Data-driven approaches for peer-to-peer botnet detection and forecasting

Priyangika Piyasinghe

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

Part of the Computer Sciences Commons

Recommended Citation

Piyasinghe, Priyangika, "Data-driven approaches for peer-to-peer botnet detection and forecasting" (2019). Graduate Theses and Dissertations. 17767.

https://lib.dr.iastate.edu/etd/17767

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Data-driven approaches for peer-to-peer botnet detection and forecasting

by

Priyangika R. Piyasinghe

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
J. Morris Chang, Co-major Professor
Carl K. Chang, Co-major Professor
Yong Guan
Simanta Mitra
Wensheng Zhang

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

Copyright © Priyangika R. Piyasinghe, 2019. All rights reserved.
DEDICATION

To my mother, my wife, my daughter, my mother-in-law, my father-in-law, my sister, my brothers, and my friends.
# TABLE OF CONTENTS

| LIST OF FIGURES | ................................................................. | v  |
| LIST OF TABLES  | ......................................................................... | vii|
| ACKNOWLEDGMENTS | ........................................................................... | vii|
| ABSTRACT        | ................................................................................ | ix |

## CHAPTER 1. OVERVIEW

1.1 Peer to peer (P2P) botnets ................................................................. 1
1.2 Proposed approaches to detect and forecast P2P botnets as a community in dynamic networks ................................................................. 2

1.2.1 Research problem ........................................................................... 2
1.2.2 Proposed approach to P2P botnet detection ....................................... 3
1.2.3 Proposed approach to P2P botnet forecasting ....................................... 4
1.3 Dissertation organization ................................................................... 4

## CHAPTER 2. REVIEW OF LITERATURE

2.1 P2P botnet detection ............................................................................ 6
2.2 P2P botnet forecasting ......................................................................... 8

## CHAPTER 3. PEER-TO-PEER (P2P) BOTNET DETECTION

3.1 Communities in graphs ....................................................................... 10
3.2 Dynamic community to detect P2P botnets ......................................... 14

3.2.1 Basic idea ....................................................................................... 15
3.2.2 Mutual contact graph ...................................................................... 16
3.2.3 Dynamic community ....................................................................... 18

3.2.3.1 Numerical community features .................................................... 19
3.2.3.1 Structural community feature ...................................................... 20
3.3 System Design .................................................................................... 21

3.3.1 Mutual contact graph component ..................................................... 22
3.3.1.1 Generating contact sets ................................................................. 23
3.3.1.2 Computing edge and vertex attributes .......................................... 24
3.3.2 Dynamic community detection component ....................................... 25
3.3.2.1 Update mutual contact graph ......................................................... 26
3.3.2.2 Community detection ................................................................. 27
3.3.3 Suspicious botnet detection .............................................................. 29

3.4 Experimental Evaluation .................................................................... 30

3.4.1 Experimental environment ............................................................... 30
3.4.2 Datasets and analysis tool ............................................................... 30
3.4.3 Experimental dataset generation ...................................................... 31
3.4.4 Results ............................................................................................. 33

3.5 Discussion ........................................................................................... 36
LIST OF FIGURES

Figure 1.1 Botnet architectures ........................................................................................................2

Figure 3.1 Illustrate of mutual contact communities for a network (a) and its mutual contact graph (b), in which there are two mutual communities (C1: Host A, B, and C2: Host C, D, E)..............................................................................16

Figure 3.2 Number of mutual contacts between different types of host pairs contain in Table1 dataset.................................................................................................................................17

Figure 3.3 An overview of proposed community detection in a dynamic network ........22

Figure 3.4 Mutual Contact Graph Component.................................................................................23

Figure 3.5 Implementation of the Girvan-Newman algorithm on Spark using Redis memory structure to store the shortest paths between node pairs. ...............27

Figure 3.6 Comparison of time taken to detect dynamic communities natively and in Spark. ..................................................................................................................................................33

Figure 3.7 Comparison of time taken to detect dynamic communities in Spark with and without using Redis. ..................................................................................................................34

