytes sent and received, and byte rate sent and received. The authors claim that their approach can detect even single infected bot inside a network perimeter. However, their work refrains from providing sufficient justification for this claim.

[Singh *et al.* 2014] presented a 'Big data analytics' framework for the detection of P2P botnets. Using certain well-known flow-based features, the authors utilized the Random forest classifier to build detection models using the Hadoop framework. However, the detection results obtained by the authors are for a small botnet dataset. Hence, it is difficult to say whether their detection methodology will generalize to other P2P botnets.

2.2.3 The notion of 'conversations'

Most of the past works have employed the classical five-tuple categorization of network flows. Indeed, one of our preliminary work [Narang *et al.* 2013] also utilized five-tuple categorization of flows to study the impact of feature selection on

detection of P2P botnets. Packets were classified as 'flows' based on the five-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 [Karagiannis *et al.* 2005] and even in the early days of P2P traffic classification [Karagiannis *et al.* 2004]. 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 refer to this aspect.

In response to this, some recent work has utilized super-flow and conversation based approaches which are port-oblivious and protocol-oblivious. [Hang *et al.* 2013] used 2-tuple 'super-flows' based approach with a graph-clustering technique to detect P2P botnet traffic. Although authors in [Hang *et al.* 2013] presented interesting insight using super-flows 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 exhibits many similarities to benign P2P traffic. 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.

Similar to the notion of 'super-flows' used by [Hang *et al.* 2013], work of [Li *et al.* 2013] used 'conversation-based' approach for the detection of overlapping

P2P communities in Internet backbone. Their work, however, is directed on a different problem. They do not focus on identification of any specific P2P application – whether malicious or benign – and limit themselves to identification of P2P communities in the Internet backbone. The notion of 'conversation-based' approaches for detection of P2P botnets was first seen in [Zhang 2013]. But the approach chosen by authors in [Zhang 2013] has certain practical limitations. The authors created 'thirty second conversations' from network traces and extracted statistical features for the detection of P2P botnets. If a bot is stealthy in its communication patterns, a small time window of thirty second will fail to observe any botnet activity.

2.2.4 Limitations of past efforts & issues addressed in this thesis

Many previous solutions such as BotMiner [Gu *et al.* 2008a] detect botnets by observing noisy activities such as spamming or DDoS attacks generated by multiple infected hosts inside the network perimeter. In contrast, our approaches (in Chapters 3 and 4) can detect stealthy P2P botnets which perform malicious activities in a manner not observable to a network administrator, and which try to lie low and generate little traffic in order to evade detection by Intrusion Detection Systems. Certain other approaches such as BotSniffer [Gu *et al.* 2008b] and "Friends of an enemy" by [Coskun *et al.* 2010] rely on correlating activity amongst multiple hosts in order to detect hosts infected by bots. Our approaches do not require activity correlation or presence of multiple infected hosts, and can detect individual infections inside a network. Moreover, the challenging context of detection of P2P botnets in the presence of traffic from benign P2P applications was not addressed by several past solutions such as BotGrep [Nagaraja *et al.* 2010], BotTrack [François *et al.* 2011] or Botyacc [Nagaraja 2014]. It has received only a little attention [Kheir & Wolley 2013, Rahbarinia *et al.* 2014, Zhang *et al.* 2014]. Our

approaches presented in this thesis address this challenging context.

None of the past works employing super-flow or conversation-based approaches [Hang *et al.* 2013, Li *et al.* 2013] 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 miss certain finer details in the process.

Our work in this thesis, presented in Chapter 3, presents a 'best of the both worlds' approach utilizing flow-based approaches as well as conversation-based approaches in a two-tier architecture. It begins with the *de facto* standard of five-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. Further, these features are used to build supervised machine learning models which can accurately differentiate between benign P2P applications and P2P botnets. Our system was extensively evaluated with real-world traces of P2P applications and botnets. Our approach can effectively detect activity of *stealthy* P2P botnets even in the presence of benign P2P applications in the network traffic. It could also detect *unknown* P2P botnets (i.e., those not used during the training phase) with high accuracy.

Furthermore, the context of P2P botnet detection is adversarial in nature. Statistical or behavioral models for detection are created using 'botnet' data which has been generated by an adversary. Hence, the adversary is in a position to evade the detection mechanisms if he can change the behavior of his bots. Thus, this

necessitates the evaluation of the performance of these detection models in the presence of noise injected by an adversary. That is, if the bot-master slightly alters the behavior and communication patterns of the bots, are these detection models *robust* and *resistant* towards such a change? To the best of our knowledge, this question has not received sufficient attention. We attempt to address this context in our work in Chapter 4.

Our approach utilizes conversation-based mechanisms and attempts to enhance them with techniques of Discrete Fourier Transforms (DFTs) and information entropy by leveraging the *timing* and *data patterns* in P2P botnet traffic. We extract two-tuple conversations from network traffic and treat each conversation as a time-series sequence (or a 'signal'). We leverage on the fact that communication of bots amongst each other follows a certain regularity or periodicity with respect to timing and exchange of data. Bots tend to repeatedly exchange same kind of commands—which are often in the same format and of the same size. The repeated C&C communication also follows certain timing patterns—with bots generally contacting their fellow peers at predefined intervals [Tegeler *et al.* 2012]. In order to uncover the hidden patterns between the communications of bots, we convert the *time-domain* network communication to the *frequency-domain*. From each conversation, we extract features based on Fourier transform and information entropy. We use real-world network traces of benign P2P applications and P2P botnets to compare the performance of our features with traditional flow-based features employed by past research (such as [Livadas *et al.* 2006, Saad *et al.* 2011, Kheir & Wolley 2013, Zhang *et al.* 2014]). We build detection models with multiple supervised machine learning algorithms. We inject noise in our test data to demonstrate that our detection approach is more resilient towards variation in data or introduction of noise in the data by an adversary. With our approach, we could detect P2P botnet traffic in the presence of injected noise with True Positive rate as high as 90%.

We would like to draw a comparison of our approach with some recent works that have also employed similar 'signal-processing' approaches in the domain of botnet detection. [Kang & Zhang 2009] applied entropy theory for the detection of the Storm P2P botnet. A significant difference between our approach and the approach of [Kang & Zhang 2009] is that they used entropy over the payload content by employing DPI, which is not suitable for botnets which use encryption. [Yu *et al.* 2010] used DFTs on 'feature streams' for the detection of botnets. Their approach employed Snort (a popular Network Intrusion Detection System), which also uses DPI and will not be effective for botnets which use encryption. Further, their approach was primarily targeted towards the online detection of IRC bots, which is conceptually different from the problem of detection of stealthy P2P bots addressed in our work. BotFinder [Tegeler *et al.* 2012] is a recent work which is conceptually similar to ours. BotFinder applies DFT over a binary sampling of the C&C communication, whereas our work aims to leverage the hidden traffic regularities in C&C communication by applying DFT over payload lengths and inter-arrival times. Further, BotFinder was evaluated only for the detection of IRC bots. Our work addresses detection of stealthy P2P botnets by focusing on their 'P2P' C&C communication.

Further, in Chapter 6, we present a scalable, Hadoop-based framework for the detection of P2P botnets which extracts statistical features *per host* for all P2P hosts involved in network communication. Although some past research have utilized host-level features [Zeng *et al.* 2010, Yen & Reiter 2010], these approaches did not consider the problem from the perspective of a scalable framework. Our approach relies on the header information in the network and transport layer, and extracts statistical features which quantify the 'P2P' behavior of the P2P applications running on a host. Statistical features are extracted *per host* for all P2P hosts involved in network communication, and are then used to train supervised machine learning models which can differentiate P2P botnets from P2P applications.

We propose a distributed data collection architecture wherein data collectors are distributed at multiple locations inside an enterprise network and sit close to the peers, say at an Access switch or a Wi-Fi access point. This allows *inside-to-inside* communication view, which can be vital for detecting smart P2P bots inside a network which communicate to each other over LAN.

2.3 Game theoretic perspective

P2P networks have been studied from the game theoretic perspective mainly with regard to incentives for sharing [Anceaume *et al.* 2005], collaboration [Ye *et al.* 2004], managing trust [Kamvar *et al.* 2003], etc. Modeling of malicious behavior in this context has received much less attention from the research community. Security in P2P networks *per se* has not received much attention from a game theoretic perspective. However, the topic of network security with game theory has attracted a lot of attention. [Manshaei *et al.* 2013] provide a good survey of the same.

2.3.1 Incentives for collaboration

Conventional P2P networks did not provide service differentiation and incentives for users. Therefore, users could easily obtain resources without contributing any information or service to the P2P community. This led to the well-known free-riding problem. Consequently, most of the information requests are directed towards a small number of P2P nodes which are willing to share information or provide service, causing the "tragedy of the commons" [Ma *et al.* 2006].

In [Gatti *et al.* 2004], authors model peers as participants in the system who have opposing interests. They can choose the level of resources they want to devote

in fighting each other. A malicious peer gains utility from thwarting a particular transaction, such as file sharing, document retrieval, etc., while a benign peer derives utility from succeeding with the transaction. To secure such a network, it should be sufficient to ensure that the cost of an attack is much more than the benefit derived by an attacker in launching that attack.

In [Gupta & Somani 2005], the authors devise a model to combat free-riding. They derive a Nash equilibrium based proof for the probability with which peers will serve (or not serve) other peers in the network. The authors prove that even in case of selfish peers, it is in their best interest to serve.

In [Feldman *et al.* 2006], the authors devise a model to study the phenomenon of free-riding and free-identities in P2P systems. They model their user as a player who decides whether to contribute or to free-ride based on how the current contribution cost in the system compares to his/her 'type', where the 'type' can be intuitively thought of as a quantitative measure of decency or generosity. The solution proposed by the authors for dealing with free-riding is that imposing penalty on all users that join the system is effective under many scenarios.

2.3.2 Modeling malicious peers

Authors in [Moscibroda *et al.* 2006] model peers in a P2P network by considering *all* peers as selfish individuals whose only goal is to optimize their own benefit. Their model also considers malicious or 'byzantine' peers who want to attack the system, destabilize the network, deteriorate the overall performance, etc. This way, their model considers a P2P system with selfish peers and byzantine players, and evaluates the impact of presence of such byzantine players on a selfish system's efficiency.

In [Theodorakopoulos & Baras 2007, Theodorakopoulos & Baras 2008], the au-

thors model malicious users in unstructured networks with a repeated game model, considering that the users are either good or bad, and not selfish or unselfish. All peers are considered as pay-off maximizing strategic agents. The 'good peers' are always cooperative, and they value the network's benefit more than their individual profits or losses. Whereas the malicious or bad peers aim at disrupting the functioning of the network and waste the resources of the good peers. With this model, the authors find the Nash equilibrium for the strategies of legitimate and malicious peers, by which they identified those strategies of the legitimate users which enforce upper bounds on the damage that the malicious users can cause to the network.

2.3.3 Security games

A security game is a two-player game between a defender and an attacker. Security games provide an analytical framework for modeling the interaction between malicious attackers who aim to compromise networks, and owners or administrators defending them. The 'game' is played on complex and interconnected systems, where attacks exploiting vulnerabilities as well as defensive countermeasures constitute its moves. The strategic struggle over the control of the network and the associated interaction between attackers and defenders is formalized using the rich mathematical basis provided by the field of game theory. The underlying idea behind the game theoretic models in security is the allocation of limited available resources from both players' perspectives. Game theoretic solutions have been well-applied to security in different areas of network security.

In [Bensoussan *et al.* 2010], the authors analyze the economic aspects of botnet activity and suggest feasible defensive strategies. They provide a comprehensive game theoretical framework that models the interaction between the botnet herder and the defender group (network/computer users). The authors propose that a

botnet herder's goal is to intensify his intrusion in a network of computers for pursuing economic profits whereas the defender group's goal is to defend botnet herder's intrusion. The authors claim that their equilibrium solution in which the botnet herder exerts a constant attack and the defender maximizes his defense level is consistent with the real world observations for the Conficker botnet which targeted the Microsoft Windows operating system. The latest variants of Conficker (Conficker D and Conficker E) are known to be P2P-based for their update propagation.

In [Chen & Leneutre 2009], the authors formulate the network intrusion detection as a non-cooperative game and perform an in-depth analysis on the Nash equilibrium and the engineering implications behind. Based on their game theoretical analysis, the authors derive the expected behaviors of rational attackers, the minimum monitor resource requirement, and the optimal strategy of the defenders. The authors have also provided guidelines for IDS design and deployment, and elaborated on the feasibility of the game theoretical framework to be applied to configure the intrusion detection strategies in realistic scenarios.

2.3.4 Issues addressed in this thesis

Although the decentralized and distributed nature of P2P network offers resilience towards network-breakdowns, the super-peer architecture is more sensitive in this regard since an adversary can disrupt (albeit not breakdown) the entire P2P network by attacking the super-peer nodes. For example, a DoS/DDoS attack targeted on relay nodes in Tor can lead to an increased latency and higher number of time-outs in the network. In our approach presented in this thesis (in Chapter 5), we consider the problem of securing a super-peer based P2P network from an adversary who may become part of the P2P network by joining from any part of the network. A malicious peer can disrupt the P2P network by attacking a super-

peer through various attacks at the overlay layer, such as route table poisoning, index poisoning, or other traditional attacks (malicious payloads, etc.). Running an IDS at each peer may not be feasible since self-interested peers may not want to dedicate resources for that. Peers may try to secure the network by running IDS at certain strategic locations in the network. But, a deterministic schedule of running and positioning the IDS can be observed and thwarted by an adversary. In our work, we explore the problem of strategically positioning IDS in a P2P network with a game theoretic approach. Our approach distributes the responsibility of running the IDS between the peers in a randomized fashion and minimizes the probability of a successful attack.

From a network security perspective, two works close to ours are [Kodialam & Lakshman 2003] and [Vaněk *et al.* 2012]. Both of these consider optimal resource allocation by a defender in a network against potentially malicious packets by adopting a game theoretic approach of inspecting only a fraction of all packets. The work of [Kodialam & Lakshman 2003] was limited to a single source and single target, whereas [Vaněk *et al.* 2012] considered multiple targets. Our work deals with a different problem of game theoretic strategies for IDS deployment in a P2P network. By running IDS at strategically chosen nodes in the P2P network, we aim to conserve the resources of participating peers and minimize the gain of an attacker. This model is also applicable to 'collaborative IDS' involved in 'P2P intrusion detection'. [Janakiraman *et al.* 2003] presented a collaborative, P2P approach for building a distributed, scalable IDS amongst trusted peers. [Locasto *et al.* 2005] deployed a decentralized system for efficiently distributing alerts to collaborating peers by creating a distributed 'watch-list' from alert streams. [Duma *et al.* 2006] explore the challenge of collaborative 'P2P intrusion detection' from the perspective of insider threat. While these works have proposed techniques for sharing information between trusted peers [Janakiraman *et al.* 2003], distributing alerts among collaborating peers [Locasto *et al.* 2005] and managing trust to

address insider threats [Duma *et al.* 2006], we address the issue of strategic deployment of IDS within a P2P network. Past research on different aspects of 'P2P intrusion detection' stands to gain from such strategic deployment of IDS since a deterministic schedule of running or positioning the IDS might be observed and thwarted by an adversary.

Chapter 3

Flow-clustering and conversation-generation for P2P botnet detection

3.1 Introduction

Detection of P2P botnets by analysis of their network behavior has frequently utilized 'flow-based' mechanisms [Saad *et al.* 2011, Narang *et al.* 2013, Singh *et al.* 2014]. Due to certain limitations of these approaches in identifying modern P2P applications (discussed in Chapter 2), alternatives have been proposed in the form of super-flow-based and conversation-based mechanisms. However, even these approaches are not yet mature and suffer from certain drawbacks.

In this chapter, 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 for network admin-

istrators to segregate unwanted P2P traffic and detect P2P botnets.

`PeerShark` 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` does not assume the availability of any 'seed' information of bots through blacklist of IPs. It does not rely on DPI or signature-based mechanisms which are rendered useless by botnets/applications using encryption.

`PeerShark` begins with the *de facto* standard of 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' approach 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 IP/TCP/UDP headers, and does not require DPI. Thus, it 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, which has received little attention in past research (such as [Zhang *et al.* 2011a, Zhang *et al.* 2014]).
- We extensively evaluate our system `PeerShark` with real-world P2P botnet traces. `PeerShark` also detects *unknown* P2P botnets (i.e., those not used during the training phase) with high accuracy. This approach appears in [Narang *et al.* 2014a].

This chapter is organized as follows: in the next section (3.2), we discuss the system design of `PeerShark`. Section 3.3 gives the details of design choices and implementation of `PeerShark`, followed by its evaluation in Section 3.4. In Section 3.5, we discuss the limitations and possible evasions of our approach, and briefly mention about multi-class classification. We conclude this chapter in Section 3.6.

3.2 System design

P2P botnets engage in C&C using custom or well-known P2P protocols. As a result, their traffic can *blend in* with benign P2P traffic flowing in a network and thus pass undetected through IDS 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 3.4, 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.

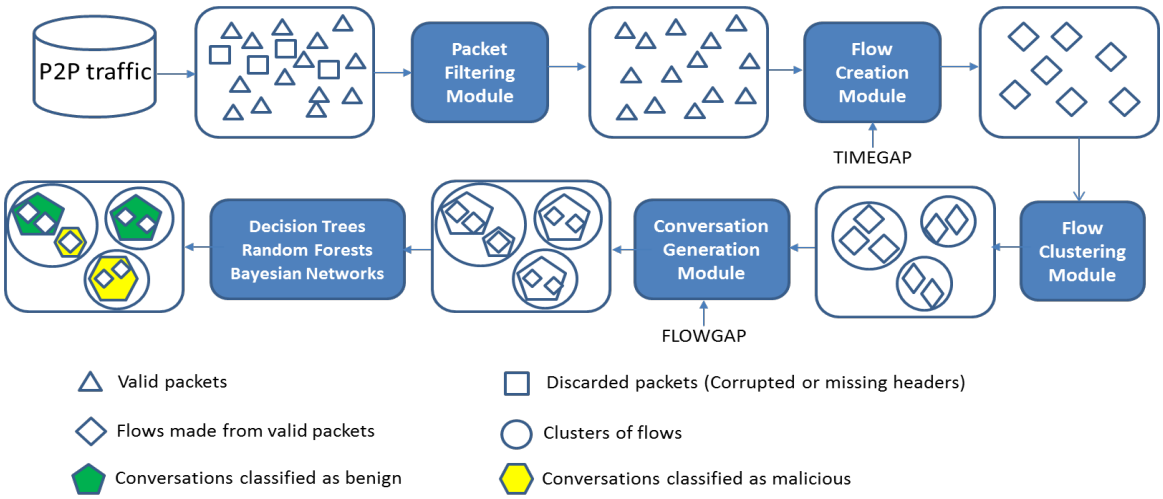


Figure 3.1: PeerShark: architecture

Figure 3.1 gives the architecture of `PeerShark`. The system design of `PeerShark` is explained here:

3.2.1 Flow-clustering phase

Flow-based analysis has been the *de facto* standard for Internet traffic classification and has yielded great success in the past [Karagiannis *et al.* 2004, Karagiannis *et al.* 2005]. Typically, 'flows' are constructed based on the 5-tuple: source IP, destination IP, source port, destination port and 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 bidirectional in nature. This differentiating factor has been leveraged by some recent works [Rahbarinia *et al.* 2014, Narang *et al.* 2013] 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 (obtained from [Rahbarinia *et al.* 2014]). 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)
- 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 & payload exchanged. A more detailed discussion on the choice of features and clustering algorithm will follow in Section 3.3.

3.2.2 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 each other in the P2P overlay network maintain regular communication amongst themselves to check for updates, exchange commands, and/or 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; e.g., 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. Two-tuple conversations provide a *bird's eye view* of the communication happening between different hosts in the network, which can be beneficial for a network administrator to hunt for malicious conversations in the network traffic.

As has been explained in Chapter 2, the present works utilizing conversation-based (or super-flow based) approaches do not detect systems infected by a P2P bot if the system runs benign P2P applications as well. The main reason behind this flaw is that conversations attempt to provide a *bird's eye view* of the network activity to the network administrator, but miss out certain finer details in that process. Since all flows between two IPs are aggregated into a single conversation, this approach creates a single conversation for two IPs having malicious and benign flows between them, and thus fails 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 earlier work with conversation-based approaches, 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, 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.

3.3 Design choices and implementation details

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

3.3.1 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 [Rahbarinia *et al.* 2014]. The data for P2P applications was generated by [Rahbarinia *et al.* 2014] by running those applications in their lab environment for a number of days, using AutoIt[1] scripts to simulate human-user activity on the P2P hosts. The data of P2P botnets corresponds to real-world traces obtained from third parties. We also obtained real-world traces of another P2P botnet named Nugache from the authors of [Masud *et al.* 2008]. Altogether, we used four bots and four apps for this work.

As mentioned in the previous section, network traces of each application were parsed to create flows and further generate conversations. 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 P2P botnet, 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

[1]https://www.autoitscript.com/site/autoit/

Table 3.1: Dataset details

Name	No. of flows	No. of conversations	Purpose
eMule	4,13,995	2,93,704	Train/Test
uTorrent	14,09,291	4,58,624	Train/Test
Vuze	12,07,963	6,03,145	Train/Test
Frostwire	8,90,300	2,34,335	Train/Test
Storm	95,316	59,157	Train/Test
Waledac	81,778	5,765	Train/Test
Zeus	43,593	2,751	Validation
Nugache	51,428	49	Validation

network traces are benign or malicious[1]. Hence, 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 the conversation as benign. Full details of this dataset are given in Table 3.1. Our training/testing datasets are representative of the real-world, where the majority of traffic flowing in any network is benign [Cisco 2014]. Our datasets contain more than 90% benign traffic.

3.3.2 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. From each packet, its 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

[1]Personal communication with Babak Rahbarinia [Rahbarinia *et al.* 2013] in November 2013

TCP packets because TCP flags are required to construct flows.

This module is explained in Algorithm 3.1.

Algorithm 3.1: Packet parsing module

```
   Data: packetCapture
   Result: filteredPackets
1  begin
2      ArrayList < ModifiedPkt > filteredPackets;
3      for Packet pkt in packetCapture do
4          ts ← pkt.getTimestamp();
5          if pkt has IPHeader then
6              ip ← pkt.getIPHeader();
7              IP1 ← ip.getSourceIP();
8              IP2 ← ip.getDestIP();
9              if pkt has TCPheader or UDPheader then
10                 IP1_port ← pkt.getSourcePort();
11                 IP2_port ← pkt.getDestPort();
12                 header ← pkt.getTransportHeader();
13                 pSize ← header.getPayloadSize();
14                 nextPkt ← ModifiedPkt(ts, IP1, IP1_port, IP2, IP2_port, pSize);
15                 filteredPackets.add(nextPkt);
16     return filteredPackets;
```

3.3.3 Flow creation module

The output of the packet filtering module is fed to this module to generate bidirectional flows. Each flow is identified by source IP, destination IP, source port, destination port and the transport layer protocol (TCP or UDP). The filtered packets 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 start 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 (virtual) flows, only the TIMEGAP is employed. The module is explained algorithmically in Algorithm 3.2 (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 the traffic flowing in the network. Through our experiments, we observed that a high TIMEGAP value is more suitable because many bots exchange 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 3.2: Flow creation module

```
   Data: filteredPackets
   Result: initFlowList
1  begin
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 (nextFlow.start − TIMEGAP) &&
               (nextFlow.end + TIMEGAP) then
10                 nextFlow.addPacket(p);
11             else
12                 nextFlow ← Flow(p);
13                 initFlowList.add(nextFlow);
14     return initFlowList;
```

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.