Figure 3.8 Precision on 40 graphs tested using the proposed model.................................35

Figure 3.9 Recall on 40 graphs tested using the proposed model. ........................................35

Figure 4.1 Simple long short term memory (LSTM) network.................................................40

Figure 4.2 More concise representation of LSTM ........................................................................41

Figure 4.3 Intensity of Zeus bot over the time .................................................................43

Figure 4.4 Size(host connected) of Zeus bot over the time ...............................................43

Figure 4.5 An overview of proposed P2P botnet forecasting approach .........................44

Figure 4.6 Intensity of Zeus bot with stationary series......................................................46

Figure 4.7 Attack’s intensity and prediction distributions for forecasting horizon1........51

Figure 4.8 Attack’s intensity and prediction distributions for forecasting horizon2.......51
Figure 4.9 Attack’s intensity and prediction distributions for forecasting horizon2........52
Figure 4.10 Attack’s intensity and prediction distributions for forecasting horizon4.......52
Figure 4.11 Attack’s size and prediction distributions for forecasting horizon1 ............53
Figure 4.12 Attack’s size and prediction distributions for forecasting horizon2 ............54
Figure 4.13 Attack’s size and prediction distributions for forecasting horizon3 ............54
Figure 4.14 Attack’s size and prediction distributions for forecasting horizon4 ............55
LIST OF TABLES

Table 3.1 Preliminary Experiment Dataset Summary .............................................. 15
Table 3.2 Experiment Dataset Summary ................................................................. 32
Table 4.1 Preliminary experiment dataset summary ................................................. 42
Table 4.2 MAE for the attack’s intensity for different forecasting horizons .............. 50
Table 4.3 MAE for the attack’s size for different forecasting horizons ...................... 50
ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisor Dr. Morris Chang for his guidance, caring, and providing me with an excellent atmosphere to do research.

Besides my advisor, I would like to thank the rest of my committee members: Dr. Carl Chang, Dr. Yong Guan, Dr. Simanta Mitra, and Dr. Wensheng Zhang for their encouragement and constructive comments. I would like to especially thank Dr. Carl Chang for providing me a space in his research lab and giving me the opportunity to present and discuss my research with his group for the past several years which helped me to develop my research skills.

Last but not the least, I would like to thank my wife, Nimanthi Atukorala, my daughter Sanuki Piyasinghe, my mother, Anoma Weeraman, my mother-in-law Lalani Dhanasekara, my father-in-law Nandasoma Atukorala, my brothers, Chamika, Nishan and Lakshan, my sister, Bhagya for all the encouragement, love and the support they provided from the beginning.
ABSTRACT

Peer-to-Peer (P2P) botnet is one of the major threats in network security for serving as the infrastructure that is responsible for various cybercrimes. Enterprises routinely collect terabytes of security-relevant data. This proposed work exploits such data to propose a novel Internet-scale P2P botnet detection that fuses big data behavioral analytics in conjunction with graph theoretical concepts. In addition to detecting botnets in large data sets, our method capable of meeting the challenges that incur botnet having encrypted command-and-control (C&C) channels, the stealthy botnet that hard to observe any malicious activities in the network traffic, and botnet with randomized communication patterns.

In a popular botnet-assisted attack scenario, the attacker(s) commands a swarm of bot-infected computers to send out flooding packets to a target server, intending to reduce the services provided by the server, to a state where they cannot be accessed by legitimate users. It is essential to detect these attacks commonly known as Distributed Denial of Service (DDoS) attacks accurately in a timely fashion so that mitigation can be done before a server down.

Apart from detecting the threat, it is important to the organization that they have significant insights about the targeted attack to understand future short and long term trends of an ongoing P2P botnet attack. This helps to quantify attack impacts like intensity and estimated number of compromised machines. The second part of our work focused on using time series analysis to identify those features and provide the capability to recognize the current as well as future similar situations and hence appropriately respond to the threat.
Experimental evaluation for detection and forecasting has demonstrated both high accuracy and great scalability of the proposed system.