3.3.4 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 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 [Pelleg & Moore 2000]. X-means algorithm is a variant of K-means which scales better and does not require number of clusters as input.

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 an obvious 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 50.12%. The results obtained with classes-to-clusters evaluation are given in Table 3.2. 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 four P2P apps having the majority of instances, an outcome of four clusters was quite expected.

3.3.5 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

Table 3.2: Classes-to-clusters evaluation with X-means

Application	**Cluster indices**			
	0	1	2	3
Zeus	1	9,304	154	34,134
Waledac	0	1,260	0	80,518
Storm	1,202	90,236	0	3,878
Nugache	0	165	0	51,263
eMule	2,032	297,373	3,852	11,0738
uTorrent	641,909	526,861	24,295	216,226
Vuze	18,097	1,018,941	23,893	147,032
Frostwire	172,995	308,016	32,0365	88,924
Clustered Instances				
Cluster 0:	uTorrent	83,6236	20%	
Cluster 1:	Vuze	2,252,156	54%	
Cluster 2:	Frostwire	372,559	9%	
Cluster 3:	eMule	732,713	17%	

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 hours, 24 hours, etc., thus giving him visibility into the network logs as required. Such flexibility is especially valuable to identify 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 hour. This module is explained in Algorithm 3.3.

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

Algorithm 3.3: Conversation generation module

```
Data: initFlowList, FLOWGAP
Result: ConvoList
1  begin
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 (nextConvo.start − FLOWGAP) &&
           (nextConvo.end + FLOWGAP) then
9              nextConvo.addConvo(cg);
10         else
11             nextConvo ← Conversation(cg);
12             ConvoList.add(nextConvo);
13     return ConvoList;
```

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 thus the median value was found to be more suitable from our experiments.

3.4 Results and evaluation

3.4.1 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

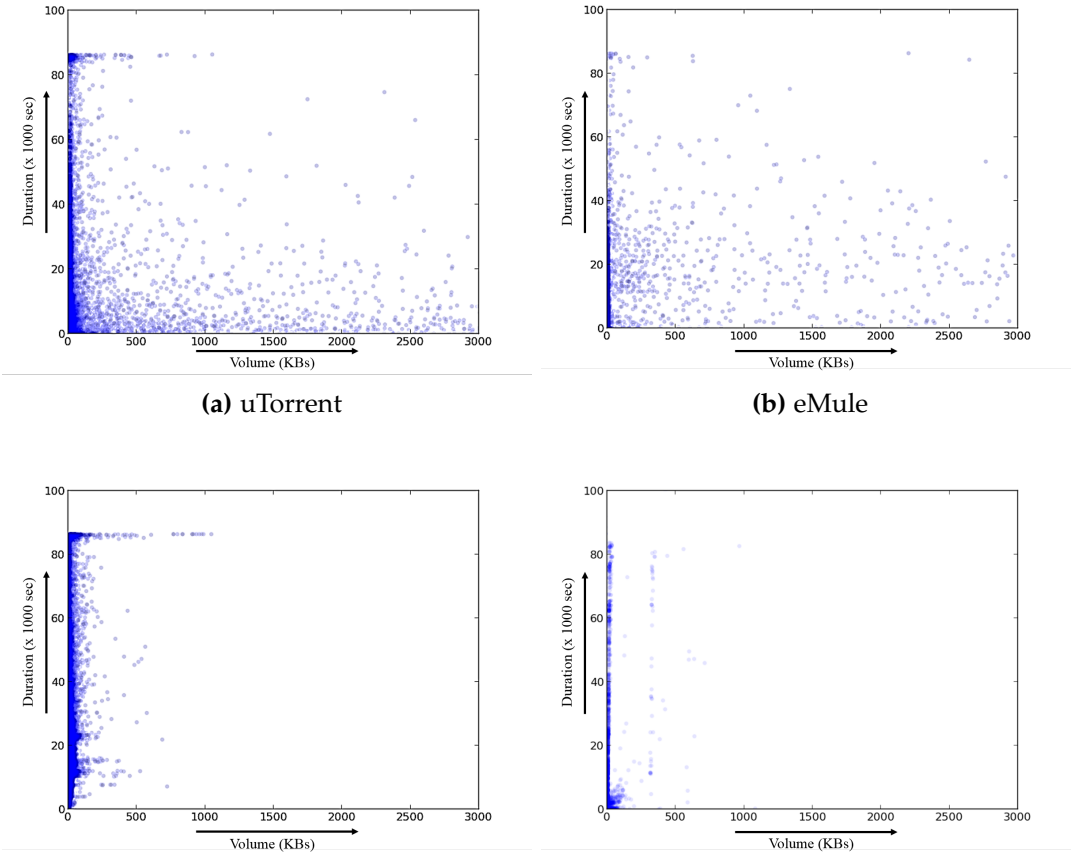
558,348 conversations (540,566 benign and 22,042 malicious). The training as well as test splits contain more than 90% benign data. Although such class imbalance makes the task of detecting P2P botnets more challenging, this ratio is representative of the real-world scenario where majority of traffic flowing in a network is benign [Cisco 2014].

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 3.1) and thus not suitable to build detection models. Nevertheless, they can be used to evaluate `PeerShark`'s capability on profiling unknown P2P bots.

3.4.2 Data visualization

Figure 3.2 visualizes a portion of the dataset. A comparison of data of uTorrent, eMule, Storm, and Zeus is given for one hour `FLOWGAP` duration in the form of scatter plots. Each point on the plot denotes a conversation, with the intensity of the color denoting higher number of conversations at that point. The X axis has the Volume (in Kilobytes), while Duration (in thousands of seconds) is plotted on the Y axis. The data displayed (for each application) corresponds to a 24 hour period. Hence 86.4 (86,400 seconds) forms the upper limit on the Y axis.

From the scatter plots, it is evident that bots tend to 'lie low' and remain stealthy. Their traffic is low-volume and high-duration, with only very few conversations being high in volume. In clear contrast, most conversations seen in benign P2P traffic (eMule and uTorrent) do not exhibit the trend of low-volume and high-



(a) uTorrent **(b)** eMule

(c) Storm **(d)** Zeus

Figure 3.2: Comparison of network traces of uTorrent, eMule, Storm and Waledac

duration. Rather the points are widely spread over the entire length and breadth of the plot. The few high-duration conversations seen in benign P2P traffic can be attributed to the fact that this is dataset was generated at the University of Georgia [Rahbarinia *et al.* 2014] by continuously running P2P applications (with sharing, downloading, etc.) over several days. Since the applications were running continuously, high durations conversations are present. Such a pattern is not expected to be seen with an average user of the Internet.

3.4.3 Classifiers

The training and testing of our models was performed using the Weka machine learning suite [Hall *et al.* 2009]. Weka provides a large number of standardized implementations for data preprocessing techniques, supervised learning algorithms, unsupervised learning algorithms, etc.

Our training and testing dataset contains a high 'class imbalance' towards the benign class. This imbalance is 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 C4.5 decision trees, which are simple to train and fast classifiers, and can handle class imbalance problems well [Drummond *et al.* 2003, Chawla 2003].

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 over-fitting of data and run efficiently on large datasets [Breiman 2001].

Along with tree-based classifiers, we use a stochastic learning algorithm—Bayesian network [Friedman *et al.* 1997]. 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. We use the Weka implementation of Bayesian network, which utilizes the K2 search algorithm for learning the network.

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 3.3.

As the results in Table 3.3 show, `PeerShark` could consistently detect P2P bots

Table 3.3: 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%

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 training and testing 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.

3.4.4 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. From the high detection accuracy presented in Table 3.4, it is evident 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

Table 3.4: Performance of classifiers on unseen P2P botnets

	Decision trees			Random torests			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%

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 [Fisher 2007]. Nugache became well known amongst analysts as a 'TCP port 8 botnet' [Stover *et al.* 2007] 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. This could possibly explain high detection rate for Nugache.

3.5 Discussion

3.5.1 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 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 malicious and benign 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 misclassify 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 [Nappa *et al.* 2010] 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 conversation 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.

3.5.2 A note on multi-class classification

In the preliminary version of our work [Narang *et al.* 2014d], 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%. This comes up to thousands of conversations in terms of the actual number. In particular, we saw many misclassified conversations between Vuze and Frostwire. Such misclassification may be attributed to the fact that 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 torrent-based applications are the only applications of P2P which continue to dominate [Sandvine 2014].

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 [Drinkwater 2014]. 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 [Rahbarinia *et al.* 2014]). 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) [Reynolds 2009]. In the flow-clustering phase discussed 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) [Moon 1996]. The flow-clustering phase with GMMs was run for number of clusters ranging from zero to eight. Lowest classification error in classes-to-clusters

evaluation was achieved for seven clusters (for a total of eight applications), with each application except Storm having a cluster where it was dominantly present (clusters of Storm and Waledac overlapped). The results of this evaluation are given in Table 3.5.

However, we observed that GMMs with EM is an *extremely slow* clustering approach. X-means outperforms GMMs by hundreds of times. Moreover, X-means has an added benefit that it does not require number of clusters as an input. Thus, we did not find GMMs with EM suitable for `PeerShark`. Since this approach is not the central component of `PeerShark`, we do not elucidate it further. But an interested reader may leverage from the ability of GMMs and EM to separate overlapping clusters.

3.6 Conclusion

In this chapter, 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 using 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.

Table 3.5: Classes-to-clusters evaluation with GMMs/EM

Application	**Cluster indices**						
	0	1	2	3	4	5	6
Zeus	5	170	0	9,155	8,346	14,423	11,494
Waledac	70	378	0	3,146	415	73,722	4,047
Storm	9,611	0	1,110	1,322	363	11,433	71,477
Nugache	3	2,417	0	11,272	6,653	14,414	16,669
eMule	56,126	726	785	275	19,871	1,11,541	2,24,671
uTorrent	1,788	1,365	639,361	9,654	22,511	212,871	521,741
Vuze	20,028	19,992	362,715	3,163	30,105	164,636	607,324
Frostwire	67,525	12,453	27,424	667	135,034	388,775	258,422

Clustered Instances			
Cluster 0:	eMule	155,156	4%
Cluster 1:	Waledac	37,501	1%
Cluster 2:	uTorrent	1,031,395	25%
Cluster 3:	Nugache	38,654	1%
Cluster 4:	Zeus	223,298	5%
Cluster 5:	Frostwire	991,815	24%
Cluster 6:	Vuze	1,715,845	41%

Chapter 4

Noise-resistant mechanisms for P2P botnet detection

4.1 Introduction

In the previous chapter (in Section 3.5), we had argued that the detection mechanism of `PeerShark` may fail if bot-masters design smarter bots which randomize the communication patterns between bot-peers. This chapter will address the context of detection of P2P bots in the presence of such randomness or noise injected by an adversary. Furthermore, `PeerShark` presented a two-tier architecture for the detection of P2P botnets. Although a two-tier architecture undoubtedly offers certain benefits, it comes at the expense of increased computation. The approach presented in this chapter directly deals with conversation-based mechanisms and enhances them by leveraging on the timing and data patterns of communication amongst bot-peers.

The context of P2P botnet detection is adversarial in nature. Statistical or behavioral models for detection are created using 'botnet' data which has been generated by an adversary. Hence, the adversary is in a position to evade the detection

mechanisms if he can change the behavior of his bots. Thus, this necessitates the evaluation of the performance of these detection models in the presence of deliberate injection of noise by an adversary. That is, if the bot-master slightly alters the behavior and communication patterns of the bots, are these detection models *robust* and *resistant* towards such a change? This question has not received sufficient attention in past research.

The contributions of this work are two-fold. First, we propose a novel approach for detection of P2P botnets in the presence of benign P2P traffic which utilizes signal-processing techniques of Discrete Fourier Transforms and information entropy. Our approach does not rely on DPI or signature-based mechanisms which are easily defeated by applications using encryption. Neither do we assume the availability of any 'seed' information of bots through honeypots or a blacklist of IPs. We aim to detect *stealthy* P2P bots solely on the basis of their 'P2P' behavior and C&C communication with other bots.

Second, we use real-world network traces of benign P2P applications and P2P botnets to compare the performance of our features with traditional flow-based features employed by past research (such as [Livadas *et al.* 2006, Saad *et al.* 2011, Kheir & Wolley 2013, Zhang *et al.* 2014]). We build detection models with multiple supervised machine learning algorithms, and demonstrate that our detection approach is more resilient towards evasive attacks wherein the bot-master alters the behavior and communication patterns of the bots by deliberate injection of randomness or noise. With our approach, we could detect P2P botnet traffic in the presence of injected noise with True Positive rate as high as 90%.

Our approach utilizes the notion of conversations as explained in the previous chapter. We extract two-tuple conversations from network traffic and treat each conversation as a time-series sequence (or a 'signal'). We leverage on the fact that communication of bots amongst each other follows a certain regularity or

periodicity with respect to timing and exchange of data. Bots tend to repeatedly exchange same kind of commands—which are often in the same format and of the same size. The repeated C&C communication also follows certain timing patterns—with bots generally contacting their fellow peers at predefined intervals [Tegeler *et al.* 2012]. In order to uncover the hidden patterns between the communications of bots, we convert the *time-domain* network communication to the *frequency-domain*. From each conversation, we extract features based on DFTs and information entropy. These features were introduced in [Narang *et al.* 2014b].

Fourier transforms can effectively extract the contributing ingredients of any signal. Most of the energy of any time series is contained in the top peaks of its Fourier transform (for details, refer to Chapter 2 (Fourier series representation) of [Oppenheim *et al.* 1997]). By modeling a network communication in the form of a signal and performing a Fourier transform over it, we aim to extract the *highest contributing* ingredients of a signal or the top peaks of the Fourier transform. We focus our attention on these peaks and utilize them as 'features' to build our detection models. Since our detection models rely only on these 'top peaks' of the communication, they are more resilient towards variation in the data or injection of noise.

Information entropy, as introduced by Shannon, defines the amount of information contained in an event. The concept has found extensive use in data compression techniques. We use this measure to quantify the *randomness* of the data patterns in network communications (modeled as a time-series sequence). Since bots usually run automated scripts and commands, their network behavior often displays more periodicity as compared to a human user of the Internet. That is, communication of bots often exhibits *less randomness* [Tegeler *et al.* 2012]. We harness this factor as a feature in our detection models.

The contents of this chapter are organized as follows: an overview of our approach

is described in Section 4.2, and implementation details are given in Section 4.3. We evaluate our system in Section 4.4. Section 4.5 presents a discussion on the results obtained with our approach. Finally, we conclude this chapter in Section 4.6.

4.2 System Overview

P2P botnets form structured P2P topologies using custom as well as well-known P2P protocols. As a result, such traffic can easily *blend in* with legitimate P2P traffic coming from applications such as BitTorrent, Gnutella, eMule, etc., and thus pass under the radars of IDS/Firewalls monitoring a network. We attempt to differentiate P2P bot traffic from P2P applications by leveraging on the timing and data patterns of communication amongst bot-peers. The C&C communication between bots often follows certain regular patterns of timing and exchange of data. We harness this periodicity by converting the *time-domain* network communication to the *frequency-domain*. Our approach makes use of two-tuple conversations instead of the traditional approach of five-tuple flows. In particular, we extract 'conversations' for each pair of hosts occurring in network traffic. We model each conversation as a time-series sequence and extract features based on DFTs and information entropy.

Fourier transforms provide the frequency domain representation of any signal. We use them to capture the patterns in timings and exchange of data in C&C communication of bot-peers. A conversation between two hosts is modeled in the form of a time-series sequence. The inter-arrival time of packets and payload size of the packets are taken as two series of discrete symbols. Fourier transform is applied over these two series. We focus our attention on the top peaks of the Fourier transform and utilize them for building detection models.

Bots often engage in repeated exchange of similar commands, which often occur

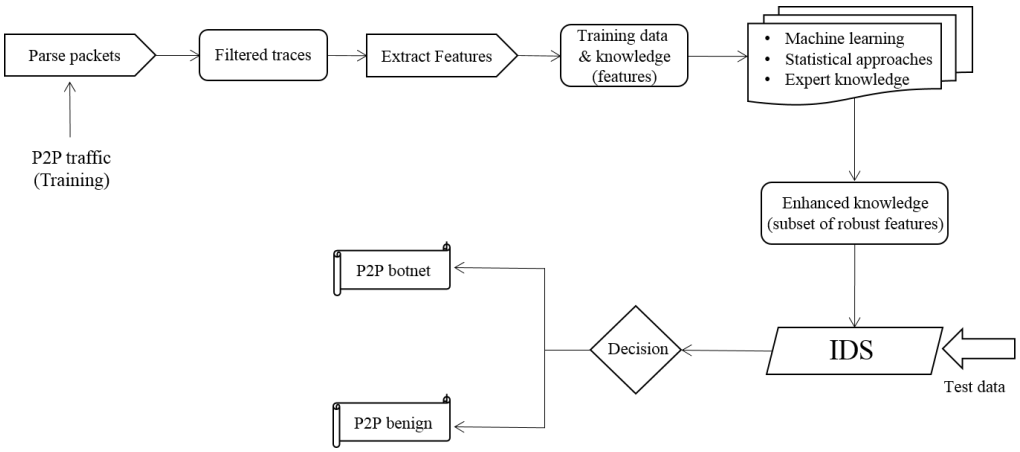


Figure 4.1: Design of the IDS used in our module

in a similar format and thus packets of similar sizes. This is to say that *conversations* among bots often have packets of similar sizes, and thus exhibit less *randomness*. This is quite expected since bots are designed to run simple and structurally repetitive tasks. Conversations created from benign applications, on the other hand, are not expected to display such behavior since different users of Internet may use different applications and may display different usage patterns. We quantify this randomness or entropy in payload variation by modeling payload size per packet as a discrete symbol, and calculate the 'compression ratio' using information entropy (more details will follow in next section).

The data used in this work corresponds to real-world traces obtained from [Rahbarinia *et al.* 2014]. We extract the above-mentioned features from the network traces of P2P bots and P2P applications. These features are used to build P2P bot detection models using multiple supervised machine learning algorithms.

4.3 System Implementation

Figure 4.1 gives the design of the IDS used in this module. The IDS design combines various principles from machine learning, statistical approaches (us-

ing Fourier transforms and information entropy) and expert knowledge (of P2P bot behavior). We will discuss each of these in detail as we describe each module of our system.

4.3.1 Packet parsing module

Our system operates on a dump of network traces (`pcap` files) as the input. The packet parsing module uses `libpcap` library to parse `pcap` files and extract features of relevance. We keep only those packets which have a valid TCP/UDP header. Corrupted packets or packets which have the header information missing are discarded. Background traffic in the form of IP Broadcasts, ARP requests, etc. is filtered out at this stage. For each packet, we extract its timestamp, source IP, destination IP, and payload length at the transport layer (for TCP or UDP, as applicable). This information is used by the next modules to generate conversations and extract all the required features.

Algorithm 4.1: Packet parsing module

```
   Data: packetCapture
   Result: filteredPackets
1  begin
2      ArrayList < ModifiedPkt > filteredPackets;
3      for Packet pkt in packetCapture do
4          ts ← pkt.getTimestamp();
5          if pkt has IPHeader then
6              ip ← pkt.getIPHeader();
7              IP1 ← ip.getSourceIP();
8              IP2 ← ip.getDestIP();
9              if pkt has TCPheader or UDPheader then
10                 header ← pkt.getTransportHeader();
11                 pSize ← header.getPayloadSize();
12                 nextPkt ← ModifiedPkt(ts, IP1, IP2, pSize);
13                 filteredPackets.add(nextPkt);
14     return filteredPackets;
```

The Packet parsing module is very similar to Algorithm 3.1 explained in Chapter

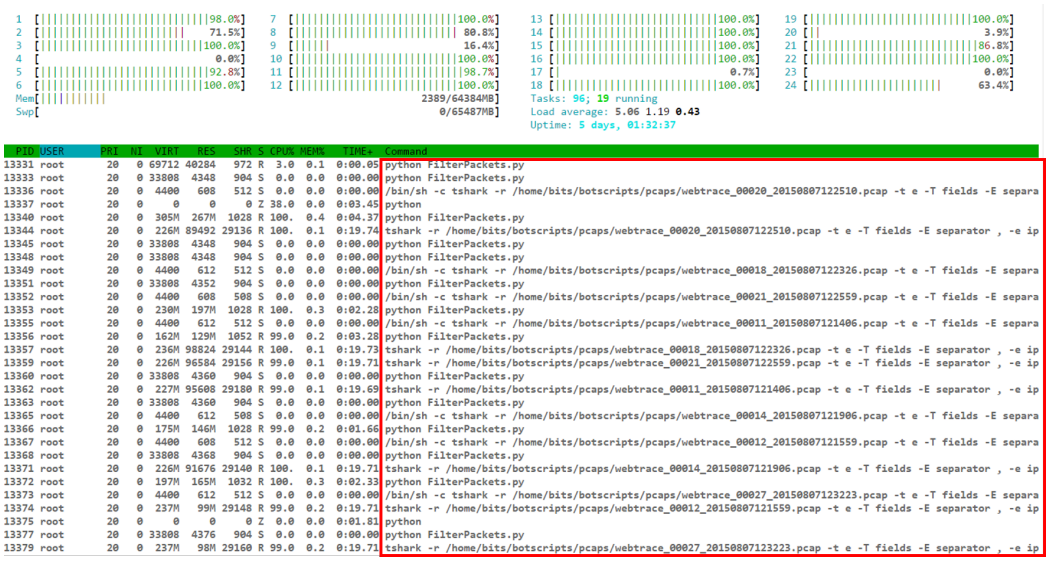


Figure 4.2: A snapshot of the run of packet parsing module

3. But, for the sake of completeness, we explain it again in Algorithm 4.1.

The Packet parsing module, implemented in Python as `FilterPackets.py`, internally invokes a `libpcap` library based network protocol analyzer `Tshark`. `Tshark` is used to parse the `.pcap` files as explained above. `FilterPackets.py` is invoked in parallel on a set of `.pcap` files. Parallel execution is achieved by using GNU Parallel[1]. Figure 4.2 shows a snapshot from `htop`[2], and the portion marked in red indicates parallel execution of `FilterPackets.py`.

4.3.2 Conversation creation & feature extraction modules

4.3.2.1 Conversation creation module

The output of the Packet parsing module (*filteredPackets*) is fed to this module to create conversations. Conversations are created by aggregating packet-level data. Each Conversation is identified by `<IP1,IP2>` and a `TIMEGAP` parameter. `TIMEGAP` is defined as the maximum permissible inter-arrival time between two

[1]http://www.gnu.org/software/parallel/
[2]http://linux.die.net/man/1/htop

packets in a conversation.

Whenever a new packet p is seen, it is added to an existing conversation or allotted a new conversation based on the following criteria:

1. If the IP pair of p does not belong to any existing conversation, a new conversation is created for the IP pair of p.
2. If the IP pair of p belongs to an existing conversation, we check the timestamp of p.
   (a) If timestamp of p lies within the `TIMEGAP` range from the end of the existing conversation, p will be added to that conversation.
   (b) Otherwise, a new conversation is created for the IP pair of p.

`TIMEGAP` is a 'tunable' parameter, which is to be chosen by a network administrator based on his understanding of the type of traffic flowing in the network. From our experiments, we observed that a higher value of `TIMEGAP` is more suitable for bots since many bots exchanged only a few packets after long gaps. A low `TIMEGAP` value would result in 'timeouts', and imply just one or two packets per conversation. This will not be useful to extract any statistical metrics from conversations. We use a `TIMEGAP` of 2,000 seconds.

The Conversation creation module is similar to the Conversation generation module (Algorithm 3.3) explained in Chapter 3. However, this module creates conversations directly from packet data, while Algorithm 3.3 utilized flow data to create conversations. We present this module in Algorithm 4.2. The feature extraction module is used along with this module to extract relevant the features from every conversation.

A snapshot from the implementation of this module is given in Figure 4.3. The module, implemented in Python as `convo.py`, is invoked in parallel using GNU Parallel. The parallelized implementation allows each `convo.py` script to process

Algorithm 4.2: Conversation creation module

```
Data: filteredPackets
Result: Conversation list
1  begin
2      ArrayList < Conversation > initConvList;
3      ArrayList < PacketGroup > pgList;
4      pgList ← filteredPackets.groupPktsByIPpair();
5      for PacketGroup pg in pgList do
6          sort packets in pg by ts;
7          nextConv ← Conversation(NULL);
8          for Packet pkt in pg do
9              if pkt.timestamp between (nextConv.start – TIMEGAP) &&
               (nextConv.end + TIMEGAP) then
10                 nextConv.addPacket(pkt);
11             else
12                 nextConv ← Conversation(pkt);
13                 initConvList.add(nextConv);
14     return initConvList;
```

the parsed `.pcap` files in parallel. Figure 4.3 shows a snapshot from `htop`, and the portion marked in red indicates parallel execution of `convo.py`.

4.3.2.2 Feature extraction module

The Feature extraction module extracts features based on DFTs and information entropy for every conversation. In any time-series, most of the energy is contained in the top peaks of the Fourier transform. Hence, it may be argued that it should be sufficient to focus our attention only on the top peaks itself. We harness this characteristic by considering payload length (in Kbs) per packet in a conversation and the inter-arrival time of packets in a conversation as two series of discrete symbols. We apply a Fast Fourier Transform algorithm to both these series, and focus our attention on the 'top peaks' obtained from the Fourier transform.

On the sequence of payload lengths, we also compute the 'compression ratio' by using information entropy based on Shannon's source coding theorem. The source

Figure 4.3: A snapshot of the run of conversation creation module

coding theorem establishes the limits of data compression. We use information entropy to calculate the 'compression ratio' achieved over the sequence of discrete symbols obtained by modeling each conversation as a time-series sequence. As noted before, it is expected that communication of bots will have more 'structure' and 'regularity' associated with it. The data patterns generated by bots will have less randomness. Thus, higher 'compression ratio' is expected for communication of bots.

Fourier transform: Discrete Fourier Transforms (DFT) uncover the periodicities in input data, and also bring out the strength of these periodic components. Given a vector of N input amplitudes, $x(0), x(1) \ldots x(N-1)$, the DFT yields a set of N frequency magnitudes.

$$X[k] = \sum_{n=0}^{N-1} x(n) \cdot e^{\frac{-2j\pi kn}{N}} \tag{4.1}$$

In the equation above, each $X[k]$ is a complex number which encodes both the amplitude and phase of sinusoidal component of function $x(n)$. Here, N is the length of the sequence to be transformed, k is used to denote the frequency domain ordinal, and n is used to represent the time-domain ordinal. The sinusoidal frequency is k cycles per N samples.

We compute Fourier transform over payload length (in Kbs) per packet and the inter-arrival time of packets in a conversation. For each Fourier transform thus obtained, we sort its values in descending order of magnitude. The top ten magnitude values are retained. The 'phase' component of a Fourier transform has no meaningful interpretation in this context, and is dropped. We also extract the vector sum of the top ten values in the form of the 'prime wave'. We obtain 11 magnitude values corresponding to features of inter-arrival time as well as payload lengths. In total, we extract 22 features based on Fourier transform.

Our features based on Fourier transform rely only on the top peaks of communication obtained from a conversation. Slight variation in the timing or data patterns of the conversation will not hamper the top peaks. As a result, our approach is expected to be more resilient towards variation in the data or injection of noise. We test this proposition in Section 4.4.4 by injecting noise in our test data, and present the results of our evaluation in Section 4.5.2.

Entropy: In order to maintain connectivity with each other, bot-peers engage in regular exchange of C&C messages. It has been noted [Tegeler *et al.* 2012] that such communication often occurs in the same format because bots are usually hard-coded to perform simple and structurally repetitive tasks. Indeed, we noticed in the network traces of Zeus (obtained from [Rahbarinia *et al.* 2014]) that the bots periodically engaged in such communication with small, fixed-size payloads, and often switched between TCP and UDP ports. The payload sizes of benign Internet

traffic will exhibit much more variation. This can be attributed to the fact that human users of the Internet differ in their usage patterns and use a variety of applications, each having a different behavior with regard to its payload sizes. Thus, we do not expect to see any such trend(s) in the payload sizes. Benign conversations are expected to have more randomness or entropy, which will lead to less compression, while bot conversations are expected to have low entropy leading to a higher compression.

From Shannon's source coding theorem [MacKay 2003], we can say that N outcomes from a source X can be compressed into $N \times H(X)$ bits with negligible risk of information loss, where $H(X)$ is entropy. In order to quantify the amount of entropy or randomness in payload variation, we model the pair of IPs in a conversation as the source(s), and payload size per packet as the discrete symbol being generated.

In a conversation, let us consider that x_i are the payload sizes generated by the source(s) X, which are modeled as discrete symbols. However, if an adversary adds slight randomness in the values of x_i, the bot communication will no longer occur in packets of similar size/format, and the entropy will shoot up. To account for this slight variation, we round-off each payload value to its nearest tens. Hence, if payload sizes in a conversation are $259, 263, 271, 277, 278, 267, 261$, they will be rounded-off to nearest tens, and the resulting sequence will be: $260, 260, 270, 280, 280, 270, 260$.

For this modified sequence, entropy $H(X)$ will given by:

$$H(X) = \quad -\sum_{x_i \epsilon X} p(x_i) \cdot \log_2(p(x_i)) \tag{4.2}$$

Here $p(x_i)$ is the probability of occurrence of x_i. For example, for the sequence given above with 7 symbols, the $p(x_i)$ value for 260 is $3/7$.

We need $H(X)$ bits per symbol to encode any sequence without loss of information. If N is the total number of packets in the conversation and U is the count of unique payload sizes in the conversation, the compressed size of this sequence can be given by:

$$compressed\_size = N \times H(X) + U \tag{4.3}$$

The count of unique payload sizes is added to the 'compressed size' to account for the number of symbols used in the encoding.

To calculate the actual/unencoded size of this sequence, we consider that each discrete symbol (i.e., the payload size per packet) requires 16 bits to represent. With 16 bits, the highest value of the payload which can be represented is 65,535 bytes, which is the maximum theoretical limit for transport layer payloads, and far greater than what is usually seen in Ethernet-dedicated Internet of today. However, the reader is requested to note that the value of 16 is used only for modeling and computing the unencoded size of the sequence.

Thus, unencoded length required to represent a conversation with N packets will be given by $16 \times N$.

The 'compression ratio' will be calculated as the ratio of the actual size to the compressed size.

$$compression\_ratio = \frac{uncompressed\_size}{compressed\_size} = \frac{16 \times N}{N \times H(X) + U} \tag{4.4}$$

This gives us one more feature of compression ratio. This feature is based on the inherent behavior of bots repeatedly performing structurally repetitive tasks. The 'entropy' of the network behavior will not vary significantly even if slight variation/noise is introduced in the communication patterns of a bot. In our approach, we further round-off the payload values (to nearest tens), and thus further reduce the impact of noise injected by an adversary. Hence, 'compression

ratio' is expected to be tolerant towards introduction of noise by an adversary.

In total, we extracted 23 features from every conversation. It might be noted that, at this stage, we justified the choice of all these features primarily based on the behavioral understanding of P2P botnets [Tegeler *et al.* 2012]. In Section 4.4.1, we will evaluate the suitability of these 23 features using feature selection algorithms.

4.3.3 Firewall module

The output from our IDS module is in the form of IPs which are P2P applications or IPs infected by a P2P bot. These results can be of practical benefit to a firewall module which can reject or limit P2P traffic as desired. We use these results for a 'firewall' module to generate a dynamic rule-set governing P2P traffic.

An Iptables based firewall was setup on a Gateway machine of our lab environment. The 'firewall' module utilizes Iptables and bash scripts to create automated firewall rules for hosts identified as running P2P apps and for hosts infected with bots. For the prototype implementation of our module, we implemented firewall rules for top 20 IPs outside our University which were identified as P2P, top 20 IPs outside our University which were identified as bot-infected, and top 20 IPs inside our University which were identified as bot-infected. Apart from creating firewall rules pertaining to these set of hosts, we also created generic firewall rules to rate-limit connections of popular P2P applications. Our firewall rules have a dynamic nature. We do not block/reject P2P traffic on weekends. On working days, P2P traffic is disallowed during the working hours (9:00 AM – 5:00 PM). Apart from working hours, P2P traffic is allowed. Bot traffic, however, is shown zero tolerance and is rejected under all circumstances.

Given below are the rules written using Iptables[3].

[3]http://linux.die.net/man/8/iptables

All traffic to the top 20 bot IPs (as detected by our module) outside the BITS network is outrightly rejected so that any attempt from bot-infected hosts inside the BITS network attempting to connect to them is denied:

```
iptables -I FORWARD -t filter -d $ip -j REJECT
```

Further, the traffic from top 20 bot-infected hosts inside the BITS network is limited to 3 connections per host:

```
iptables -I FORWARD -s $ip -m connlimit --connlimit-above 3
-j REJECT
```

All P2P traffic is permitted during weekends, and blocked during the working hours of weekdays. P2P traffic is blocked by rejecting all traffic on known standard ports of popular P2P applications. After working hours, this traffic is again permitted. Iptables processes rules in the order in which they are written. So, we add this rule after the rules meant for bot traffic because we want to continue blocking bot traffic irrespective of day and time. This rule is given here:

```
iptables -I FORWARD -p tcp -m time --timestart 09:00
--timestop 17:00 --weekdays Mon,Tue,Wed,Thu,Fri -m multiport
--sports 411,412,2323,6347,1214,6346,6881,6889,6699 -j REJECT
```

Certain P2P applications may escape this rule by using ports other than those mentioned above. For this purpose, we reject all traffic to the top 20 P2P IPs during the working hours of weekdays:

```
iptables -I FORWARD -t filter -d $ip -m time --timestart 09:00
--timestop 17:00 --weekdays Mon,Tue,Wed,Thu,Fri -j REJECT
```

The rules given above are explained in Algorithm 4.3.

Algorithm 4.3: Iptables pseudo-code

```
Weekday ← [Mon, Tue, Wed, Thu, Fri];
P2P_ports ← [411, 412, 2323, 6347, 1214, 6346, 6881, 6889, 6699];
begin
    while do
        if IP in Top20_Bot_IPs then
            REJECT traffic to IP;
        else if IP in Top20_local_Bot_IPs then
            Rate_limit traffic to IP to 3 connections;
        else if Time in [9 : 00 − 17 : 00] and Weekday then
            REJECT TCP traffic on P2P_ports;
        else if IP in Top20_P2P_IPs then
            if Timein [9 : 00 − 17 : 00] and Weekday then
                REJECT traffic to IP;
        else
            ALLOW traffic;
```

4.4 System Evaluation

This work uses data of benign P2P applications and P2P bots obtained from [Rahbarinia *et al.* 2014]. The dataset contains network traces of four benign P2P applications, namely uTorrent, eMule, Vuze and Frostwire, and three P2P bots, namely Storm, Waledac and Zeus. The traces of P2P bots correspond to real-world traces obtained by [Rahbarinia *et al.* 2014] from third parties. The network traces of P2P applications were generated by them 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. Altogether, we used three bots and four apps for this work.

The network traces of all applications were parsed using the modules described in the previous section. Two-tuple conversations and accompanying features were extracted from each application. The conversations thus obtained were labeled to create a 'labeled dataset', which will be used for training and testing purposes.

For all conversations corresponding to P2P applications, we use the label 'benign'.

The network traces of each of the P2P bots contain certain 'known malicious hosts'. These hosts are confirmed hosts which are infected by the bot. Zeus had 1, Waledac had 3 and Storm had 13 'known malicious hosts'. However, it is not known whether the other IP addresses seen in the network traces are benign or malicious[4]. For the evaluation of our system, we require a labeled dataset which can be treated as the 'ground truth'. Thus, to create a labeled dataset from the network traces of P2P bots, we treat a conversation as 'malicious' if either of the IPs (source or destination) is known to be 'malicious'. If none of the IPs in a conversation are known to be malicious, it is treated as benign.

4.4.1 Feature selection

Extraction of features based on Fourier transform and entropy was motivated based on behavioral understanding of P2P botnets. We extracted 23 features – 22 features based on Fourier transform, and one feature of compression ratio. Before we begin to create detection models using them, we must test the suitability of these features for this task. To this end, we turn our attention to popular feature selection algorithms.

Feature selection is a process of selecting a subset of relevant features from the entire set, with aim of removing those features which do not significantly contribute towards the classification problem, or are totally irrelevant to the classification problem. Presence of irrelevant features can lead to construction of poor detection models which do not generalize to newer data. Since feature selection results in lesser number of features being used for the classification task, it also leads to higher classification speed.

[4]Personal communication with Babak Rahbarinia [Rahbarinia *et al.* 2013] in November 2013

Specifically in our context, we had argued that we are selecting the magnitude of the top ten peaks of the Fourier transform (as well as their vector sum) since most of the energy of any time series is contained in the top peaks of its Fourier transform. But, it is very much possible that certain peaks of the Fourier transform do not provide any further information for the classification problem than what is already provided by other peaks. Also, we need to evaluate whether the feature of compression ratio provides valuable information for the classification problem of detection of P2P bots. Feature selection will help us accomplish this task.

We use two well-known feature selection algorithms for this purpose – Correlation-based Feature Selection (CFS) algorithm [Hall 1999] and Consistency-based Subset Evaluation (CSE) search algorithm [Dash & Liu 2003]. An overview of these algorithms has already been presented in Chapter 1.

With the labeled datasets P2P bots and benign applications, we used CFS and CSE algorithms with stratified ten-fold cross-validation. We select only those features in the final subset which were chosen by CFS *as well as* CSE in all the ten runs of cross-validation. The final subset of features is given in Table 4.1 ('Fourier Transform' is abbreviated as 'FT', and 'inter-arrival time' as 'iat').

For features based on Fourier transform, we observe that the magnitude of first three peaks and the prime-wave contributed highly towards the classification, and were thus chosen by both the algorithms. The feature of 'compression ratio' was also selected by both the algorithms. Henceforth, we proceed with these nine features in our dataset and discard other features.

4.4.2 Dataset creation

The 'labeled' conversations of all four benign P2P applications along with Storm, Waledac and Zeus were used for training our detection models and testing them.

Table 4.1: Features selected after 'feature selection' using CFS and CSE

FT (payload) – prime-wave (magnitude)
FT (payload) – peak 1 (magnitude)
FT (payload) – peak 2 (magnitude)
FT (payload) – peak 3 (magnitude)
FT (iat) – prime-wave (magnitude)
FT (iat) – peak 1 (magnitude)
FT (iat) – peak 2 (magnitude)
FT (iat) – peak 3 (magnitude)
compression ratio

The details are given in Table 4.2. `TR` is the training dataset which contains conversations of all four benign P2P applications along with conversations of Storm, Zeus and Waledac, and is used to build and train detection models with multiple supervised machine learning algorithms. `TE` is the test dataset which is used for evaluation of the trained models. It contains conversations (distinct from `TR`) of all benign P2P applications and P2P bots. Further, we wish to evaluate the resilience of our features towards noise injected in the data by an adversary. For this purpose, we create the dataset `TN` from `TE` by adding noise to the conversations of bots. The process of injecting noise will be explained in Section 4.4.4.

4.4.3 Extracting flow-based features

In this work, we have used two-tuple conversation-based approach. Most of the prior approaches on the detection of P2P bots, on the other hand, employ traditional five-tuple flow-based approaches. In order to compare our approach with past research, we need a comparison with flow-based approaches, and require to extract the features used by previous researchers. Hence, we also perform the entire feature extraction procedure using five-tuple flows.

Just as we extracted two-tuple conversations from the network traces of P2P bots

Table 4.2: Dataset details

Dataset	Type	No. of conversations	Description
TR	benign	189,100	Training data
	malicious	91,620	
TE	benign	94,530	Testing data
	malicious	45,700	
TN	benign	94,530	Testing data
	malicious	45,700	injected with noise

and benign applications, we extracted five-tuple flows using similar modules given in Section 4.3 for the entire dataset. Each flow is identified by source IP, destination IP, source port, destination port, and transport layer protocol, as explained before. Flows are created based on the five-tuple and a TIMEGAP value. We use the same TIMEGAP value as was used in the conversation creation module. Here, TIMEGAP is defined as the maximum permissible inter-arrival time between two consecutive packets in a *flow*, beyond which the later packet is marked as the beginning of a new flow. For TCP, a new flow is initiated only after the regular TCP three-way handshake has been established. The termination criteria is met either by TIMEGAP or the TCP connection termination sequence (in terms of FIN packets or RST packets), whichever is encountered first. For the termination of UDP (virtual) flows, only the TIMEGAP can be considered (since UDP does not have any specific criteria for flow termination).

For each flow, statistical features given in Table 4.3 were extracted. Majority of these features have been taken from past research on detection of P2P bots. For example: 'Protocol' has been used as a feature by almost all past research using flow-based approaches; 'First packet size' was used by [Saad *et al.* 2011]; 'Flow duration', 'Total payload exchanged', 'Total packets exchanged' etc. were used by [Livadas *et al.* 2006]; Number of packets sent/received and number of bytes sent/received were used by [Zhang *et al.* 2014] and [Kheir *et al.* 2014]; Bytes per second sent/received were employed by [Kheir & Wolley 2013]; etc.

Table 4.3: Traditional flow-based statistical features

Protocol	Flow Duration
First Packet Size	Max Packet Size
Payload Sent	Payload Received
Total Payload exchanged	Bytes Per Sec
Bytes Sent Per Sec	Bytes Received Per Sec
Median iat (Sent)	Median iat (Received)
Median iat	Avg iat
Avg iat (Sent)	Avg iat (Received)
Packets Sent	Packets Received
Total Packets Exchanged	Packets Per Second
Packets Per Second Sent	Packets Per Second Received
Avg Payload Size (Sent)	Avg Payload Size (Received)
Avg Payload Size	Variance Of Packet Size
Variance of Packet Size (Sent)	Variance of Packet Size (Received)

Two-tuple conversations, by definition, are port and protocol-oblivious. They may combine several flows between a pair of IPs into a single *conversation*. Thus, the number of flows obtained from the same network trace are a lot more than the number of conversations. Hence, for each P2P application, we sample out number of flows equal to the number of conversations. Using these flows, we create the datasets `TR`, `TE` and `TN`, as we did for conversations. The number of flows in each dataset are kept same as the number of conversations. This was done to facilitate an easy and intuitive comparison between both the approaches (which will be presented in Section 4.5).

4.4.4 Noise injection

The problem of detection of P2P bots is adversarial in nature. If a detection model relies on observing the network behavior of bots, the adversary (in this case, the bot creator) can try to defeat the detection by adding some randomness to the network behavior of his bots. Thus, it becomes important to evaluate whether a suggested approach for P2P bot detection can withstand the injection of noise in

the data. To this end, we inject noise in our testing data and create the dataset `TN`.

For the purpose of this work, we utilize a single attack model. The 'range' of noise is chosen to be between 25% and 33% of the original value. We will discuss more on this choice in Section 4.5.3.

The first level of noise injection is applied at the packet-level, and hence affects both flows and conversations. The noise injection module is explained in Algorithm 4.4. The input to this algorithm is the list of filtered packets ($filteredPackets$) belonging to a five-tuple (in case of flows) or a two-tuple (in case of conversations). Each *packet* carries its payload value and the inter-arrival time (abbreviated as 'iat') of this packet and the packet previous to it.

The variable *var* is randomly assigned the value 0, 1 or -1. The variable *noise* is randomly assigned a value between 0.25 and 0.33 for each *packet*, and indicates the percentage of noise injected. If *var* is one, this value will be added to the original value of payload and inter-arrival time of each *packet*. If *var* is -1, this value is subtracted. Nothing is done if *var* is 0. This implies that the adversary, at random, *adds* noise in one-third of the cases, *subtracts* it in one-third of the cases, and does nothing at all in the remaining one-third.

Further, the second level of noise is applied at the 'flow' level. The transport layer protocols are swapped for a flow if *var* is 1. This is done to take into account the nature of bots which switch between TCP and UDP. However, conversations do not consider the protocol in their definition, and will not be affected by this.

Algorithm 4.4: Noise injection module

```
Data: filteredPackets
begin
    for packet in filteredPackets do
        var ← random.choice(0, 1, −1);
        noise ← random_number(25, 33)/100;
        packet.iat ← packet.iat + var · packet.iat · noise;
        packet.payload ← packet.payload + var · packet.payload · noise;
    for flow in FlowList do
        var ← random.choice(0, 1, −1);
        if var is 1 then
            swap_protocol();    /* not applicable to conversations */
```

4.5 Results & Discussion

4.5.1 Classifiers

The training and testing of our models was performed using the Weka machine learning suite [Hall *et al.* 2009].

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 (where malicious traffic is always in minority). Hence, we need to utilize learning algorithms which can handle class imbalance problems well.

We train multiple classifiers for this work which have been employed in prior research [Saad *et al.* 2011, Livadas *et al.* 2006, Kheir & Wolley 2013, Kheir *et al.* 2014, Singh *et al.* 2014].

Decision trees are simple to train and fast classifiers, and can handle class imbalance well. However, they are notorious for creating complex structures and over-fitting the data. Instead of using regular 'C4.5 decision trees', we used 'reduced error-pruning' (REP) trees. The REP tree implementation of Weka allows

us to limit the maximum depth of the tree. By limiting the maximum depth, we can avoid complex structures in the tree. Limiting the depth may lower the training accuracy, but allows us to obtain a simpler model (as compared to C4.5) which is less prone to over-fitting and generalizes better. After experimenting with different values for the maximum depth, we chose the value of 10 as it gave highest training accuracy (over datasets of conversations as well as flows).