*Keywords:* big data, DDoS attacks, P2P botnet, time series analysis, quantify attacks
CHAPTER 1. OVERVIEW

In this chapter, we provide an overview of our proposed data-driven approach for peer-to-peer botnet detection and forecasting.

1.1 Peer to peer (P2P) botnets

A botnet is a collection of compromised machines that are remotely controlled by one or more botmasters, through a command and control (C&C) channel. Botnet infrastructure responsible for a variety of cyber-crimes [1]: Government departments, commercial institutions, and common users suffered a lot from net theft, Distributed Denial of Service (DDoS) attack and a mass of spam caused by the botnets. DDoS activities, indeed, dominate today’s attack landscape.

Based on the architecture, botnets can be divided into two sets: (a) the centralized architecture, which uses Command and Control (C&C) channels such as Internet Relay Chat (IRC) to receive instructions from remote controller (botmaster), and (b) the decentralized architecture, which utilizes a Peer-to-Peer protocol to coordinate its operation. Figure 1.1 shows two examples of these two botnet architectures. A P2P botnet lifecycle consists with the following stages [2]: (a) Infection stage, during which the bot spreads, via email attachments, drive-by downloads, malicious software installation, etc.; (b) Rally stage, where the bot connects with a peer list and form P2P network; (c) Waiting stage, where the bot waits for the botmasters command; and (d) Executing stage, in which it actually attack carries out, such as a DDoS attack, generate mass of spam emails, etc. In this proposed work, we are first focusing on detecting P2P botnet at the waiting stage.
1.2 Proposed approaches to detect and forecast P2P botnets as a community in dynamic networks

1.2.1 Research problem

In this work we are addressing two main tasks related to P2P botnets:

1. Detecting P2P botnets

2. Forecasting future possibilities of P2P botnets

The detection of P2P botnets is challenging. For example, there exist some popular P2P sharing and communication applications like skype, uTorrent, and eMule. If the hosting machines of those applications are compromised, then identifying malicious traffic separately is a challenge. Moreover, botmaster can modify the encrypted traffic by injecting some bytes to ongoing traffic. As the volume of network traffic grows rapidly in a quick time deployed detection system is required to process a huge amount of information efficiently.
Even though the P2P botnet is detected, since the threat will likely become more common in the future, the propagation and communication mechanisms of P2P botnets should be carefully studied. An accurate forecasting model provides insights on features that affect how quickly a botnet is able to increase its size. These insights can help guide the development of anti-malware systems that are effective against P2P botnets. Forecasting also helps to identify other unique characteristics. What type of attack botnet making? Are all the requests coming from the same number of peers? Answers to these questions will identify any pattern in the attack which helps to block the requests with a firewall. Forecasting also helps in mitigation of the attack. High rate attacks must be mitigated by specialized hardware to withstand the attack load while allowing legitimate traffic to pass through.

1.2.2 Proposed approach to P2P botnet detection

Botnet always works as a community, triggers the very first question for our work: Can we detect P2P botnets through the analysis of community behaviors, without a "seed"[3] or any statistical traffic fingerprints [4]? To answer this question, we propose a novel community behavior based P2P botnet detection system, which is capable of detecting P2P bots as communities in a monitored network, with several challenges: (a) we assume botnets stay in the waiting stage, which also means there is no malicious activity can be observed [5]; (b) we assume the botnets C&C channels have been encrypted, so that no deep-packet-inspection (DPI) approaches can be deployed; (c) we assume botnets can randomize their communication patterns, as described in [6], [7], [8]; (d) we assume there is no bots blacklist or seed information available; (e) we can detect unknown P2P botnets without knowing any statistical traffic patterns; and (f) we do not require to monitor each host individually. In this proposed work, we present an efficient and fully distributed method to detect dynamic
communities on Spark (Apache Spark) to identify P2P botnets. The proposed detection method works in real-time. To process graphs efficiently, we use the GraphX component on Spark which exposes fundamental operators of Pregal API.

1.2.3 Proposed approach to P2P botnet forecasting

The forecasting approach is rendered by extracting session flows from P2P botnet traffic. Subsequently, feature-based time series are created using different time intervals which commonly known as lags. Differencing techniques are applied to time series data to get rid of seasonality and trends. Finally, forecasting technique is applied on time series data.