Next, 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 over-fitting of data and run efficiently on large datasets. We present our results with Random forest of 10 trees in this section.

Along with tree-based classifiers, we use a stochastic learning algorithm—Bayesian network [Bouckaert 2008]. 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. We use the Weka implementation of Bayesian network, which utilizes the K2 search algorithm for learning the network.

We also use a Naïve Bayes and Decision tree hybrid classifier, namely Naïve Bayes tree (NB tree) [Kohavi 1996]. Furthermore, we use the *K* nearest neighbors algorithm because of its inherent simplicity which does not over-fit the training data. *K*-nn was used with number of neighbors equal to 5.

Lastly, we use the 'stacking' ensemble learning technique [Wolpert 1992] (also known as 'stacked generalization'). The 'stacking' method uses several classification algorithms (known as 'base' classifiers) together in the first step, and then combines their predictions in the second step using a 'meta' classifier. This is pictorially represented in Figure 4.4. We use Bayesian networks, NB tree and

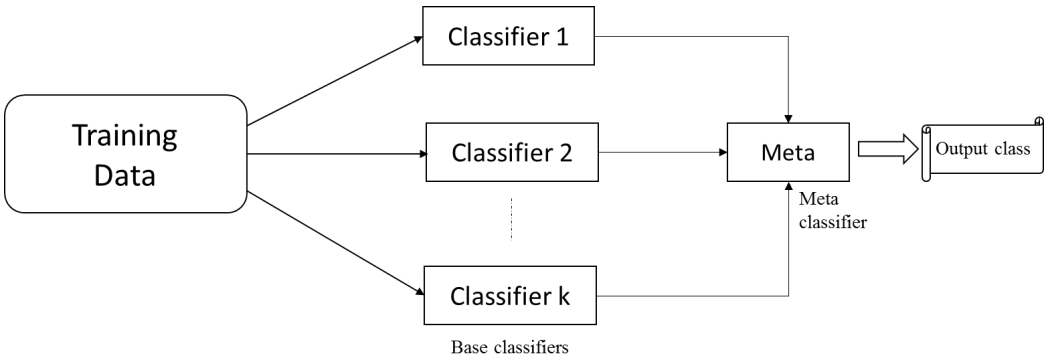


Figure 4.4: 'Stacking' ensemble learning

REP trees as the base classifiers, and use Naïve Bayes as the meta classifier. By combining several algorithms, we hope to achieve a robust prediction.

These classifiers were used to build detection models using the training dataset `TR` of conversations as well as flows. Ten-fold stratified cross-validation was used to build the models. The models were tested with the test dataset `TE` and the test data injected with noise (`TN`), for conversations as well as flows, in each case. These results are presented in Figure 4.5. The 'True Positive (TP) rate' (or the 'recall') for the 'bot' class is presented for each dataset. Each sub-figure gives, on the left, the TP rate obtained over all four datasets with the new features presented in this work, and the TP rate over all four datasets with traditional 'flow-based' features to the right.

The `TR`, `TE` and `TN` datasets also had benign P2P data in them. Since noise injection was performed only for bot data, `TE` and `TN` are the same in case of benign data. All classifiers performed exceedingly well in detection of benign traffic—with TP rate for `TR` around 98–99%, and TP rate for `TE` (or `TN`) around 95–99%. We skip these results from Figure 4.5 to facilitate easy viewing of TP rate of bot data over the different datasets.

4.5.2 Results

Table 4.2 mentions the datasets that were used in this work. Herein we will discuss the results obtained over each dataset.

4.5.2.1 Training and testing

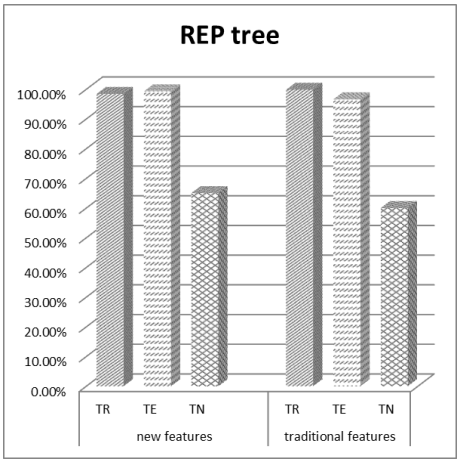
All the classifiers were trained over `TR` and tested over `TE`. The results in Figure 4.5 clearly show that the approach proposed by us performs as good as the traditional features in most cases, and outperforms the traditional features in some cases. The TP rate over `TR` and `TE` were consistently above 98%. Although high TP rate was observed even with traditional features with most classifiers, over the dataset `TE` it fell down to less than 90% with Random forests, and to 80% with *K*-nn.

The training and testing datasets establish that our approach fares at least as good as traditional approaches for the detection of P2P bot traffic. We will further evaluate both the category of features for their capability in profiling botnet data injected with noise.

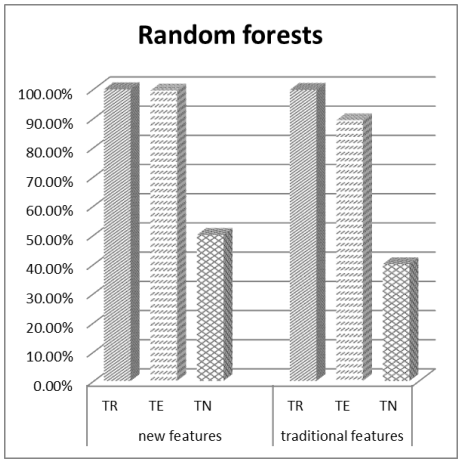
4.5.2.2 Testing with injected noise

As described before, `TE` was injected with noise to create the dataset `TN`. Results over `TN` will give better indication that whether a detection approach is resilient towards variation or noise introduced in the data by an adversary.

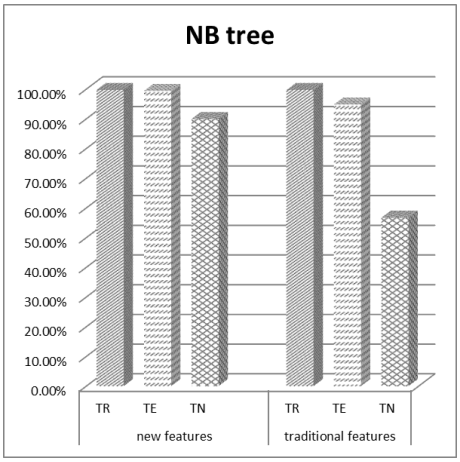
We consider the performance of tree-based classifiers first. We observe that TP rate for REP trees with our approach is only 65%. Although the values are not high, REP trees fared nearly as good with our features as with traditional features—the TP rate for traditional features being 60%. However, Random forests do not perform well with either of the features. The TP rate of 50% over new features is



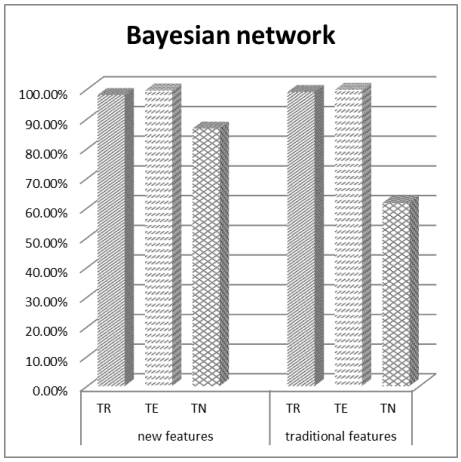
(a) REP trees



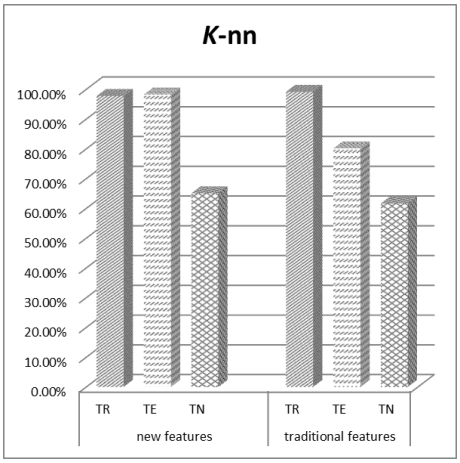
(b) Random forests



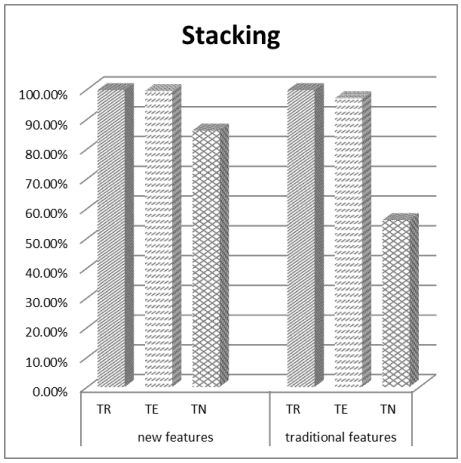
(c) NB tree



(d) Bayesian network



(e) K-nn ($K = 5$)



(f) Stacking

Figure 4.5: TP rate for P2P bot detection, obtained with different machine learning algorithms over new features and traditional 'flow-based' features

better than 40% over traditional features. However, neither of them are valuable since such a low TP rate is hardly better than a random or naïve guess.

NB tree performs exceptionally well with our features—with TP rate of 90%. On the other hand, the TP rate with traditional features is only 56%. A similar trend is seen with Bayesian networks too. The TP rate with our features is 87%, while with traditional features it is 61%. With *K*-nn, our features fare better only by a small margin.

As expected, combination of several algorithms through stacking presents good classification results. With new features, the TP rate is 88%. However, the corresponding value for traditional features stands at a mere 55%.

4.5.3 Discussion

The performance obtained with tree-based classifiers draws a clear observation that they do not perform well in the presence of variation or noise injected in testing data. This observation held true with traditional features as well as our features—albeit our features performed at least as good as traditional features.

This observation regarding tree-based classifiers is interesting since a lot of past research has used and emphasized on tree-based classifiers and flow-based approaches for Internet traffic classification [Williams *et al.* 2006], P2P traffic classification [Li *et al.* 2008, Zhang *et al.* 2010] or detection of P2P bots [Rahbarinia *et al.* 2014, Singh *et al.* 2014]. Our experiments clearly establish that, although tree-based classifiers give high accuracy in a controlled environment (over `TR` and `TE`), they fail to perform in the presence of noise and do not generalize well. Hence, such classifiers will be unsuitable for deployment in real-world networks.

Although *K*-nn classifier also performed well with our features, it is known to be a 'lazy learner' classifier which has poor run-time performance if the training data

is large.

Bayesian network, NB tree and 'stacking' approach (which used both of these classifiers coupled with REP trees as 'base classifiers', and Naïve Bayes as the meta classifier) performed exceedingly well, indicating that a 'Bayesian' approach is more suitable for such classification problems. These classifiers demonstrated good resilience towards presence of noise in the test data, with the TP rate of almost 90% with the new features.

Although our noise-resistant approach enhances the capabilities of current P2P bot detection mechanisms and raises the bar for bot-creators, it is not perfect.

The attack model used by us considers the range of 25% to 33% of injected noise, and does not consider very low or very high values. We had initially experimented with low levels of injected noise (between 0 to 25%). However, it was observed that the detection models were quite tolerant to low levels of noise. Thus, any discussion on them is not necessary. As the level of injected noise was increased beyond 33%, detection models built with traditional features as well as our features saw degradation in performance. Nevertheless, our detection models continued to give higher TP rate when compared to traditional features. With further higher levels of noise (more than 50%), the TP rate dropped sharply (below 20%) and both detection approaches were rendered quite useless. The proposed approach will not work if a powerful adversary is in a position to launch such attacks with high levels of randomness in the communication pattern of his bots.

If bot-creators can model their bots to mimic benign P2P behavior, our system may find it difficult to identify bots based on their timing or data patterns. If bots are *programmed* to engage in occasional file-sharing, exchange of chat messages, etc. with other peers, their malicious behavior may get subdued by the benign-like activity. We acknowledge that our system may miss such bots. However, creating such a botnet is also fraught with risks for the bot-creator. Activities in

a bot-infected system (occasional file sharing, uploads, downloads etc.), which have not been initiated by the owner of the system, may catch the attention of the owner/administrator, and thus the infection is likely to be detected.

Furthermore, if bots perform C&C communication over anonymity networks like Tor, Tor's anonymity service may help in concealing the C&C activity. Our approach will be unable to detect such bots. However, use of Tor by bots also leaves behind certain recognizable patterns, as pointed out by [Casenove & Miraglia 2014].

4.6 Conclusion

In this work, we presented a novel approach for the detection of P2P bots which relies on conversation-based mechanisms and extracts features based on Fourier transform over the inter-arrival time in a conversation, Fourier transform over payload sizes in a conversation, and compression ratio over payload sizes in a conversation. Using these features, our approach differentiates between benign P2P traffic and P2P bot traffic. By building models with several supervised machine learning algorithms, we demonstrated that our approach could detect P2P bot traffic in presence of benign P2P traffic with high True Positive rate. We also compared the performance of our approach with traditional flow-based approaches. In comparison with flow-based approaches, our approach performs better in the presence of noise injected in the test data.

Through our experiments, we also established that tree-based classifiers—a popular choice in many prior detection approaches such as [Singh *et al.* 2014], [Rahbarinia *et al.* 2014], etc.—are not a good choice for real-world networks since they do not generalize to variation in test data. 'Bayesian' approach fares better in this regard.

We acknowledge that we explored a single attack/noise model in our work. If an adversary is in a position to add high levels of randomness to his bots, our current models will not work. We intend to perform a more detailed study of the effect of deliberate injection of noise in detection of P2P botnet traffic. Furthermore, other concepts from the 'signal-processing' domain, such as Wavelets and Discrete Cosine Transforms, can also be explored for their suitability in identifying malicious traffic.

Chapter 5

Game theoretic strategies for IDS deployment in P2P networks

5.1 Introduction and motivation

A P2P network lacks any centralized authority, and is thus more vulnerable to security threats than the traditional client-server architectures. Although the decentralized and distributed nature of P2P network offers resilience towards network-breakdowns, the super-peer architecture is more sensitive in this regard since an adversary can disrupt (albeit not breakdown) the entire P2P network by attacking the super-peer nodes. For example, a DoS/DDoS attack targeted on certain relay nodes in Tor can lead to an increased latency and higher number of time-outs in the network. In traditional networks, such a scenario can be secured by use of Intrusion Detection Systems (IDS), which might be deployed at the backbone router of an enterprise (NIDS) or at each end-host (HIDS). Owing to its decentralized and distributed nature, a NIDS is not feasible in P2P networks. Furthermore, a HIDS-based solution will provide security only to the node running the IDS, and not to the network. Although a solution based on Distributed IDS (DIDS) can be

explored, self-interested peers may not want to spend their resources in running an IDS. However, such a solution can be viable if this responsibility and load of running distributed IDS is divided amongst different peers.

Early work on P2P networks by [Daswani & Garcia-Molina 2002] showed that super-peer based P2P networks such as Gnutella are susceptible to query-flooding based DoS attacks. Traffic analysis attacks on Skype Voice-over-IP (VoIP) calls, which compromise the privacy of Skype calls, have also been explored using application-level features extracted from VoIP call traces [Zhu & Fu 2011]. Past research has also demonstrated attacks on Tor where anonymity in the Tor network can be compromised by traffic-analysis attacks by a global passive adversary [Murdoch & Danezis 2005] or by non-global adversaries with minimal resources [Bauer *et al.* 2007].

We consider the scenario of a P2P network such as that of Gnutella or Skype which involves a super-peer architecture where super-peers hold higher responsibilities and/or privileges in the network. It is also applicable to networks like Tor which bear resemblance to P2P networks. Tor uses 'relay nodes' which can be equated to super-peers since they take the responsibility of routing and relaying the traffic in the Tor network. This scenario also applies to 'collaborative IDS' involved in 'P2P intrusion detection'.

The P2P architecture is inherently about peers coming together to share and mobilize resources. However, self-interested peers would want to maximize their benefit received from the network and minimize their contribution (in terms of bandwidth, storage, files or data shared, etc.). Consequently, most of the requests for service are directed towards a small number of P2P nodes which are willing to share information or provide service, causing the "tragedy of the commons" [Ma *et al.* 2006] in the P2P network. In order to motivate the peers to spend their resources in running an IDS and contribute to a DIDS approach, it is important

that the load and responsibility of running the IDS is distributed among the peers in the network. Thus, we explore the problem of IDS deployment in P2P networks from a game theoretic perspective.

In this work, we consider the problem of securing a P2P network from an adversary who may become part of the P2P network by joining from any part of the network. A malicious peer can disrupt the P2P network by attacking a super-peer through various attacks at the overlay layer, such as route table poisoning, index poisoning, or other traditional attacks (malicious payloads, etc.). Running an IDS at each peer may not be feasible since self-interested peers may not want to dedicate resources for that. Peers may try to secure the network by running IDS at certain strategic locations in the network. But, a deterministic schedule of running and positioning the IDS can be observed and thwarted by an adversary. In this chapter, we explore the problem of strategically positioning IDS in a P2P network with a game theoretic approach. Our approach distributes the responsibility of running the IDS between the peers in a randomized fashion and minimizes the probability of a successful attack. This approach appears in [Narang & Hota 2015].

This chapter is organized as follows: Section 5.2 presents a detailed discussion on the 'game' environment and explains about the players of the game, the payoffs, etc. with the example of a P2P network. Section 5.3 discusses our proposed solution, and Section 5.4 presents a discussion on the proposed approach. We conclude the chapter in Section 5.5.

5.2 The 'game' environment

For the purpose of this work, we consider a super-peer based P2P architecture, involving super-peer nodes and leaf-peer nodes. In our approach, we limit our discussion to the super-peer nodes. Owing to high 'churn' (joining and leaving

of peers) seen in leaf-peers, any solution involving them is bound to be inefficient since a leaf-peer's lifespan in the network may be very short. Thus, we neglect all leaf peers connected to the super-peers, and consider only super-peers. In general, super-peers are selected in a network based on factors such as high uptime, higher network bandwidth, publicly visible IP address etc., and have lesser churn than leaf-peers.

The P2P network—with only super-peers considered—is modeled as a graph $G(V, E)$. The terms and symbols used in this work are given in Table 5.1. Our approach operates on a snapshot of the network topology, and thus requires the network topology to remain constant. But, since we base our approach only on super-peers, high churn-rate in leaf-peers has no impact on our approach.

Every peer is modeled as a vertex, and links between nodes are modeled as edges between the vertices. An adversary can join the network in the form of a leaf-peer or infect an existing leaf-peer, and thus connect with existing super-peers. The adversary may try to disrupt the network by attacking certain super-peers which hold higher value/responsibilities in the network. The attack(s) can be in the form of overlay attacks or other traditional attacks (malicious packets which cause a buffer overflow, injecting malware in shared files, APTs etc.). For the sake of simplicity, we assume that an attacker gains entry into the network only through a selected set of 'source' nodes.

To model super-peers which hold higher value/responsibilities in the network, we consider certain 'target' nodes in the network and assign a weight or a value to them. These values might be computed based on the node's uptime, the number of super-peers and/or leaf-peers attached to it, its reputation etc. For the purpose of this work, these values were chosen arbitrarily. Furthermore, we model the traffic flowing in the network by assigning weights to the edges. A high weight for edge between two nodes can indicate, for example, higher network traffic over

that link between the peers. An attacker may find it more luring to choose an edge with heavy traffic in its attack path since it would be easier for the attack traffic to blend with heavy network traffic.

Note that it is assumed that other super-peers in the network do not have any value or worth, and thus an attacker will only attack the target nodes. We also assume that the attacker has full knowledge of the network topology, the location of super-peers, the paths leading to them, etc. This is true in real-life networks as well since an attacker is in a position to gain such information through a 'reconnaissance' involving port-scans etc.

Since game theory primarily deals with rational players, we limit our discussion of the attacker or the defender(s) to rational, utility-maximizing players. For the purpose of the game theoretic formulation, we limit ourselves to a zero-sum game – primarily because computational limits are reached in non-zero-sum games even for small network graphs. More specifically, we deal with zero-sum Stackelberg security games. In case of a successful attack, if the attacker gains a payoff of x, the payoff of the defender is $-x$. The payoff is zero for other cases. Limiting this discussion to zero-sum games is important because in finite two-person zero-sum games, different game theoretic solution concepts of Nash Equilibrium, Minimax strategy, Strong Stackelberg Equilibrium (SSE) etc. are equivalent [Korzhyk *et al.* 2011].

5.2.1 The game, players and payoffs

Stackelberg security games are leader-follower games wherein the attacker or the defender takes the first move and the other follows sequentially. In our game formulation, the 'attacker' is a malicious peer who wants to disrupt the P2P network. The 'defenders' are benign super-peers in the network who want to collaboratively

Table 5.1: Terms and Symbols used

Symbol	Meaning
G	The graph of P2P network with vertices V and edges E
V	The vertices in G, which are the super-peer nodes in the network
E	All edges in graph G
T	The set of Target nodes in V
S	The set of Source nodes in V
P_j	The payoff associated with the target node j
$H_{k,j}$	Simple path from source node k to target node j
A	The set of attacker paths chosen $[A = A_1, A_2, A_3 \ldots]$
D	The set of defender allocations $[D = D_1, D_2, D_3 \ldots]$
M_D	Defender's mixed strategy over D
M_A	Attacker's mixed strategy over A
U	Value of the game

ensure smooth and secure operation of the P2P network. We restrict the scope of our discussion to the attacker-defender model, and do not consider the question of trust or effective collaboration amongst the peers in a P2P network, which are separate areas of study covered in past research [Anceaume *et al.* 2005, Ye *et al.* 2004].

We attach values to super-peers which indicate their worth in the network. This value may be chosen based on their contribution to the network, the average number of super-peers/leaf-peers connected to them, their uptime etc. We also assign weights to edges to indicate the amount of network traffic flowing through a link. For this work, we choose these values arbitrarily. We consider the scenario where super-peers in the P2P network want to collaborate to protect those nodes which hold higher value in the network and are thus lucrative targets for an attacker.

Although continuously running an IDS on all peers will provide the maximum security, such continuous monitoring is not viable because peers may not feel incentivized to do so. Although participating peers want successful operation of the P2P network, they are expected to be selfish towards contributing their resources. Moreover, even if multiple IDS are deployed in the P2P network at

certain locations, a deterministic scheduling of IDS can be observed and thwarted by an active adversary.

We use game theoretic strategies to generate a randomized schedule of deploying IDS at different nodes in the network. The schedule is generated in the form of a probability distribution over the peers. This probability distribution may be practically implemented in the form of *(a)* the percentage of time in a given time-slice for which a peer should run the IDS, or *(b)* the *strength* of the IDS running at the node in a given time-slice (with a small probability indicating a light-weight IDS, while a high probability indicating a IDS which can perform heavy tasks such as DPI and SSL inspection in high-speed networks).

This game theoretic schedule will be generated off-line by one of the super-peers in the network – say, by the Skype-owned super-peers or Tor's relay nodes – by considering the 'value' of the 'target' super-peers and the weights of the possible paths which the attacker may choose to attack those nodes.

If the attacker successfully compromises a super-peer whose value is x, the attacker gains a payoff of x plus the weight of the edges which lie on the path from the source to the target. And the P2P network loses an equivalent amount. But, the attacker can launch a successful attack only if there is no active IDS on the path between the attacker and its target node. If there is an active IDS, the attack is detected and thwarted[1]. In this scenario, there is no payoff for the attacker or the defender(s). A non-zero-sum game scenario can consider negative payoff for the attacker launching an unsuccessful attack (which takes into account the cost of launching an attack, the punishment on getting caught etc.), or a cost for the defender(s) to monitor the network. But, as already mentioned, non-zero-sum games are out-of-scope for this work.

[1] we assume a perfect detector

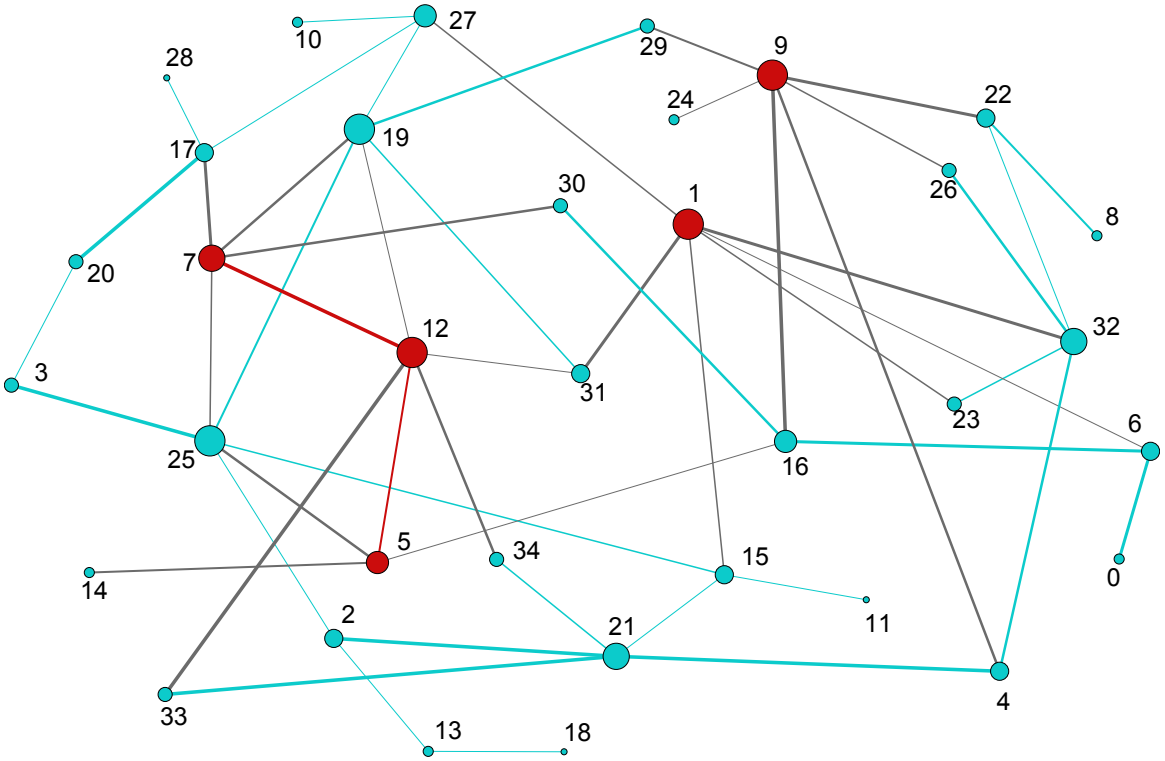


Figure 5.1: A snapshot of the P2P network with only super-peers considered (best viewed in color)

5.2.2 Example

Since our approach deals only with super-peers, we consider a random P2P network topology in simplified form as given in Figure 5.1. All the nodes shown in the figure are super-peers which may have any number of leaf-peers connected to them. The leaf-peers and their connections with super-peers are neglected, and only super-peers and their connections with other super-peers are shown. For the sake of simplicity, let us consider that only five nodes in this network (shown in red) have a value attached to them: nodes 1, 5, 7, 9 and 12. Thus, these are the probable 'target nodes' for a malicious peer, and peers in the network would want to minimize the probability of attack on these targets. All other nodes are taken to have zero value. Further, an attacker may gain entry in the network only through certain 'source' nodes: nodes 0, 6, 8, 10, 21, 26. The complete details of the graph in Figure 5.1 are given in Table 5.2 and 5.3.

Table 5.2: Details of the Network graph given in Figure 5.1

No. of vertices	35 (# 0 to 34)
No. of edges	53
Source nodes #	0, 6, 8, 10, 21, 26
No. of Target nodes	5
Value of target nodes:	
Node #	*value*
1	24
5	17
7	34
9	18
12	32

By noticing the values of target nodes given in Table 5.2, we observe that an attacker can gain twice as much as payoff by attacking node 7 than by attacking node 5. He may connect himself to the P2P through any of the 'source' nodes, and attempt to attack one of the target nodes through any of the possible paths from the source to the target. For example, the attacker may connect to node 6 and attack node 7 using the path `6->16->30->7`, and obtain a payoff equal to the sum of values of each edge and the target. If any node on that path has an active IDS, the attacker will not be successful and he gains zero payoff. Then, the attacker may attempt to attack node 9 using the path `8->22->9`, and so on.

Our solution proposes a randomized strategy for deploying the IDS at different nodes in the network with different probabilities. These probabilities are determined using a game theoretic approach, as described in the next section.

5.3 Proposed solution

Our graph-based modules were implemented using the `igraph` network analysis library, and the game theoretic modules were implemented using the `gambit`

Table 5.3: Details of the edges of the Network graph given in Figure 5.1

Edge	Weight	Edge	Weight	Edge	Weight	Edge	Weight
0-6	6	4-9	5	9-16	7	16-30	5
1-31	6	4-32	5	9-22	6	17-20	7
1-32	6	5-25	5	9-29	4	17-27	1
1-27	3	5-12	4	9-26	3	18-13	1
1-23	3	5-14	4	9-24	1	19-29	5
1-15	3	5-16	1	10-27	2	19-25	4
1-6	2	6-16	6	11-15	1	19-31	3
2-21	7	7-12	7	12-33	7	19-27	1
2-25	2	7-17	6	12-34	5	21-33	7
2-13	1	7-19	5	12-19	2	21-34	3
3-25	7	7-30	5	12-31	1	22-32	2
3-20	2	7-25	3	15-25	3	23-32	3
4-21	7	8-22	4	15-21	2	26-32	5
						28-17	1

library. Both these modules were integrated to create a game theoretic framework which can be applied on any kind of undirected graphs. For the purpose of this work, we demonstrate our solution through the example of a random graph topology as given in Figure 5.1.

The network snapshot in Figure 5.1 represents an overlay network of trusted peers modeled by the undirected graph $G(V, E)$. An Intrusion Detection System (IDS) can run on each node (super-peer). Our work uses a Distributed IDS (DIDS) approach. The IDS can monitor the network traffic on all the edges connected to that node. The resources spent by the network in running IDS can be conserved if the IDS are strategically deployed in the network and all peers do not need to run the IDS. Furthermore, this allocation must be done in such a way that the probability of attack on the 'target nodes' is kept to the minimum. The peers (super-peers, to be precise) play the role of defenders who want to defend the 'target nodes' from probable attacks, and collaborate to save their resources at the same time.

Each target node j has a payoff (value) associated with it, given by P_j (a real number). The value of P_j for each target node is determined by the relative importance of the node, which may be modeled as a function of its up-time, associativity etc.

A pure strategy D_i for the defender is to activate the IDS on node i. The set of all such allocations is D. A pure strategy A_j for the attacker is a path from any 'source' node to one of the target nodes j in T. The set of all such paths is A. The game is a zero-sum game. The attacker gets a utility (or payoff) of P_j for successfully attacking the target j, and zero otherwise. Similarly, the defender gets a utility of $-P_j$ if the attacker successfully attacks target j. Success and failure are defined by the intersection of the Attacker and Defender allocations. If the defender has used node i in its allocation and an attacker path A_x passes through i, the attacker will get detected and his attack will fail.

The value of the game is modeled as the utility derived by the attacker for playing the mixed strategy M_A over A. The objective is to find a Minimax strategy M_D for the defender. Since it is a zero-sum game, the Minimax solution is also a Nash Equilibrium.

Below, we describe the computation of the solution for this attacker-defender game. This computation for the solution is done off-line by the defender(s) by computing the best responses for the attacker as well as the defender in each situation.

The solution space of the defender will grow as the number of IDS running in the network increase. If there is only a single IDS running in the network (as considered in our preliminary work [Narang *et al.* 2014c]), the defender's solution space equals the number of nodes, i.e. $|V|$. For a solution involving n IDS, the Defender's strategy space will grow to ${}^{|V|}C_n$. For the sake of simplicity, we will explain the details of our algorithms by considering a single IDS. However, the proposed solution is equally valid with multiple IDS. The results obtained with

multiple IDS will also be discussed later sections.

5.3.1 'Trivial' zero-sum game

Since an attacker is targeting some selected 'target nodes' in the P2P network, we describe a trivial solution wherein the IDS is run on the target nodes themselves. In this case the attacker's strategy space will be filled with a simple path from any source node to each of the target nodes: $A = A_1, A_2, A_3 \ldots$ where each A_j is an arbitrarily chosen simple path to target node j. The defender's strategy space will include all the target nodes: $D = D_1, D_2, D_3 \ldots$ where each D_j is a target node when number of IDS are 1. For n IDS ($n > 1$), each D_j consists of a tuple of n target nodes. The algorithm is presented in Algorithm 5.1:

Algorithm 5.1: Trivial Solution

```
1 begin
2     for each node j ∈ T do
3         k = ChooseRandomNode(S);
4         A = A ∪ H_{k,j};
5     D ← T;
6     (M_A, M_D) = MMSC(A, D);
```

The function $ChooseRandomNode(S)$ will select a random 'source' node k from the set of source nodes S. In its next step, a random path between k to j is added to A.

MMSC implies Minimax Mixed Strategy Calculator. It calculates attacker's and defender's mixed strategies M_A and M_D respectively over A and D using the Von Neumann's Minimax Theorem [Motwani & Raghavan 2010]. This works by formulating linear equations and solving them as follows: Both, the attacker and the defender, aim at making the other player indifferent towards choosing any of the available strategies. This is achieved by equating the average payoff that the opponent receives on choosing any of the available strategy. This is known as

the Principle of Indifference (For detail, we refer the reader to the Chapter 4 of [Keynes 2013]).

If the game matrix between the attacker and defender is represented as C having $A = A_1, A_2, A_3 \ldots$ and $D = D_1, D_2, D_3 \ldots$, then

$$C = \begin{bmatrix} c_{11} \ldots & c_{1m} \\ c_{21} \ldots & c_{2m} \\ \ldots & \\ \ldots & \\ c_{n1} \ldots & c_{nm} \end{bmatrix}$$

The attacker is the row player and the defender is the column player. c_{ij} is the payoff to the attacker when attacker chooses A_i and defender chooses D_j. M_A is represented as n-tuple $(p_1, p_2, p_3 \ldots)^T$ such that $\sum p_i = 1$. Similarly, M_D is the m-tuple $(q_1, q_2, q_3 \ldots)^T$ such that $\sum q_i = 1$. If attacker chooses the mixed strategy M_A and defender chooses a pure strategy D_j, then the average payoff to attacker is $\sum_{i=1}^{n} p_i c_{ij}$. The defender (who is choosing only the strategy j) receives its exact negative. Similarly, if the defender chooses M_D and attacker chooses a pure strategy A_i then the average payoff to the defender is $-\sum_{j=1}^{m} q_i c_{ij}$. Alternatively, the pure strategy of selecting a single row or column can be represented as a unit vector e_i or e_j where all elements are zero except the i^{th}/j^{th} elements, which are 1.

The probability distribution (mixed strategy) for attacker is found by solving the equations

$$\sum_{i=1}^{n} p_i = 1 \tag{5.1}$$

and

$$\forall j, k \in D, \sum_{i=1}^{n} p_i c_{ij} = \sum_{i=1}^{n} p_i c_{ik} \quad for \quad j \neq k \tag{5.2}$$

Table 5.4: Trivial solution for graph in Figure 5.1

Number of IDS	Convergence time	Value of the game	Prob. dist. over IDS nodes
1	4 sec	24.15174709	0.2452 ← [1] 0.2991 ← [7] 0.1823 ← [9] 0.2732 ← [12]
2	5 sec	17.2453919	0.1421 ← [1, 5] 0.3189 ← [1, 7] 0.0578 ← [7, 9] 0.1227 ← [7, 12] 0.3582 ← [9, 12]
3	5 sec	11.49692794	0.3459 ← [1, 5,7] 0.0432 ← [1, 5, 12] 0.2515 ← [1, 9, 12] 0.0388 ← [5, 9, 12] 0.3204 ← [7, 9, 12]
4	5 sec	5.748463968	0.1729 ← [1, 5, 7, 9] 0.1946 ← [1, 5, 7, 12] 0.1668 ← [1, 5, 9, 12] 0.2859 ← [1, 7, 9, 12] 0.1796 ← [5, 7, 9, 12]
5	6 sec	0	1.0000 ← [1,5,7,9,12]

In a similar way, the mixed strategy for the defender is found by solving

$$\sum_{i=1}^{m} q_j = 1 \tag{5.3}$$

and

$$\forall i, l \in A, \sum_{j=1}^{m} q_j c_{ij} = \sum_{j=1}^{m} q_j c_{lj} \quad for \quad i \neq l \tag{5.4}$$

The value of the game is the average payoff to attacker when both play the above mixed strategy.

$$U = \sum_{i=1}^{n} \sum_{j=1}^{m} p_i c_{ij} q_j \tag{5.5}$$

The Solution: With the trivial solution, we obtain a probability distribution to run the IDS on the target nodes. The solution thus obtained for the graph in Figure 5.1 is given in Table 5.4. We give results for number of IDS in the network ranging from 1 to 5, given in column 1 of the table. The time taken for the game to reach convergence, i.e. the 'convergence time', is given in column 2. Column 3 gives the value of the game obtained for each run. A single IDS solution has the highest value of the game, and thus results in the highest loss to the defender. As we increase the number of IDS, the value of the game falls heavily. The time for convergence of the game does not vary significantly with the increase in number of IDS.

In column 4, the probability distribution obtained over the participating peers is listed. Note that only 'target' nodes appear in the probability distribution. With 5 IDS running on all 5 target nodes, the value obtained by an attacker reduces to zero. This is but obvious since highest security is achieved for target nodes by running IDS at all the target nodes themselves. If all targets are running IDS, the attacker will always be caught. Although it gives the highest security, this is also the most *expensive* solution since we are putting the burden of running IDS on all the target nodes themselves. Thus, such a solution is called 'trivial'. A better case will be explored in the non-trivial solution when we do not have the IDS running on the 'target' nodes.

5.3.2 'Non-trivial' zero-sum game

The target nodes are super-peers which hold certain prime responsibilities or higher privileges in the P2P network. They already have high load on them. A good example would be of Tor's relay nodes. The trivial solution proposed running IDS on the target nodes themselves. Since the targets themselves become defenders, surely such a solution achieves lowest value of the game. However, at

the cost of slightly higher value of the game, we explore the possibility of running the IDS on super-peers apart from the target nodes (and thus reduce the burden on the target nodes). This 'non-trivial' case is explained in Algorithm 5.2. We follow the same notations as used in the trivial solution. The value of the game is calculated in the same way as in the case od trivial solution. The functioning of **MMSC** also remains same. In the algorithm, A.B.R. and D.B.R. stand for 'Attacker's Best Response' and 'Defender's Best Response' respectively. We discuss about them next.

Algorithm 5.2: Non-trivial Solution

```
   Data: G = (V, E)
   Result: (M_D, M_A)
1  begin
2      D ← ChooseRandomNode(V − T);
3      A ← ChooseRandomPath(S, T);
4      while U_old!=U_new do
5          Calculate U_old;
6          (M_D, M_A) = MMSC(D, A);
7          a = A.B.R.(D, A, U_old);
8          d = D.B.R.(D, A, U_old);
9          D = D ∪ d;
10         A = A ∪ a;
11         Calculate U_new;
12     return (M_D, M_A);
```

The function $ChooseRandomNode(V - T)$ will select a random node from the set of nodes in $(V - T)$, which implies the set of all nodes apart from the 'target' nodes. We exclude the target nodes because the non-trivial solution explores IDS deployment without considering them. $ChooseRandomPath(S, T)$ will choose an arbitrary attack path for the Attacker from an arbitrary source in S to any of the target nodes in T.

Attacker's Best Response (A.B.R.): Given the defender's strategy M_D, A.B.R. determines the best response by the attacker. The response in this case is addition

of a new path to the attacker's strategy space which maximizes the value of game (or equivalently, the payoff to the attacker). For this, the attacker adds a simple path (p), from an arbitrary source node to a possible target node, to its strategy space A. It now calls MMSC with this modified strategy space (A') and defender's strategy space (D) to arrive at Minimax Mixed Probability Distributions. Using these new probability distributions, the value of the game (U') is calculated in the same way as outlined in the trivial solution case. This is repeated for all possible simple paths from all possible sources to all the target nodes. The path yielding maximum value (U') is selected as the best response. It is explained in Algorithm 5.3.

Algorithm 5.3: Attacker's Best Response

```
Data: D, A, U_current
Result: q
1  begin
2      U_max ← U_current;
3      q ← NULL;
4      for ∀ paths p from all source to all targets do
5          A' = A ∪ p;
6          (M'_D, M'_A) = MMSC(D, A');
7          Calculate U' using (M'_D, M'_A);
8          if U' > U_max then
9              U_max = U';
10             q = p;
11     return q
```

Defender's Best Response (D.B.R.): Given Attacker's mixed strategy M_A, D.B.R. calculates the best response by the defender. The defender aims to minimize the value of the game (thereby minimizing its own loss) and hence the best response will be the addition of such a node to the defender's strategy pool which reduces the current value of the game to the minimum possible. In order to find the best-response node, the defender adds a single node to its already existing strategy space (D) and uses MMSC with this new strategy space (D') and attacker's strat-

egy space (A) to arrive at the Minimax Mixed Strategy Distributions. These new distributions are used to calculate the value of the game (U') at this point, in the same way as mentioned in the trivial solution case. This is done for all the (super) peers in the network and the one yielding minimum value (U') is selected as the best response. This can be explained algorithmically in Algorithm 5.4.

Algorithm 5.4: Defender's Best Response

```
Data: D, A, U_current
Result: r
begin
    U_min ← U_current;
    r ← NULL;
    for ∀ nodes n ∈ {V − T} do
        D' = D ∪ n;
        (M'_D, M'_A) = MMSC(D', A);
        Calculate U' using (M'_D, M'_A);
        if U' < U_min then
            U_min = U';
            r = n;
    return r
```

The Solution: With the non-trivial solution, we obtain a probability distribution for running the IDS on super-peers other than the target nodes. For the graph described in Figure 5.1, the non-trivial solution obtained is given Table 5.5. Note that the probability distribution over participating peers does not include any target nodes.

The number of IDS kept equal, the non-trivial solution leads to a higher value of the game in each run (as compared to the 'trivial' case). It is close to the value of the game in the trivial solution when number of IDS are small (1 or 2), but the difference becomes quite notable as we increase the number of IDS. Moreover, increasing the number of IDS also leads to significant increase in the convergence time of the game. This will be further discussed in Section 5.4.

As we had mentioned before, the value of the game will be lowest in the trivial

Table 5.5: Non-trivial solution for graph in Figure 5.1

Number of IDS	Convergence time	Value of the game	Prob. dist. over IDS nodes
1	6 sec	25.55339806	0.5214 ← [3] 0.4258 ← [4] 0.0304 ← [21] 0.0221 ← [25]
2	8 min	17.62875407	0.0904 ← [0, 3] 0.0844 ← [3, 21] 0.2693 ← [6, 19] 0.0074 ← [8, 19] 0.0454 ← [6, 21] 0.0071 ← [6, 32] 0.0064 ← [8, 21] 0.2334 ← [10, 32] 0.0531 ← [15, 32] 0.2025 ← [21, 32]
3	1 hr 18 min	14.44420696	0.0417 ← [0, 2, 27] 0.0098 ← [0, 8, 25] 0.0606 ← [0, 10, 25] 0.0855 ← [0, 21, 27] 0.1778 ← [0, 2, 32] 0.1781 ← [8, 10, 25] 0.1778 ← [0, 3, 32] 0.0907 ← [0, 10, 32] 0.1775 ← [2, 26, 27]
4	7 hrs	11.25517573	0.0038 ← [0, 8, 26, 27] 0.3096 ← [0, 3, 26, 27] 0.0420 ← [0, 2, 21, 32] 0.2968 ← [0, 8, 10, 21] 0.0241 ← [0, 2, 8, 21] 0.0908 ← [0, 2, 3, 21] 0.2325 ← [31, 32, 33, 34]
5	18 hrs	6.5640430556	0.0801 ← [30, 31, 32, 33, 34] 0.3133 ← [0, 2, 6, 8, 32] 0.0870 ← [0, 2, 6, 21, 32] 0.1347 ← [0, 2, 6, 10, 21] 0.1719 ← [0, 2, 8, 10, 21] 0.2129 ← [0, 2, 10, 21, 32]

case since the targets themselves become the defenders. Running the IDS on nodes other than the target nodes does result in a small demerit in terms of higher value of the game (and equivalent negative payoff to the defender), but it saves the target nodes from the extra load of running an IDS.

5.4 Discussion

The results given in sections above are for a single, random graph topology (as given in Figure 5.1). However, these experiments were repeated with multiple random graph topologies in order to understand the impact of various parameters on the convergence of the solution, such as increasing the degree of target nodes, increasing the number of edges in the graph, etc.

The defender's strategy space grows to ${}^{|V|}C_n$ for a solution involving n IDS. The solution space is dependent only on two parameters of number of vertices ($|V|$) and number of IDS (n). However, convergence of the game is dependent on the strategy space of the defender as well as the attacker. Attacker's strategy space is defined by the number of paths he can take to attack the target nodes. Every new edge added to the graph will create multiple new paths to the target nodes. Addition of new edges will significantly change the attacker's strategy space.

As noted above, neither the defender's strategy space nor the attacker's strategy space is directly dependent on the degree of the target node. As expected, increasing the degree of target nodes had no impact on the convergence time of the game. In fact, if number of edges in the graph are kept constant, increasing the degree of target nodes may reduce the total number of paths in the graph, thereby decreasing the time for convergence. However, increase in the number of edges had a significant impact. As the number of edges are increased, the number of paths to the target also increase. With the increase in paths, the time for conver-

gence of the game grew significantly. For example, for a random graph topology with double the vertices and edges as compared to the graph used in work, it took nearly 24 hours for the convergence of the non-trivial solution with 3 IDS. The convergence time of trivial solution, however, did not vary much irrespective of the change in the graph input.