The major contributions of this work include,

- Characterizing and predicting P2P botnet attacks’ impact features, namely, intensity/rate and size
- Leveraging various time series analysis and forecasting methods, including, moving average, weighted moving average, and Recurrent Neural Networks (RNN)
- Evaluating the proposed approach using real P2P botnet traffic
- The ability of forecasting to both long term and short term time periods.

1.3 Dissertation organization

The rest of this dissertation is organized as follows: Chapter 2 reviews the literature on existing approaches in both P2P botnet detection and forecasting. Chapter 3 presents our proposed P2P botnet detection approach in detail. In Chapter 4, the proposed P2P botnet
forecasting approach is given. Finally, Chapter 5 concludes the dissertation with a future research road map.
CHAPTER 2. REVIEW OF LITERATURE

This chapter reviews the P2P botnet detection techniques and provides a brief introduction to the most common approaches being used in recent P2P botnet detection.

2.1 P2P botnet detection

Community identification for dynamic networks has received less attention in the field compared to identifications of communities in static networks. The community detection methods in dynamic networks can be categorized into two classes: incremental or online community detection where data is evolving in real-time; and offline community detection where all the changes of the network evolution data are known at the beginning.

In offline community detection, Tantipathananandh et al. [9], propose a clustering framework based on finding optimal graph colorings which is proved to be an NP-hard problem. They present heuristic algorithms that find near-optimal solutions and are demonstrated on small networks with little evolution. However, when scalability is considered, their algorithm in the current form is not ideal for large networks.

In online community detection, evolution clustering proposed by Chakrabarti et al. [10], simultaneously optimizes two potentially conflicting criteria. First, the clustering at any point should remain fixed to the current data as much as possible. Then, the clustering should not dramatically move from one time step to the next time step. One can obtain a balanced community structure between the quality of the present clustering result and the previous result. However, the network structure at each time step is usually required to be clustered separately, which results in higher complexity. Therefore, it is difficult to apply to identify and analyze large scale dynamic networks.
Ning et al. [11], propose an incremental algorithm that is initialized by a standard spectral clustering algorithm, followed by updates of the spectra as the dataset evolves. Compared with recomputation by standard spectral clustering for web blog data, their algorithm achieves similar accuracy but smaller computational costs. Leung et al. [12], discuss the theoretical label propagation algorithm for dynamic network data. The label propagation algorithm initializes each node with a unique label and proceeds by allowing each node to adopt the label most popular among its neighbors. This iterative process produces densely connected groups of nodes from the current state of the network. The static version of the label propagation algorithm is efficient. However, they did not discuss how efficient the algorithm can be in dynamic networks.

As a solution to large data growth over the past few years, researchers have proposed scalable data analytic methods along with storage and processing models (Kumar et al. [13], Sharma, et al. [14]). Known implementations to detect communities using Hadoop MapReduce big data framework include: SLPA using MPI (Kuzmin et al. [15]), the Louvain method using Apache Graph (Distributed Louvain Modularity [16]), the propinquity dynamics method using Hadoop MapReduce (Zhang et al. [17]), Scalable Community Detection (Prat-Perez et al. [18]) and many others. However these methods address the scalability of community detection, but in our work, we address more of efficiency in order to detect communities online in real-time.

Apache Spark’s implementation of a community detection method has been proposed in Buzan et al. [19]. They used the label propagation method with the help of friendship groups of individual users in social networks to identify the communities. In their work, the detection process is viewed as static rather than dynamic. To the best of our knowledge, this
work is the first to report of an Apache Spark implementation of community detection in dynamic networks. Firstly, we make use of the GrpahX ecosystem that comes with Spark to do the operations like message passing, merging and aggregating. Secondly, we benefit from Resilient Distributed Datasets (RDDs) - highly optimized data abstractions perfectly tailored for iterative algorithms (Zaharia et al. [20]).