The output of our approach is in the form of a probability distribution over super-peer nodes. This probability distribution may be interpreted in terms of 'chunks of time' for which these nodes will run the IDS in any given time-slice. In general, this time slice should not be very large. If the time-slice is very large, the IDS will remain fixed for a long time. An attacker may be able to learn the position of the IDS by, say, getting caught once, and then evade the IDS in the next turn by choosing a different path. The strength of our approach lies in its randomized, game theoretic approach. A very high 'time-slice' value will make it ineffective. We envision a value of about one hour to be suitable for most practical purposes. Another way to interpret this probability distribution is in terms of the *strength* of the IDS running at that node in any given time-slice. A node at a less strategic location may run a light-weight IDS, while a node deemed to be at a highly strategic location may be required to perform intensive intrusion detection (with DPI/SSL inspection capabilities in high-speed networks).

It must be noted that our approach operates on a snapshot of the network. If the P2P network's topology changes significantly due to high peer-churn, the solution will need to be re-computed. However, churn rate is higher in leaf-peers than in super-peers. Rather, a node with short uptime is usually never chosen as a super-peer. Since our approach does not rely on leaf-peers, it is unaffected by new leaf-peers joining or leaving the network. As far as the issue of churn in super-peers is concerned, it will not only impact our approach, but also effect the attacker's knowledge of the network. The attacker will be forced to perform

a fresh reconnaissance to gain knowledge of the network. In fact, even the P2P network's routing and indexing does get impacted by ungraceful departure of an important node.

5.5 Conclusion

This work presented an evaluation of game theoretic strategies for IDS deployment in P2P networks. The P2P network was modeled in the form of a graph. All leaf-peers were neglected and only super-peers were considered for the modeling. An attacker may try to gain control of one of the super-peer nodes, and use it to launch attack on nodes with higher value in the network (i.e., the 'target' nodes). We modeled the P2P nodes in the network as the defenders who wish to protect the network from an arbitrary attacker. Zero-sum Stackelberg games were used to model the game between the attacker and the defender. By using game theoretic approaches, we obtained a probability distribution over the super-peers to run an IDS. We demonstrated the results obtained by our approach for number of IDS ranging from 1 to 5. Two different solutions were demonstrated – a trivial solution, wherein the responsibility of running the IDS lies on the target nodes themselves, and a non-trivial solution, wherein the responsibility of running the IDS is given to nodes other than the target nodes.

In this work, the value of target nodes and weights of edges were chosen arbitrarily in order to simulate random graph topologies. In future work, we plan to perform detailed modeling of these values in order to obtain more comprehensive simulation of scale-free networks.

This work considered a zero-sum, Stackelberg security game. The game theoretic solution concepts of Nash Equilibrium, Minimax strategy and Strong Stackelberg Equilibrium (SSE) etc. are equivalent in finite two-person zero-sum games. How-

ever, considering non-zero-sum games will bring up new challenges in terms of identifying the equilibrium solution. In future work, we wish to explore non-zero-sum games and also consider solution concepts beyond the Nash equilibrium.

Chapter 6

A Hadoop-based framework for detection of P2P botnets

6.1 Introduction

With enterprise-level networks regularly generating billions of events and gathering Terabytes of data each day, tracking malicious activity inside a network is nothing less than the proverbial *needle in the haystack* problem. The evolution of Peer-to-Peer (P2P) based botnets, which have a distributed and decentralized architecture, has created further challenges in detection of malicious activities.

Detecting P2P botnets is a challenging task because P2P botnet traffic can very easily blend with benign P2P traffic in a network, like that of Gnutella, BitTorrent, eMule etc. Although many approaches have been suggested which evaluated the detection of P2P botnets in Internet traffic [François *et al.* 2011] or provide mechanisms for the detection of P2P botnets in the presence of benign P2P traffic [Rahbarinia *et al.* 2014], building a scalable detection framework has received very little attention (such as in [Zhang *et al.* 2014]).

Most of the previous work utilizing network behavior of botnets uses the traditional 5-tuple flow-based analysis of network traces. The 'flow' information is typically obtained in the form of Cisco's NetFlow (or by using tools like Argus[1]) from a backbone router of an enterprise. Large-scale networks may involve multiple routers. The NetFlow data collected at one router will not give a complete picture of the communications which happened to and from the network. A distributed data collection approach – where data collectors sit closer to the nodes in the network – can give a much better view of the communications. Such a distributed approach is especially beneficial and essential for the detection of smart P2P bots *inside* the perimeter of a network, which talk to each other and send upgrades to themselves on LAN in a P2P fashion, and limit communication to the outside world via one or two peers only. The activity of such bots, which communicate to each other on LAN in a P2P fashion, cannot be detected by traditional 'flow-based' approaches which only monitor the data crossing the backbone router(s). However, most enterprise network see Gigabit speeds at their border routers itself. Traffic on LAN is expected to be of much higher volumes. A scalable framework is necessary for any approach which handles LAN traffic.

Table 6.1 shows the statistics of a one minute capture of the network traffic at the backbone router of the author's University. The statistics were generated using the `capinfos`[2] tool which is a part of the Wireshark (a popular network analyzer) tool. In a 60 second duration, nearly 1.8 GB of network traffic was generated. The total number of packets seen were 17,22,539, which comes out to be 28,792.50 packets per second. The reader is urged to note that this only accounts for the inside-to-outside and outside-to-inside traffic. LAN traffic – generated by LAN gaming or P2P applications such as Direct Connect – often sees much higher speeds and is not a part of these statistics. Such high volumes of data (coupled

[1] www.qosient.com/argus/
[2] https://www.wireshark.org/docs/man-pages/capinfos.html

Table 6.1: Statistics of one minute capture of network traffic from the backbone router of the author's University

File name:	testrun_00001_20150813114729.pcap
File type:	Wireshark/tcpdump/... - libpcap
File encapsulation:	Ethernet
Packet size limit:	file hdr: 65,535 bytes
Number of packets:	17,22,539
File size:	1,792,405,888 bytes
Capture duration:	60 seconds
Start time:	Thu Aug 13 11:47:29 2015
End time:	Thu Aug 13 11:48:29 2015
Data byte rate:	29,499,658.04 bytes/sec
Data bit rate:	235,997,264.32 bits/sec
Average packet size:	1,024.56 bytes
Average packet rate:	28,792.50 packets/sec

with the need of inside-to-inside traffic visibility) motivated us to use the Hadoop ecosystem for building a distributed and scalable framework for the detection of P2P botnets.

In this work, we present `Hades`, which is an acronym for '**H**ost-**a**ggregation based **de**tection **s**ystem' for P2P botnets. `Hades` utilizes the distributed computing power of the Hadoop ecosystem to parse large network traces and extract 'behavioral' features for every P2P host seen in network communications. The extracted feature-set is then used to train supervised machine learning models which can differentiate P2P botnets from P2P applications. `Hades` does not require signature-based detection approaches, DPI or a 'seed' information of bots obtained from a blacklist of IPs. `Hades` just relies on the header information in the network and transport layer, and extracts statistical features which quantify the 'P2P' behavior of the P2P applications running on a host. This approach appears in [Narang *et al.* 2014e].

`Hades` addresses certain limitations of past works and makes the following con-

tributions:

1. `Hades` is built on top of the Hadoop ecosystem, which is a *de facto* standard for big data analytics. Since it utilizes the power of distributed computing through Hadoop [Bialecki *et al.* 2005], `Hades` is scalable by design.

2. We propose a distributed data collection architecture wherein data collectors are placed at multiple locations inside an enterprise network and sit close to the peers, say at a Distribution switch or a Wi-Fi access point. This approach allows *inside-to-inside* communication view, which can be vital for detecting P2P botnets inside a network which communicate to each other over LAN. `Hades` is the first attempt at distributed data collection for the detection of P2P botnet traffic.

3. `Hades` adopts a Host-aggregation based approach which obtains statistical features *per host* for all P2P hosts involved in network communication.

In order to facilitate reproducible research, we also discuss the implementation aspects of `Hades` in detail.

This chapter is organized as follows: Section 6.2 presents the system design of `Hades` and provides its implementations details. In Section 6.3, we evaluate this host-aggregation based approach and present the results for detection of P2P botnets. Section 6.4 discusses the limitations of `Hades` and possible evasions by P2P botnets. We conclude this chapter in Section 6.5.

6.2 System design and implementation details

The system design of `Hades` employs the `libpcap` library for collecting and parsing network traces. It utilizes the Hadoop ecosystem for aggregation of host-based data and for building scalable models for the detection of P2P botnets. `Hades` has

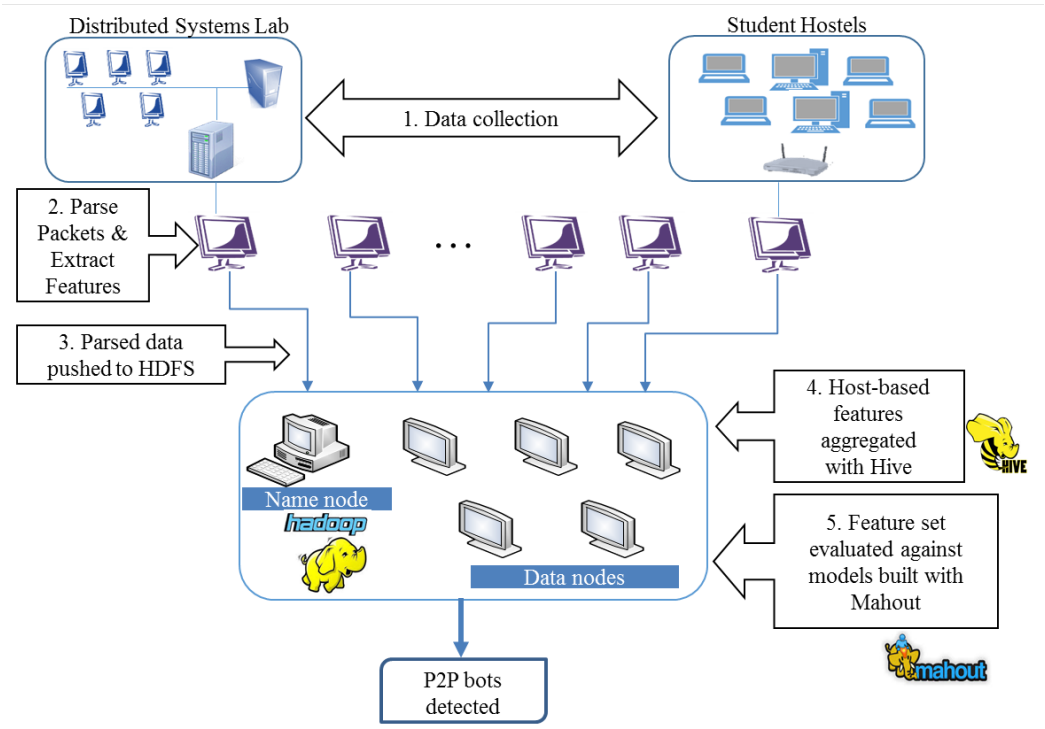


Figure 6.1: Hades: system architecture

been implemented on top of the Hadoop ecosystem with the open-source projects of Apache Hive [Thusoo *et al.* 2009] and Apache Mahout [Mahout 2012]. The system architecture of `Hades` is given in Figure 6.1.

6.2.1 Distributed data collection

Instead of relying upon NetFlow data obtained at a backbone router, `Hades` proposes a distributed data-collection technique wherein data collectors sit close to the peers inside the network perimeter. As mentioned above, placing data collectors closer to the peers allows `Hades` to have a view of inside-to-inside conversations and inside-to-outside (or vice-versa) conversations as well.

The implementation of `Hades` has multiple data collectors distributed inside the network perimeter. The prototype deployment of the system has data collectors deployed at Wi-Fi access points within the University campus of the author. The

multiple data collectors consist of commodity-grade hardware machines with 2 GB RAM, 2 CPU cores and 200 GB of disk space. The Wi-Fi access points used are NetGear N150 and Belkin N150.

Each data collector uses a `libpcap` library based module to capture traffic in the form of network traces `.pcap` files. Each data collector runs an automated parser module (built with `libpcap` library and Python) which parses the network traces and extracts packet-level features of interest to us. Features are extracted from the IP header and TCP/UDP header, and no DPI is performed. For this work, the features extracted from each packet are:

1. Time-stamp of the packet
2. Source IP
3. Destination IP
4. Time-to-live (TTL) value
5. Transport layer protocol (TCP/UDP)
6. TCP or UDP payload length (as applicable)

The extracted features are stored in a `.csv` file at each data collector. Instead of transferring large `.pcap` files, these `.csv` files are periodically transferred from all data collectors to the Hadoop Distributed File System (HDFS) [Borthakur 2011]. For the purpose of data sanitization, all packets which are found to *not* contain a valid IPv4 header are removed (E.g.- corrupted packets). The present approach of `Hades` also disregards all packets corresponding layers below the IP layer, such as ARP broadcast messages. The implications of this choice will be further discussed in Section 6.4.

6.2.2 Host data aggregation

The Hadoop cluster deployed for `Hades` consists of a 'name node' Virtual Machine with 8 GB RAM, 8 CPU cores and 200 GB disk space, and ten 'data node' Virtual Machines, each having 2 GB RAM, 2 CPU cores and 200 GB of disk space. Each Virtual Machine runs Ubuntu 12.04 Operating System.

Packet-level data obtained from multiple data collectors is aggregated *per host* for every host seen in network communication. The packet-level data is stored in Hive in the form of external tables. Hive commands are written in HQL (Hive query language) which is very similar to SQL [Thusoo *et al.* 2009]. The Hive command used to create the table for storing packet-level data is given here:

```
CREATE EXTERNAL TABLE packet_data (
timestamp DECIMAL, ip_source STRING,
ip_destination STRING, ttl INT,
proto INT, payload_length INT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION '/user/hdfs/PacketDump';
```

For the task of detecting P2P botnets, we aggregate the following statistical features over a time-period T (say, one hour) for every P2P host inside the network:

1. **Number of distinct destination hosts contacted:** P2P hosts are involved in sharing content and downloading different chunks of a file from different peers across the globe. A benign P2P host might be involved in downloading a certain file (or its chunk) from Adelaide, uploading a file to another peer at Birmingham, and download music content from a peer at California. As a normal user of the Internet engaged in P2P file sharing, a benign P2P host is expected to contact a number of peers in the different parts of the world, with no specific pattern involved in the destinations contacted. Moreover,

due to the sheer size of these networks, most interactions in P2P file sharing networks are one-time transactions, where peers who share content with each other may never interact again. But, the behavior of hosts infected by P2P botnets gives a contrast. Bot-peers do not engage in file sharing or downloads. Rather they regularly and repeatedly contact their set of bot peers to receive or propagate commands and updates. Thus, the number of unique destination hosts contacted by bot-peers are expected to be less as compared to benign P2P hosts.

2. **The total volume of data sent from the source host:** As stated above, benign P2P hosts are engaged in file transfers, downloads, uploads etc. whereas bot hosts are not expected to be engaged in these activities. The volume of data sent from a benign host is, quite clearly, expected to be more than the exchange of data seen at bots which are primarily involved in exchange of C&C information.

3. **The average of the TTL value of the packets sent from the source host:** A user of P2P file sharing systems who is involved in downloading some music content will not bother whether the seeding peers happen to be from his/her home country or some other part of the world. Rather, while the user might himself be situated in India, he may download one part of the file from a peer in China, another chunk from a peer in Holland, and another from a peer in Australia. Since the file requests of benign P2P users travel all over the world, these requests typically have high TTL values associated with them. In contrast, bot hosts tend to repeatedly contact their set of bot-peers. For the sake of efficient design and avoiding latency/overheads, bot-masters would not want their bots to talk to peers in different parts of the world. Bots are expected to engage in communication with other bot peers near to them. This leads to the requests sent by bots having lower TTL values when

compared to requests seen from benign P2P hosts. However, we admit that it is possible to see this behavior in benign P2P applications as well if their search and routing functionalities have been optimized in this regard.

The host-aggregated features described above are stored in another table:

```
CREATE TABLE host_data (
host STRING, destinations DECIMAL,
avg_ttl DECIMAL, volume BIGINT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,’
LINES TERMINATED BY ‘\n’ STORED AS TEXTFILE;
```

Since the packet-level data arrives from different data collectors periodically, a Hive script is run periodically to convert it into host-aggregated form and store it in the table `host_data` created above. The Hive script given below uses a ‘GROUP BY’ operation to obtain data in host-aggregated format as described below:

```
INSERT INTO TABLE host_data
SELECT ip_source, COUNT (DISTINCT ip_destination),
AVG(ttl), SUM(payload_length)
FROM packet_data
GROUP BY ip_source;
```

6.2.3 Detecting P2P bots from hosts

The host-based features extracted above are used to train and test supervised machine learning models. Apache Mahout is used for this purpose. Mahout is a fairly new tool, and at present does not offer many machine learning algorithms. Further, many of Mahout’s algorithms (for classification and clustering) do not run as MapReduce jobs. Parallelized implementations are important for scalability of

`Hades` over large datasets. Thus, for this work, we stick to the Random forest implementation of Mahout which is a parallelized implementation (in contrast to other implementations like Linear regression, AdaBoost etc., which are not). More details on the data used are given in the next section.

Results generated from `Hades` can be used to alert a network administrator for suspicious activity in the network, trigger rules to a firewall, and/or log or drop botnet traffic. This way, `Hades` can be used by network administrators as an assisting tool which is 'P2P-aware'.

6.3 Evaluation and results

For evaluation of `Hades`, we use P2P data obtained from the University of Georgia [Rahbarinia *et al.* 2014]. Our dataset consists of network traces of two P2P applications, namely Vuze and Frostwire, and two P2P botnets, namely Storm and Waledac. The data of P2P applications was generated by [Rahbarinia *et al.* 2014] by running these applications in their lab environment for a number of days. The data of P2P botnets was obtained by them from third-parties, and corresponds to real-world traces of these botnets.

The size of this dataset was around 20 GB (14 GB for Vuze and FrostWire, and 6 GB for Storm and Waledac) in `.pcap` format, and around 10.5 GB (6.6 GB for Vuze and FrostWire, and 3.9 GB for Storm and Waledac) when parsed to `.csv` format.

After extracting host-based features from each application, we created a 'labeled' dataset. Instances belonging to P2P hosts (Vuze or Frostwire) are labeled 'benign', while the instances belonging to P2P bots (Storm or Waledac) are labeled 'malicious'. This dataset was split into training and testing data in 2:1 ratio. With the training data, Random forest models were built for different number of trees in

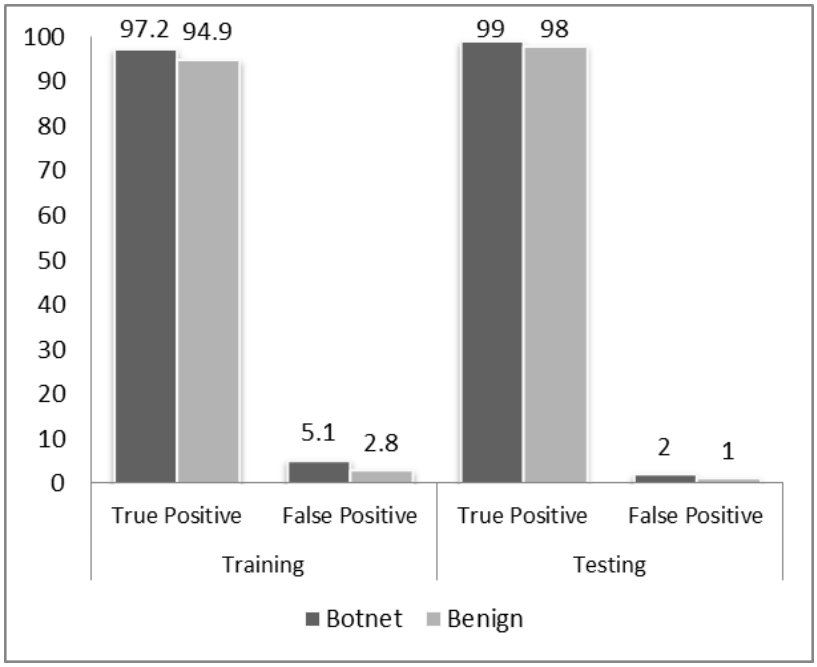


Figure 6.2: True Positive rate and False Positive rate with training and testing data for Random forests of 10 trees

each run. The models were then evaluated for their accuracy with the test data. We limit the discussion on results to Random forest of ten trees since that number gave the highest accuracy. The 'confusion matrix' obtained over training data and testing data is given in Table 6.2. Figure 6.2 and Table 6.3 show the accuracy obtained for the training and testing data with a Random forest of ten trees. Our system could detect bot-infected hosts with a True Positive rate of 97% and 99%, and a low False Positive rate of 5% and 2% over training and testing datasets respectively.

6.4 Limitations and possible evasions

`Hades` utilizes host-aggregation based features which can either classify a host as 'P2P benign' or 'P2P malicious'. `Hades` cannot attribute the exact P2P application (malicious or benign) running on a host. Also, if a bot-infected machine is running a P2P application, it is highly likely that the higher volumes of benign traffic will

Table 6.2: Confusion Matrix for Training and Testing data for Random forests with 10 trees

Training Data			
Confusion Matrix			
	botnet (predicted class)	benign (predicted class)	Total Instances
botnet (actual class)	**78,508**	4,625	83,133
benign (actual class)	7,753	**76,400**	84,153
Testing Data			
Confusion Matrix			
	botnet (predicted class)	benign (predicted class)	Total Instances
botnet (actual class)	**35,094**	970	36,064
benign (actual class)	1,528	**34,536**	36,064

subdue the vision of malicious activity from `Hades`. Our approach will be unable to correctly classify it as an infected host.

Further, it was explained in Section 6.2 that `Hades` ignores messages below the IP layer, such as ARP broadcast messages. Its implications will be discussed here. We had argued on the case of 'smart' P2P bots which may exchange C&C with peers on a LAN and limit communication with the outside world to one or two peers. A bot-master may also configure such smart bots to utilize protocols lower than the IP layer – such as ARP messages – to facilitate communication between the bots on the same LAN. `Hades` will not be able to detect the communication of such bots since it does not deal with those messages in its present approach. Although no past work has touched upon this issue and no such botnets are known to exist at present, we argue that with the evolution of botnet detection mechanisms, bot-masters will also improvise their botnets in these ways to make them more efficient and harder to detect.

It may be noted that Hadoop-based solutions are natively batch processing based, and are not expected to be real-time. `Hades` does not aim to be real-time in detection of P2P bots. Rather, the proposed framework is meant to deal with and process huge volumes of data generated in enterprise networks (say the network

Table 6.3: Accuracy obtained for Random forests with 10 trees

Training Data		
Total Classified Instances:	167,286	
Correctly Classified Instances:	154,908	92.6007%
Incorrectly Classified Instances:	12,378	7.9393%
Testing Data		
Total Classified Instances:	72,128	
Correctly Classified Instances:	69,630	96.5367%
Incorrectly Classified Instances:	2,498	3.4633%

logs of past week or month), which surpasses the processing power and RAM capacities of server-grade machines.

6.5 Conclusion

This work presented `Hades`, an approach to collect P2P data inside a network in a distributed manner, and extract host-aggregated features to distinguish between P2P applications and P2P botnets using supervised machine learning approaches. To the best of our knowledge, `Hades` is the first attempt at distributed data collection for the detection of P2P botnet traffic.

The distributed data collection architecture proposed by us gives *inside-to-inside* visibility of traffic. With such an approach, `Hades` attempts to target the detection of 'smart' P2P bots. However, such botnets are not known to exist at present[3]. Thus, no network traces corresponding to such behavior could be obtained or evaluated. We plan to further evaluate our system by generating synthetic botnet data which involves inside-to-inside communication over LAN.

`Hades` can be further improved by considering a more elaborate or sophisticated set of host-based features, such as number of ports opened and number of suspi-

[3]With the exception of Stuxnet [Murchu 2010], which we ignore here since it targets SCADA systems, and evaluating its detection with Internet traffic would not be possible.

cious ports opened by a host [Zeng *et al.* 2010], percentage of new IPs contacted by a host [Yen & Reiter 2010], or evaluating whether the traffic on a host is human-driven or automated (bot-driven) [Yen & Reiter 2010].

Further, owing to the limited implementations of Mahout, `Hades` has only been evaluated with random forests in Mahout. As parallelized implementations in Mahout grow, we plan to evaluate `Hades` with other classification algorithms as well.

Chapter 7

Conclusions and future scope of work

This thesis proposed novel mechanisms for intrusion detection in P2P networks. Past research on security and intrusion detection in P2P networks was elucidated in Chapter 2. Limitations of past efforts were also indicated. A significant portion of this thesis dealt with the detection of malicious P2P traffic in the form of P2P botnet traffic (Chapters 3 and 4). A portion of this thesis also proposed game theoretic strategies for deployment of IDS in P2P networks (Chapter 5). A distributed and scalable, Hadoop-based framework for detection of P2P botnets was also proposed (Chapter 6).

7.1 Conclusions and summary of research contributions

In Chapters 3 and 4, we presented our approaches for the detection of P2P botnet traffic in the presence of benign P2P traffic at a network perimeter, by exploiting behavioral differences between P2P bots and benign P2P applications. Our approaches do not rely on DPI or signature-based mechanisms which are eas-

ily defeated by botnets/applications using encryption. They do not assume the availability of any 'seed' information of bots through blacklist of IPs. They aim to detect the *stealthy* behavior of P2P botnets on the basis of their 'P2P' behavior and C&C communications with other bots, while the bots lie dormant in their rally or waiting stages (to evade detection by IDS which look for anomalous communication patterns) or perform malicious activities (spamming, password stealing, etc.) in a manner which is not observable to a network administrator.

In Chapter 3, we presented `PeerShark` with a 'best of both worlds' approach utilizing flow-based approaches as well as conversation-based approaches in a two-tier architecture. `PeerShark` could differentiate between benign P2P traffic and malicious (botnet) P2P traffic, and also detect *unknown* P2P botnets with high accuracy. `PeerShark` begins with the *de facto* standard of 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 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, and these features are used to build supervised machine learning models which can accurately differentiate between benign P2P applications and P2P botnets. Since `PeerShark` could also detect *unknown* P2P botnets (i.e., those not used during the training phase) with high accuracy, this approach is expected to be generic enough to detect new variants of botnets.

The context of P2P botnet detection is adversarial in nature since the models created for their detection are built using 'botnet' data which has been generated by an adversary. Hence, the adversary is in a position to evade the detection mechanisms if he can change the behavior of his bots. Our work in Chapter 4 asks an important question: if the bot-master slightly alters the behavior and

communication patterns of the bots, are these detection models *robust* and *resistant* towards such a change? Our approach addressed the context of detection of P2P bots in the presence of noise injected by an adversary. Our approach utilized conversation-based mechanisms and enhanced them by extracting features based on Fourier Transforms and information entropy. We leveraged on the fact that communication of bots amongst each other follows a certain regularity or periodicity with respect to timing and exchange of data. We extracted two-tuple conversations from network traffic and treated each conversation as a time-series sequence (or a 'signal'). In order to uncover the hidden patterns between the communications of bots, we converted the *time-domain* network communication to the *frequency-domain*. From each conversation, we extracted features based on Fourier transform and information entropy. We used real-world network traces of benign P2P applications and P2P botnets to compare the performance of our features with traditional flow-based features employed by past research (such as [Livadas *et al.* 2006, Saad *et al.* 2011, Kheir & Wolley 2013, Zhang *et al.* 2014]). We built detection models with multiple supervised machine learning algorithms. We injected noise in our test data to demonstrate that our detection approach is more resilient towards variation in data or introduction of noise in the data by an adversary. With our approach, we could detect P2P botnet traffic in the presence of injected noise with True Positive rate as high as 90%.

Our work presented in Chapter 5 studied the problem of securing a super-peer based P2P network from an adversary who may become part of the P2P network by joining from any part of the network. The adversary can attack a super-peer and thus disrupt the functioning of the P2P network. Peers may try to secure the network by running IDS at certain strategically-chosen locations in the network. But, a deterministic schedule of running and positioning the IDS can be observed and thwarted by an adversary. In our work, we propose game theoretic strategies for deployment of IDS in a P2P network. Our approach distributes the

responsibility of running the IDS between the peers in a randomized fashion and minimizes the probability of a successful attack. Past research on different aspects of 'P2P intrusion detection' [Locasto *et al.* 2005, Janakiraman *et al.* 2003, Duma *et al.* 2006] stands to gain from such strategic deployment of IDS.

Although many approaches have been proposed which evaluated the detection of P2P botnets in Internet traffic [François *et al.* 2011] or proposed mechanisms for the detection of P2P botnets in the presence of benign P2P traffic [Rahbarinia *et al.* 2014], building a scalable detection framework has received very little attention in past research (such as in [Zhang *et al.* 2014]). In our approach presented in Chapter 6, we presented our system `Hades`, which utilized the distributed computing power of the Hadoop ecosystem to parse large network traces and extract 'behavioral' features for every P2P host seen in network communications. `Hades` adopts a Host-aggregation based approach which obtains statistical features *per host* for all P2P hosts involved in network communication. The extracted feature-set is then used to train supervised machine learning models which can differentiate P2P botnets from P2P applications. Another novel contribution of this system is a distributed data collection architecture wherein data collectors are placed at multiple locations inside an enterprise network and sit close to the peers, say at a Distribution switch or a Wi-Fi access point. This approach allows *inside-to-inside* communication view, which can be vital for detecting smart P2P botnets inside a network which communicate to each other over LAN in a P2P fashion and limit communications to the outside world via one or two peers only. To the best of our knowledge, `Hades` is the first attempt at distributed data collection for the detection of P2P botnet traffic.

7.2 Future scope of work

The following areas can benefit from further research:

1. A thorough evaluation of the effect of injection of noise in the detection of P2P botnet traffic is required. Statistical and behavioral models need to explore and utilize heuristics or features which are resistant towards changes in communication patterns of bots.

2. Game theoretic approaches – which consider rational, utility-maximizing peers – can benefit from more detailed modeling of the players (namely the attacker and the defender) and the payoffs. Considering other forms of games such as non-zero-sum games will bring up new challenges in terms of identifying the equilibrium solution. Future work also needs to consider solution concepts beyond the Nash equilibrium.

3. Distributed and scalable frameworks for malicious (botnet) P2P traffic need to be further improved to incorporate information from network communication (in the form of flows/conversations) as well as host-level information. Integrating these approaches with a distributed data collection approach can strengthen the detection of malicious activities.

References

[Anceaume *et al.* 2005] Emmanuelle Anceaume, Maria Gradinariu and Aina Ravoaja. *Incentives for p2p fair resource sharing*. In Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on, pages 253–260. IEEE, 2005. 43, 110

[Andriesse *et al.* 2013] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann and Herbert Bos. *Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus*. In Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on, pages 116–123. IEEE, 2013. 10

[Androutsellis-Theotokis & Spinellis 2004] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. *A survey of peer-to-peer content distribution technologies*. ACM Computing Surveys (CSUR), vol. 36, no. 4, pages 335–371, 2004. 2

[Bauer *et al.* 2007] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno and Douglas Sicker. *Low-resource Routing Attacks Against Tor*. In Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society, WPES '07, pages 11–20, New York, NY, USA, 2007. ACM. 106

[Bedrosian 2013] Brian Bedrosian. *How Peer-to-Peer Networking and Wi-Fi Direct Will Help Drive the Wearable "Internet of Things"*. http://www.broadcom.com/blog/wireless-technology/how-peer-to-peer-networking-and-wi-fi-direct-will-help-drive-the-wearable-internet-of-things/, 2013. Accessed on 6th August 2015. 13

[Bensoussan *et al.* 2010] Alain Bensoussan, Murat Kantarcioglu and SingRu Celine Hoe. *A game-theoretical approach for finding optimal strategies in a botnet defense model*. In Decision and Game Theory for Security, pages 135–148. Springer, 2010. 45

[Berket *et al.* 2004] Karlo Berket, Abdelilah Essiari and Artur Muratas. *PKI-based security for peer-to-peer information sharing*. In Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, pages 45–52. IEEE, 2004. 32

[Bialecki *et al.* 2005] Andrzej Bialecki, Michael Cafarella, Doug Cutting and Owen O MALLEY. *Hadoop: a framework for running applications on large clusters built of commodity hardware*. Wiki at http://lucene.apache.org/hadoop, vol. 11, 2005. 131

[Borisov 2006] Nikita Borisov. *Computational puzzles as sybil defenses*. In Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on, pages 171–176. IEEE, 2006. 29

[Borthakur 2011] Dhruba Borthakur. *The hadoop distributed file system: Architecture and design, 2007*. Apache Software Foundation, 2011. 133

[Bouckaert 2008] Remco R Bouckaert. *Bayesian network classifiers in weka for version 3-5-7*. Artificial Intelligence Tools, vol. 11, no. 3, pages 369–387, 2008. 97

[Breiman 2001] Leo Breiman. *Random forests*. Machine learning, vol. 45, no. 1, pages 5–32, 2001. 66

[Buford *et al.* 2008] John Buford, Heather Yu and Eng Keong Lua. P2p networking and applications. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. 2, 3, 4, 9

[Casenove & Miraglia 2014] Matteo Casenove and Armando Miraglia. *Botnet over Tor: The illusion of hiding*. In Cyber Conflict (CyCon 2014), 2014 6th International Conference On, pages 273–282. IEEE, 2014. 103

[Castro *et al.* 2002] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron and Dan S Wallach. *Secure routing for structured peer-to-peer overlay networks*. ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pages 299–314, 2002. 31

[Chawla 2003] Nitesh V Chawla. *C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure*. In Proceedings of the ICML, volume 3, 2003. 66

[Chen & Leneutre 2009] Lin Chen and Jean Leneutre. *A game theoretical framework on intrusion detection in heterogeneous networks*. Information Forensics and Security, IEEE Transactions on, vol. 4, no. 2, pages 165–178, 2009. 46

[Cisco 2014] Cisco. *2014 Annual Security Report*. http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf, 2014. Accessed on 11th August 2015. 57, 64

[Clarke *et al.* 2001] Ian Clarke, Oskar Sandberg, Brandon Wiley and Theodore W Hong. *Freenet: A distributed anonymous information storage and retrieval system*. In Designing Privacy Enhancing Technologies, pages 46–66. Springer, 2001. 4

[Condie *et al.* 2006] Tyson Condie, Varun Kacholia, Sriram Sank, Joseph M Hellerstein and Petros Maniatis. *Induced Churn as Shelter from Routing-Table Poisoning*. In NDSS, 2006. 29

[Coskun *et al.* 2010] Baris Coskun, Sven Dietrich and Nasir Memon. *Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts*. In Proceedings of the 26th Annual Computer Security Applications Conference, pages 131–140. ACM, 2010. 39

[Dash & Liu 2003] Manoranjan Dash and Huan Liu. *Consistency-based search in feature selection*. Artificial intelligence, vol. 151, no. 1, pages 155–176, 2003. 21, 91

[Daswani & Garcia-Molina 2002] Neil Daswani and Hector Garcia-Molina. *Query-flood DoS Attacks in Gnutella*. In Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS '02, pages 181–192, New York, NY, USA, 2002. ACM. 33, 106

[Dhungel *et al.* 2007] Prithula Dhungel, Xiaojun Hei, Keith W Ross and Nitesh Saxena. *The pollution attack in P2P live video streaming: measurement results and defenses*. In Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV, pages 323–328. ACM, 2007. 33

[Douceur 2002] John R Douceur. *The sybil attack*. In Peer-to-peer Systems, pages 251–260. Springer, 2002. 28

[Drinkwater 2014] Doug Drinkwater. *Gameover trojan rises from the dead*. http://www.scmagazineuk.com/gameover-trojan-rises-from-the-dead/article/357964/, 2014. Accessed on 20th January 2015. 11, 71

[Drummond *et al.* 2003] Chris Drummond, Robert C Holte*et al. C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling*. In Workshop on learning from imbalanced datasets II, volume 11, 2003. 66

[Duma *et al.* 2006] Claudiu Duma, Martin Karresand, Nahid Shahmehri and Germano Caronni. *A trust-aware, p2p-based overlay for intrusion detection*. In Database and Expert Systems Applications, 2006. DEXA'06. 17th International Workshop on, pages 692–697. IEEE, 2006. 47, 48, 145

[Falliere 2011] Nicolas Falliere. *Sality: Story of a peer-to-peer viral network*. Rapport technique, Symantec Corporation, 2011. 32

[Feldman *et al.* 2006] Michal Feldman, Christos Papadimitriou, John Chuang and Ion Stoica. *Free-riding and whitewashing in peer-to-peer systems*. Selected Areas in Communications, IEEE Journal on, vol. 24, no. 5, pages 1010–1019, 2006. 44

[Fisher 2007] Dennis Fisher. *Storm, Nugache lead dangerous new botnet barrage*. http://searchsecurity.techtarget.com/news/1286808/Storm-Nugache-lead-dangerous-new-botnet-barrage, 2007. Accessed on 20th July 2014. 68

[Fisher 2013] Dennis Fisher. *88 percent of Citadel botnets down*. http://threatpost.com/microsoft-88-percent-of-citadel-botnets-down/101503, 2013. Accessed on 9th June 2015. 11

[François *et al.* 2011] Jérôme François, Shaonan Wang, Radu State and Thomas 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, pages 1–14. Springer-Verlag, Berlin, Heidelberg, 2011. 39, 128, 145

[Friedman *et al.* 1997] Nir Friedman, Dan Geiger and Moises Goldszmidt. *Bayesian network classifiers*. Machine learning, vol. 29, no. 2-3, pages 131–163, 1997. 66

[Gatti *et al.* 2004] Rupert Gatti, Stephen Lewis, Andy Ozment, Thierry Rayna and Andrei Serjantov. *Sufficiently Secure Peer-to-Peer Networks*. In Proceedings of the Third Workshop on Economics and Information Security, 2004. 43

[Gillet 2013] Mark Gillet. *Skype's cloud-based architecture*. http://blogs.skype.com/2013/10/04/skype-architecture-update/, 2013. Accessed on 7th November 2014. 1

[Greene 2010] Tim 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, 2010. Accessed on 9th June 2013. 12

[Gu *et al.* 2008a] Guofei Gu, Roberto Perdisci, Junjie Zhang and Wenke Lee. *BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection*. In Proceedings of the 17th Conference on Security Symposium, SS'08, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association. 35, 39

[Gu *et al.* 2008b] Guofei Gu, Junjie Zhang and Wenke Lee. *Botsniffer: Detecting botnet command and control channels in network traffic*. In Proceedings of the 15th Annual Network and Distributed System Security Symposium, NDSS, 2008. 39

[Gupta & Somani 2005] Rohit Gupta and Arun K Somani. *Game theory as a tool to strategize as well as predict nodes' behavior in peer-to-peer networks*. In Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on, volume 1, pages 244–249. IEEE, 2005. 44

[Guyon & Elisseeff 2006] Isabelle Guyon and André Elisseeff. *An introduction to feature extraction*. In Feature extraction, pages 1–25. Springer, 2006. 18

[Hall *et al.* 2009] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann and Ian H Witten. *The WEKA data mining software: an update*. ACM SIGKDD Explorations Newsletter, vol. 11, no. 1, pages 10–18, 2009. 66, 96

[Hall 1999] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999. 19, 20, 91

[Hang *et al.* 2013] Huy Hang, Xuetao Wei, M. Faloutsos and T. Eliassi-Rad. *Entelecheia: Detecting P2P botnets in their waiting stage*. In IFIP Networking Conference, 2013, pages 1–9, USA, May 2013. IEEE. 38, 40

[Haribabu *et al.* 2010] K Haribabu, Dushyant Arora, Bhavik Kothari and Chittaranjan Hota. *Detecting Sybils in Peer-to-Peer Overlays using Neural Networks and CAPTCHAs*. In Computational Intelligence and Communication Networks (CICN), 2010 International Conference on, pages 154–161. IEEE, 2010. 29

[Haribabu *et al.* 2012] K Haribabu, Chittaranjan Hota and Arindam Paul. *GAUR: a method to detect Sybil groups in peer–to–peer overlays*. International Journal of Grid and Utility Computing, vol. 3, no. 2, pages 145–156, 2012. 29

[Iliofotou *et al.* 2009] Marios Iliofotou, Hyun-chul Kim, Michalis Faloutsos, Michael Mitzenmacher, Prashanth Pappu and George Varghese. *Graph-based P2P Traffic Classification at the Internet Backbone*. In Proceedings of the 28th IEEE International Conference on Computer Communications Workshops, INFOCOM'09, pages 37–42, Piscataway, NJ, USA, 2009. IEEE Press. 35

[Ipoque 2008] Ipoque. *Ipoque Internet study 2008/2009*. http://www.ipoque.com/en/resources/internet-studies, 2008. Accessed on 4 January 2014. 1

[Janakiraman *et al.* 2003] Ramaprabhu Janakiraman, Marcel Waldvogel and Qi Zhang. *Indra: A peer-to-peer approach to network intrusion detection and prevention*. In Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on, pages 226–231. IEEE, 2003. 47, 145

[Kamvar *et al.* 2003] Sepandar D Kamvar, Mario T Schlosser and Hector Garcia-Molina. *The eigentrust algorithm for reputation management in p2p networks*. In Proceedings of the 12th international conference on World Wide Web, pages 640–651. ACM, 2003. 43

[Kang & Zhang 2009] Jian Kang and Jun-Yao Zhang. *Application entropy theory to detect new peer-to-peer botnet with multi-chart CUSUM*. In Electronic Commerce and Security, 2009. ISECS'09. Second International Symposium on, volume 1, pages 470–474. IEEE, 2009. 42

[Kanich *et al.* 2011] Chris Kanich, Nicholas Weavery, Damon McCoy, Tristan Halvorson, Christian Kreibichy, Kirill Levchenko, Vern Paxson, Geoffrey M. Voelker and Stefan Savage. *Show Me the Money: Characterizing Spam-advertised Revenue*. In Proceedings of the 20th USENIX Conference on Security, SEC'11, pages 15–15, Berkeley, CA, USA, 2011. USENIX Association. 9

[Karagiannis *et al.* 2004] Thomas Karagiannis, Andre Broido, Michalis Faloutsos and Kc claffy. *Transport Layer Identification of P2P Traffic*. In Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, IMC '04, pages 121–134, New York, NY, USA, 2004. ACM. 38, 52

[Karagiannis *et al.* 2005] Thomas Karagiannis, Konstantina Papagiannaki and Michalis Faloutsos. *BLINC: multilevel traffic classification in the dark*. In ACM

SIGCOMM Computer Communication Review, volume 35, pages 229–240. ACM, 2005. 38, 52

[Keynes 2013] John Maynard Keynes. A treatise on probability. Courier Dover Publications, 2013. 117

[Kheir & Wolley 2013] Nizar Kheir and Chirine Wolley. *Botsuer: Suing stealthy p2p bots in network traffic through netflow analysis*. In Cryptology and Network Security, pages 162–178. Springer, 2013. 37, 39, 41, 75, 93, 96, 144

[Kheir *et al.* 2014] Nizar Kheir, Xiao Han and Chirine Wolley. *Behavioral fine-grained detection and classification of P2P bots*. Journal of Computer Virology and Hacking Techniques, pages 1–17, 2014. 93, 96

[Kim & Médard 2004] Minkyu Kim and Muriel Médard. *Robustness in large-scale random networks*. In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 4, pages 2364–2373. IEEE, 2004. 4

[Kodialam & Lakshman 2003] Murali Kodialam and TV Lakshman. *Detecting network intrusions via sampling: a game theoretic approach*. In INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, volume 3, pages 1880–1889. IEEE, 2003. 47

[Kohavi 1996] Ron Kohavi. *Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid*. In Second International Conference on Knowledge Discovery and Data Mining, pages 202–207, 1996. 97

[Korzhyk *et al.* 2011] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer and Milind Tambe. *Stackelberg vs. Nash in Security Games: An Extended Investigation of Interchangeability, Equivalence, and Uniqueness.* J. Artif. Intell. Res.(JAIR), vol. 41, pages 297–327, 2011. 109

[Lelli 2011] Andrea Lelli. *Waledac botnet back on rise*. http://www.symantec.com/connect/blogs/return-dead-waledacstorm-botnet-back-rise, 2011. Accessed on 1st February 2014. 12

[Letchford 2013] Joshua Letchford. *Computational aspects of stackelberg games*. PhD thesis, Duke University, 2013. 24

[Leyden 2014] John 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/, 2014. Accessed on 30th June 2015. 13

[Li *et al.* 2008] Jun Li, Shunyi Zhang, Yanqing Lu and Junrong Yan. *Real-time P2P traffic identification*. In Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008, pages 1–5, USA, 2008. IEEE. 35, 101

[Li *et al.* 2013] Liyun Li, Suhas Mathur and Baris Coskun. *Gangs of the internet: Towards automatic discovery of peer-to-peer communities*. In Communications and Network Security (CNS), 2013 IEEE Conference on, pages 64–72, USA, 2013. IEEE. 38, 40

[Liang *et al.* 2005] Jian Liang, Rakesh Kumar, Yongjian Xi and Keith W Ross. *Pollution in P2P file sharing systems*. In INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, volume 2, pages 1174–1185. IEEE, 2005. 33

[Liang *et al.* 2006] Jian Liang, Naoum Naoumov and Keith W Ross. *The Index Poisoning Attack in P2P File Sharing Systems.* In INFOCOM, pages 1–12. Citeseer, 2006. 32

[Livadas *et al.* 2006] Carl Livadas, Robert Walsh, David Lapsley and W Timothy Strayer. *Using machine learning techniques to identify botnet traffic*. In Local Computer Networks, Proceedings 2006 31st IEEE Conference on, pages 967–974. IEEE, 2006. 35, 41, 75, 93, 96, 144

[Locasto *et al.* 2005] Michael E Locasto, Janak J Parekh, Angelos D Keromytis and Salvatore J Stolfo. *Towards collaborative security and p2p intrusion detection*. In Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC, pages 333–339. IEEE, 2005. 47, 145