2.2 P2P botnet forecasting

Malware propagation in Gnutella type P2P networks was described in Ramachandran et al. [21]. An analytical model that emulates the mechanics of the decentralized Gnutella type network was formulated and the study of malware spread on such networks was performed. In Goranin et al. [22], they have proposed the Genetic Algorithm based model dedicated to forecasting the evolution of propagation techniques used by the internet worms at the initial propagation phase. They discuss the effect of countermeasures on the evolution of the IRC botnets.

Ruitenbeek et al. [23] developed a stochastic model of P2P botnet formation to provide insight on possible defense tactics and examine how different factors impact the growth of the botnet. Rajab et al. [24] discuss the difficulties in estimating the size of botnets. They focus on tracking IRC botnets and admit that peer-to-peer botnets would bring a whole new set of challenges to botnet tracking. In the area of botnet modeling, Dagon et al. [25] have monitored botnets and detected diurnal patterns that affect the botnet propagation rate and attack strength. They created a diurnal propagation model to reflect this pattern. Their work also focuses on botnets with centralized command-and-control structures.

In Yang et al. [26], a branching process approximation characterizing the file transfer was presented. Qiu et al. [27] developed a stochastic fluid model for BitTorrent-like
networks and the steady-state properties of the system are analyzed. A limitation of the above works is that they are specialized to Bit-torrent like networks. The issue of worms in peer-to-peer networks is addressed in Zhou et al. [28] wherein the authors perform a simulation study of the dangers posed by P2P worms and proceed to outline possible mitigation mechanisms.

In the current work, we formulate a comprehensive model for malware spread in P2P networks that addresses quantify intensity and size of the attack in the future. We propose and adopt a systematic approach for inferring P2P activities, testing for predictability of P2P traffic by applying prediction models. Then we leverage various time series analysis and forecasting methods, including, moving average, weighted moving average, and recurrent neural network.
CHAPTER 3. PEER-TO-PEER (P2P) BOTNET DETECTION

As we mentioned earlier, peer-to-peer (P2P) botnet detection is one of the major threats to network security for serving as the infrastructure that is responsible for various cybercrimes. In this chapter, we present our proposed approach to P2P botnet detection by identifying the evolution of communities within dynamically changing networks is important to understand the latent structure of complex large graphs.

3.1 Communities in graphs

Biological, social, technological, and information networks can be studied as graphs, and graph analysis has become crucial to understand the features of these systems. For instance, social network analysis started in the 1930s and has become one of the most important topics in sociology [29], [30]. In recent times, the computer revolution has provided scholars with a huge amount of data and computational resources to process and analyze these data. The size of real networks one can potentially handle has also grown considerably, reaching millions or even billions of vertices. The need to deal with such a large number of units has produced a deep change in the way graphs are approached [31], [32].

In a random graph, the distribution of edges among the vertices is highly homogeneous. For instance, the distribution of the number of neighbors of a vertex, or degree, is binomial, so most vertices have an equal or similar degree. Real networks are not random graphs, as they display big inhomogeneity, revealing a high level of order and organization. The degree distribution is broad; therefore many vertices with low degrees coexist with some vertices with a large degree. Furthermore, the distribution of edges is not
only globally, but also locally inhomogeneous, with high concentrations of edges within special groups of vertices, and low concentrations between these groups. This feature of real networks is called community structure [33], or clustering.

A network is said to have community structure if it divides naturally into groups of nodes with dense connections within groups and sparser connections between groups. Community detection is an important tool to reveal the latent structure of the graphs. Many concrete applications are there for communities, such as Web clients who have similar interests and are geographically near to each other served by dedicated mirror servers to improve performance of services provided on the World Wide Web [34], group detection on social and collaboration networks [33], protein community discovery in protein-protein interaction networks [35], and efficient recommendation systems set up by identifying clusters of customers with similar interests in the network of online purchase relationships between customers and products [36]. A variety of approaches for community detection have been proposed due to the wide range of these applications. For example, modularity based [37][38], weighted community clustering (WCC) based [39], graph partition approach [40], statistical inference approach [41].