[Locher *et al.* 2010] Thomas Locher, David Mysicka, Stefan Schmid and Roger Wattenhofer. *Poisoning the Kad network*. In Distributed Computing and Networking, pages 195–206. Springer, 2010. 28

[Lua *et al.* 2005] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma and Steven Lim. *A survey and comparison of peer-to-peer overlay network schemes*. IEEE Communications Surveys and Tutorials, vol. 7, no. 2, pages 72–93, 2005. 3, 4

[Ma *et al.* 2006] Richard TB Ma, Sam Lee, John Lui and David KY Yau. *Incentive and service differentiation in P2P networks: a game theoretic approach*. IEEE/ACM Transactions on Networking (TON), vol. 14, no. 5, pages 978–991, 2006. 43, 106

[MacKay 2003] David JC MacKay. *Information Theory, Inference & Learning Algorithms*. 2003. 85

[Mahout 2012] Apache Mahout. *Mahout: Scalable machine-learning and data-mining library*. http://mahout.apache.org, 2012. 132

[Manshaei *et al.* 2013] Mohammad Hossein Manshaei, Quanyan Zhu, Tansu Alpcan, Tamer Bacşar and Jean-Pierre Hubaux. *Game theory meets network security and privacy*. ACM Computing Surveys (CSUR), vol. 45, no. 3, page 25, 2013. 43

[Masud *et al.* 2008] Mohammad M Masud, Jing Gao, Latifur Khan, Jiawei Han and Bhavani Thuraisingham. *Mining concept-drifting data stream to detect peer to peer botnet traffic*. Univ. of Texas at Dallas Technical Report# UTDCS-05- 08, 2008. 56

[Microsoft 2010] Microsoft. *Microsoft Security Intelligence Report, Volume 9, January-June 2010*. http://www.microsoft.com/security/sir/, 2010. Accessed on 1st February 2014. 9

[Mimoso 2015] Michael Mimoso. *Dridex trojan*. http://threatpost.com/dridex-banking-trojan-spreading-via-office-macros/110255, 2015. Accessed on 19th January 2015. 11

[Mitchell 1997] Thomas M. Mitchell. Machine learning. McGraw-Hill, Inc., New York, NY, USA, 1 édition, 1997. 17

[Mohri *et al.* 2012] Mehryar Mohri, Afshin Rostamizadeh and Ameet Talwalkar. Foundations of machine learning. MIT press, 2012. 17

[Moon 1996] Todd K Moon. *The expectation-maximization algorithm*. Signal processing magazine, IEEE, vol. 13, no. 6, pages 47–60, 1996. 71

[Moscibroda *et al.* 2006] Thomas Moscibroda, Stefan Schmid and Roger Wattenhofer. *When selfish meets evil: Byzantine players in a virus inoculation game*. In Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing, pages 35–44. ACM, 2006. 44

[Motwani & Raghavan 2010] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. Chapman & Hall/CRC, 2010. 24, 116

[Murchu 2010] Liam O Murchu. *Stuxnet P2P component*. http://www.symantec.com/connect/blogs/stuxnet-p2p-component, 2010. Accessed on 12th February 2014. 12, 140

[Murdoch & Danezis 2005] Steven J. Murdoch and George Danezis. *Low-Cost Traffic Analysis of Tor*. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, SP '05, pages 183–195, Washington, DC, USA, 2005. IEEE Computer Society. 106

[Myerson 1999] Roger B Myerson. *Nash equilibrium and the history of economic theory*. Journal of Economic Literature, pages 1067–1082, 1999. 22

[Myerson 2013] Roger B Myerson. Game theory. Harvard university press, 2013. 22

[Nagaraja *et al.* 2010] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar and Nikita Borisov. *BotGrep: Finding P2P Bots with Structured Graph Analysis.* In USENIX Security Symposium, pages 95–110, 2010. 39

[Nagaraja 2014] Shishir Nagaraja. *Botyacc: Unified P2P Botnet Detection Using Behavioural Analysis and Graph Analysis*. In Computer Security-ESORICS 2014, pages 439–456. Springer, 2014. 39

[Nappa *et al.* 2010] Antonio Nappa, Aristide Fattori, Marco Balduzzi, Matteo Dell'Amico and Lorenzo Cavallaro. *Take a deep breath: a stealthy, resilient and cost-effective botnet using skype*. In Detection of Intrusions and Malware, and Vulnerability Assessment, pages 81–100. Springer-Verlag, Berlin, Heidelberg, 2010. 69

[Narang & Hota 2015] Pratik Narang and Chittaranjan Hota. *Game-theoretic strategies for IDS deployment in peer-to-peer networks*. Information Systems Frontiers, vol. 17, no. 5, pages 1017–1028, 2015. 107

[Narang *et al.* 2013] Pratik Narang, Jagan Mohan Reddy and Chittaranjan Hota. *Feature Selection for Detection of Peer-to-peer Botnet Traffic*. In Proceedings of the 6th ACM India Computing Convention, Compute '13, pages 16:1–16:9. ACM, 2013. 36, 37, 49, 52

[Narang *et al.* 2014a] Pratik Narang, Chittaranjan Hota and VN Venkatakrishnan. *PeerShark: flow-clustering and conversation-generation for malicious peer-to-peer traffic identification*. EURASIP Journal on Information Security, vol. 2014, no. 1, pages 1–12, 2014. 51

[Narang *et al.* 2014b] Pratik Narang, Vansh Khurana and Chittaranjan Hota. *Machine-learning approaches for P2P botnet detection using signal-processing*

techniques. In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, pages 338–341. ACM, 2014. 76

[Narang *et al.* 2014c] Pratik Narang, Kunal Mehta and Chittaranjan Hota. *Game-theoretic Patrolling Strategies for Intrusion Detection in Collaborative Peer-to-Peer Networks*. In International Conference on Secure Knowledge Management in Big-data era, 2014. 115

[Narang *et al.* 2014d] Pratik Narang, Subhajit Ray, Chittaranjan Hota and VN Venkatakrishnan. *PeerShark: Detecting Peer-to-Peer Botnets by Tracking Conversations*. In Security and Privacy Workshops (SPW), 2014 IEEE, pages 108–115. IEEE, May 2014. 70

[Narang *et al.* 2014e] Pratik Narang, Abhishek Thakur and Chittaranjan Hota. *Hades: a Hadoop-based framework for detection of peer-to-peer botnets*. In Proceedings of the 20th International Conference on Management of Data, pages 121–124. Computer Society of India, 2014. 130

[Nash *et al.* 1950] John F Nash*et al. Equilibrium points in n-person games*. Proceedings of the national academy of sciences, vol. 36, no. 1, pages 48–49, 1950. 23

[Noh *et al.* 2009] Sang-Kyun Noh, Joo-Hyung Oh, Jae-Seo Lee, Bong-Nam Noh and Hyun-Cheol Jeong. *Detecting P2P botnets using a multi-phased flow model*. In Digital Society, 2009. ICDS'09. Third International Conference on, pages 247–253. IEEE, 2009. 36

[Oppenheim *et al.* 1997] Alan V Oppenheim, Alan S Willsky and S Nawab. Signals and systems. 1997. Prentice Hall, 1997. 76

[Osborne & Rubinstein 1994] Martin J Osborne and Ariel Rubinstein. A course in game theory. MIT press, 1994. 23

[Pelleg & Moore 2000] Dan Pelleg and Andrew W. 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, pages 727–734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. 60

[Puttaswamy *et al.* 2009] Krishna PN Puttaswamy, Haitao Zheng and Ben Y Zhao. *Securing structured overlays against identity attacks*. Parallel and Distributed Systems, IEEE Transactions on, vol. 20, no. 10, pages 1487–1498, 2009. 31

[Rahbarinia *et al.* 2013] Babak Rahbarinia, Roberto Perdisci, Andrea Lanzi and Kang Li. *Peerrush: Mining for unwanted p2p traffic*. In Detection of Intrusions and Malware, and Vulnerability Assessment, pages 62–82. Springer-Verlag, Berlin, Heidelberg, 2013. 57, 90

[Rahbarinia *et al.* 2014] Babak Rahbarinia, Roberto Perdisci, Andrea Lanzi and Kang Li. *PeerRush: Mining for unwanted P2P traffic*. Journal of Information Security and Applications, vol. 19, no. 3, pages 194 – 208, 2014. 36, 39, 52, 53, 56, 65, 71, 78, 84, 89, 101, 103, 128, 137, 145

[Ratnasamy *et al.* 2001] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Shenker. A scalable content-addressable network, volume 31. ACM, 2001. 5, 29

[Reynolds 2009] Douglas Reynolds. *Gaussian Mixture Models*. In Encyclopedia of Biometrics, pages 659–663. Springer US, USA, 2009. 71

[Rossow *et al.* 2013] Christian Rossow, Dennis Andriesse, Tillmann Werner, Brett Stone-Gross, Daniel Plohmann, Christian J Dietrich and Herbert Bos. *SoK: P2PWNED-Modeling and Evaluating the Resilience of Peer-to-Peer Botnets*. In Security and Privacy (SP), 2013 IEEE Symposium on, pages 97–111. IEEE, 2013. 10, 34

[Rowstron & Druschel 2001] Antony Rowstron and Peter Druschel. *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems*. In Middleware 2001, pages 329–350. Springer, 2001. 5, 29

[Saad *et al.* 2011] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix and Payman Hakimian. *Detecting P2P botnets through network behavior analysis and machine learning*. In Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on, pages 174–180. IEEE, 2011. 36, 41, 49, 75, 93, 96, 144

[Sandvine 2014] Sandvine. *Sandvine Global Internet Phenomena Report 2013*. https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf/, 2014. Accessed on 4th July 2015. 2, 71

[Scarfone & Mell 2007] Karen Scarfone and Peter Mell. *Guide to intrusion detection and prevention systems (idps)*. NIST special publication, vol. 800-94, no. 2007, page 127, 2007. 14

[Schoof & Koning 2007] Reinier Schoof and Ralph Koning. *Detecting peer-to-peer botnets*. University of Amsterdam, 2007. Technical report. 34

[Segura 2012] Jerome Segura. *Citadel: a cyber-criminal's ultimate weapon?* https://blog.malwarebytes.org/intelligence/2012/11/citadel-a-cyber-criminals-ultimate-weapon/, 2012. Accessed on 1st July 2015. 11

[Sen *et al.* 2004] Subhabrata Sen, Oliver Spatscheck and Dongmei 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, pages 512–521, New York, NY, USA, 2004. ACM. 35

[Singh *et al.* 2006] Atul Singh*et al. Eclipse attacks on overlay networks: Threats and defenses*. In In IEEE INFOCOM. Citeseer, 2006. 28

[Singh *et al.* 2014] Kamaldeep Singh, Sharath Chandra Guntuku, Abhishek Thakur and Chittaranjan Hota. *Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests*. Information Sciences, vol. 278, pages 488–497, 2014. 35, 36, 37, 49, 96, 101, 103

[Sit & Morris 2002] Emil Sit and Robert Morris. *Security considerations for peer-to-peer distributed hash tables*. In Peer-to-Peer Systems, pages 261–269. Springer, 2002. 5

[Steiner *et al.* 2007] Moritz Steiner, Taoufik En-Najjary and Ernst W Biersack. *Exploiting KAD: possible uses and misuses*. ACM SIGCOMM Computer Communication Review, vol. 37, no. 5, pages 65–70, 2007. 30, 31

[Stewart 2007] Joe Stewart. *Storm worm DDoS attack*. http://www.secureworks.com/cyber-threat-intelligence/threats/storm-worm/, 2007. Accessed on 1st February 2014. 12

[Stoica *et al.* 2001] I Stoica, R Morris, D Liben-Nowell, DR Karger, MF Kaashoek, F Dabek and H Balakrishnan. *Chord: A scalable P2P lookup protocol for Internet applications*. In Proc. of ACM SIGCOMM, 2001. 5

[Stover *et al.* 2007] Sam Stover, Dave Dittrich, John Hernandez and Sven Dietrich. *Analysis of the Storm and Nugache Trojans: P2P is here*. USENIX; login, vol. 32, no. 6, pages 18–27, 2007. 32, 68

[Tegeler *et al.* 2012] Florian Tegeler, Xiaoming Fu, Giovanni Vigna and Christopher Kruegel. *Botfinder: Finding bots in network traffic without deep packet inspection*. In Proceedings of the 8th international conference on Emerging

networking experiments and technologies, pages 349–360. ACM, 2012. 41, 42, 76, 84, 87

[Theodorakopoulos & Baras 2007] George Theodorakopoulos and John S Baras. *Malicious users in unstructured networks*. In INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE, pages 884–891. IEEE, 2007. 44

[Theodorakopoulos & Baras 2008] George Theodorakopoulos and J Baras. *Game theoretic modeling of malicious users in collaborative networks*. Selected Areas in Communications, IEEE Journal on, vol. 26, no. 7, pages 1317–1327, 2008. 44

[Thonnard *et al.* 2012] Olivier Thonnard, Leyla Bilge, Gavin O'Gorman, Seán Kiernan and Martin Lee. *Industrial espionage and targeted attacks: Understanding the characteristics of an escalating threat*. In Research in attacks, intrusions, and defenses, pages 64–85. Springer, 2012.

[Thusoo *et al.* 2009] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff and Raghotham Murthy. *Hive: a warehousing solution over a map-reduce framework*. Proceedings of the VLDB Endowment, vol. 2, no. 2, pages 1626–1629, 2009. 132, 134

[Trifa & Khemakhem 2012] Zied Trifa and Maher Khemakhem. *Taxonomy of structured P2P overlay networks security attacks*. World Academy of Science, Engineering and Technology, vol. 6, no. 4, pages 460–466, 2012. 28

[Van Der Putten & Van Someren 2004] Peter Van Der Putten and Maarten Van Someren. *A bias-variance analysis of a real world learning problem: The CoIL challenge 2000*. Machine Learning, vol. 57, no. 1-2, pages 177–195, 2004. 19

[Vaněk *et al.* 2012] Ondřej Vaněk, Zhengyu Yin, Manish Jain, Branislav Bošanskỳ, Milind Tambe and Michal Pěchouček. *Game-theoretic resource allocation for malicious packet detection in computer networks*. In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, pages 905–912. International Foundation for Autonomous Agents and Multiagent Systems, 2012. 47

[Vishnumurthy & Francis 2004] Vivek Vishnumurthy and Paul Francis. *On random node selection in p2p and overlay networks*. Manuscript, http://www. cs.

cornell. edu/People/francis/RandSelection19. pdf, 2004. 4

[Williams *et al.* 2006] Nigel Williams, Sebastian Zander and Grenville Armitage. *A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification*. ACM SIGCOMM Computer Communication Review, vol. 36, no. 5, pages 5–16, 2006. 101

[Wolpert 1992] David H Wolpert. *Stacked generalization*. Neural networks, vol. 5, no. 2, pages 241–259, 1992. 97

[Wormald 1999] Nicholas C Wormald. *Models of random regular graphs*. London Mathematical Society Lecture Note Series, pages 239–298, 1999. 4

[Ye *et al.* 2004] Song Ye, Fillia Makedon and James Ford. *Collaborative automated trust negotiation in peer-to-peer systems*. In Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, pages 108–115. IEEE, 2004. 43, 110

[Yen & Reiter 2010] Ting-Fang Yen and Michael K. 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, pages 241–252. IEEE, 2010. 36, 42, 141

[Yu *et al.* 2006] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons and Abraham Flaxman. *Sybilguard: defending against sybil attacks via social networks*. ACM SIGCOMM Computer Communication Review, vol. 36, no. 4, pages 267–278, 2006. 29

[Yu *et al.* 2010] Xiaocong Yu, Xiaomei Dong, Ge Yu, Yuhai Qin, Dejun Yue and Yan Zhao. *Online Botnet Detection Based on Incremental Discrete Fourier Transform.* Journal of Networks, vol. 5, no. 5, 2010. 42

[Yue *et al.* 2009] Xiaowen Yue, Xiaofeng Qiu, Yang Ji and Chunhong Zhang. *P2P attack taxonomy and relationship analysis*. In Advanced Communication Technology, 2009. ICACT 2009. 11th International Conference on, volume 2, pages 1207–1210. IEEE, 2009. 28

[Zeng *et al.* 2010] Yuanyuan Zeng, Xin Hu and Kang G Shin. *Detection of botnets using combined host-and network-level information*. In Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on, pages 291–300. IEEE, 2010. 42, 141

[Zhang *et al.* 2010] Ying Zhang, Hongbo Wang and Shiduan Cheng. *A method for real-time peer-to-peer traffic classification based on C4. 5*. In Communication Technology (ICCT), 2010 12th IEEE International Conference on, pages 1192–1195. IEEE, 2010. 101

[Zhang *et al.* 2011a] Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz and Xiapu 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, pages 121–132, Washington, DC, USA, 2011. IEEE Computer Society. 36, 37, 51

[Zhang *et al.* 2011b] Ren Zhang, Jianyu Zhang, Yu Chen, Nanhao Qin, Bingshuang Liu and Yuan Zhang. *Making eclipse attacks computationally infeasible in large-scale DHTs*. In Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International, pages 1–8. IEEE, 2011. 31

[Zhang *et al.* 2014] Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo and Unum Sarfraz. *Building a Scalable System for Stealthy P2P-Botnet Detection*. Information Forensics and Security, IEEE Transactions on, vol. 9, no. 1, pages 27–38, 2014. 35, 36, 37, 39, 41, 51, 75, 93, 128, 144, 145

[Zhang 2013] Shaojun Zhang. Conversation-based p2p botnet detection with decision fusion. Master's thesis, Fredericton: University of New Brunswick, 2013. 39

[Zhu & Fu 2011] Ye Zhu and Huirong Fu. *Traffic analysis attacks on Skype VoIP calls*. Computer Communications, vol. 34, no. 10, pages 1202–1212, 2011. 106

Publications

Journal Papers

1. Narang, P., Reddy, J. M., & Hota, C. IDPT: Integrated Detection of P2P Threats in Parallel. *EURASIP Journal on Information Security*, Springer. Under review since 24th September 2015.
2. Narang, P., Hota, C., & Sencar, H. T. Noise-resistant Mechanisms for the Detection of Stealthy Peer-to-Peer Botnets. *Computer Communications*, Elsevier. Under review since 2nd April 2015.
3. Narang, P. & Hota, C. (2015). Game-theoretic Strategies for IDS Deployment in Peer-to-Peer Networks. *Information Systems Frontiers*, Springer, 2015, 17(5), 1017-1028.
4. Narang, P., Hota, C., & Venkatakrishnan, V. N. (2014). PeerShark: flow-clustering and conversation-generation for malicious peer-to-peer traffic identification. *EURASIP Journal on Information Security,* Springer, 2014(1), 1-12.

Conference Papers

1. Narang, P., Thakur, A., & Hota, C. (2014, December). Hades: A Hadoop-based framework for detection of peer-to-peer botnets. In *Proceedings of the 20th International Conference on Management of Data* (pp. 121-124). Computer Society of India.
2. Narang, P., Mehta, K., & Hota, C. (2014, December). Game-theoretic Patrolling Strategies for Intrusion Detection in Collaborative Peer-to-Peer Networks. In *International Conference on Secure Knowledge Management in Big-data era*, 2014. DOI: 10.13140/2.1.2533.2804
3. Narang, P., Khurana, V., & Hota, C. (2014, May). Machine-learning approaches for P2P botnet detection using signal-processing techniques. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems* (pp. 338-341). ACM.

4. Narang, P., Ray, S., Hota, C., & Venkatakrishnan, V.N. (2014, May). Peershark: Detecting peer-to-peer botnets by tracking conversations. In *Security and Privacy Workshops (SPW)*, 2014 IEEE (pp. 108-115). IEEE.

5. Narang, P., Reddy, J. M., & Hota, C. (2013, August). Feature selection for detection of peer-to-peer botnet traffic. In *Proceedings of the 6th ACM India Computing Convention* (pp. 16:1–9). ACM.

Biographies

Brief Biography of the Candidate

Pratik Narang is a Ph.D scholar and research assistant at the Computer Science department of Birla Institute of Technology and Science – Pilani, Hyderabad campus. Prior to pursuing PhD, he completed his M.Sc.(Tech) in Information Systems from BITS Pilani in 2011. His research interests lie in the area of network security and cyber-security with applications from machine learning and game theory, and his PhD is funded by grants from Department of Electronics & Information Technology (DeitY), Government of India. He has held a visiting position with New York University, Abu Dhabi campus. He has served as a sub-reviewer for international conferences IEEE WIFS 2015 and IEEE CNS 2014, and a reviewer for IEEE Transactions on Dependable and Secure Computing, Journal of Machine Learning and Cybernetics (Springer), Information Systems Frontiers (Springer) and EURASIP Journal on Information Security (Springer).

Brief Biography of the Supervisor

Chittaranjan Hota is a Professor and Associate Dean at Birla Institute of Technology and Science – Pilani, Hyderabad Campus, Hyderabad, India. He was the founding Head of Dept. of Computer Science at BITS, Hyderabad. Prof. Hota did his PhD in Computer Science and Engineering from Birla Institute of Technology & Science, Pilani. He has been a visiting researcher and visiting professor at University of New South Wales, Sydney; University of Cagliari, Italy; Aalto University, Finland and City University, London over the past few years. His research work has been funded by University Grants Commission (UGC), New Delhi; Department of Electronics & Information Technology (DeitY), New Delhi; and Tata Consultancy Services, India. He has guided PhD students and currently guiding several in the areas of Overlay networks, Information Security, and Distributed computing. He is recipient of Australian Vice Chancellors' Committee award, recipient of Erasmus Mundus fellowship from European commission, and recipient of Certificate of Excellence from Kris Ramachandran Faculty Excellence Award from BITS Pilani. He has published extensively in peer-reviewed journals and

conferences and has also edited LNCS volumes. He is a member of IEEE, ACM, IE, and ISTE. His research interests are in the areas of Traffic Engineering in IP Networks, Security and Quality of Service issues over the Internet, Peer-to-Peer Overlay Security, Mobile Wireless Networks and Internet of Things.

Brief Biography of the Co-Supervisor

V.N. "Venkat" Venkatakrishnan's (http://www.cs.uic.edu/ venkat) broad research interests are in computer security and privacy. He is particularly interested in the security of software systems, in vulnerability analysis and automated approaches for preventing large scale attacks on computer systems. His research work derives from techniques rooted in programming languages and compilers, operating systems, software engineering and formal methods to address practical problems in computer security. Specific areas within software security that his work has touched upon in the recent past include web application security, safe execution of untrusted third party content, web malware analysis and analysis / verification of systems software. He received his Ph.D. and M.S. degrees in computer science from Stony Brook University in 2004 and M.Sc and B.E. degrees from Birla Institute of Technology and Science (BITS), Pilani, India, in 1997. He is currently a Full Professor of Computer Science at the University of Illinois at Chicago (UIC). He is recipient of the National Science Foundation CAREER award in 2009, several best paper awards including a 2010 NYU-Poly AT&T Best Applied Cybersecurity Paper Award and multiple UIC campus level awards for research as well as his teaching.