Mainly network structures with respect to time can be identified as two types: static and dynamic. Static networks are the graphs with permanently fixed nodes and edges over the time considered. Much of the current work in community detection in complex networks are based on static networks, that either is derived from the aggregation of data overall time or a single snapshot at a particular time. However, some networks (e.g. social networks) evolve over time making topological changes such as adding new connections, removing the existing connections, merging the properties, etc. which can have major impacts on the
dynamics over the network. A dynamic representation of complex networks where nodes and edges shift according to changes in the system reflects this reality more closely. When faced with dynamic networks, traditional community mining methods may lead to unrealistic divisions [42].

Numbers of studies on analyzing communities and their evolution in dynamic networks have been proposed [43] [44]. However, there are some limitations. For example, communities and their changes are studied separately, or the results rely on expensive human interpretation [45]. Most community detection methods treat each configuration over time as a separate network although the changes are not that significant from the last timestamp. For example, dynamic networks may vary by only one node or one edge. Redundant computations are required when the information regarding communities from the previous configuration is not used and the communities have to be recomputed as a whole. The efficiency of these algorithms can be greatly improved if the recomputation is limited only to the portions of the network that are affected by the modifications.

In this chapter, we propose a new algorithm for community identification to overcome the limitations in existing methods. Our work is based on the fact that most communities tend to evolve gradually over time without dramatically moving from one time step to the next time step.

We used peer-to-peer (P2P) botnet detection as a case study for community detection in dynamic networks. A botnet is a collection of compromised machines that are remotely controlled by one or more botmasters, through a command and control (C&C) channel. Botnet infrastructure is responsible for a variety of cyber-crimes. Government departments, commercial institutions, and common users are suffering a lot from net theft, distributed
denial of service (DDoS) attacks and a mass of spam caused by a botnet. Therefore the ability to detect botnets is a crucial component of a network's security system. Botnets can be divided into two sets based on the architecture:

- the centralized architecture, which uses C&C channels such as internet relay chat (IRC) to receive instructions from a remote controller (botmaster)
- the decentralized architecture, which utilizes a P2P protocol to coordinate its operation.

When considered these two sets, centralized botnets contain a single point of failure which is a disadvantage. Therefore, most of the recently developed botnets attempt to build on P2P architecture [46] to take advantage of the resilience offered by the architecture [47]. In P2P botnet, even if a certain number of bots are identified and taken down [2], the other peers can still work and carry out the attack. Therefore, it is important to identify all the bots and take them down. In this work, we detect such P2P botnet as a community detection problem in the dynamic network.

We present an efficient and fully distributed method to detect dynamic communities on Spark [48] to identify P2P botnets. The proposed detection method works online in real-time. The basic idea is to detect P2P botnets as efficiently as possible, because the more time it takes, the more the botnet can do its malicious activities, and hence the damage is significant. To process graphs efficiently, we use the GraphX component on Spark which exposes fundamental operators of Pregal API.

Our botnet detection starts from building a mutual contact graph (MCG) for all of the targeted hosts coming from the monitored network trace. Then, it applies a community detection method on Spark to the whole graph into several different communities (subgraphs)
and clustering each bot that comes from the same botnet into the same community. Detection system clusters each bot into its own botnet communities and distinguishes the legitimate hosts and bots into different communities. The detection system then applies an outlier selection strategy on the list of communities utilizing community behavior level features to detect potential botnet communities and further identifies potential bots from each botnet community by considering community behavior level features. We design our experiments with real network trace from a public traffic archive and 2 botnet datasets 13 Storm [49] and 3 Waledac [50]. Test performed on 40 different randomly generated datasets that each contains 5000 targeted hosts, and achieve a result of both higher detection rate with a very low false-positive rate (FPR) and improved execution time. The main contributions of this paper are as follows:

- we have greatly improved the efficiency of the community detection algorithm by limiting the recalculations at each step to the portions of the network that are affected by the modifications
- we have implemented a scalable dynamic community model in a distributed environment based on Apache Spark using Pregel computational model

### 3.2 Dynamic community to detect P2P botnets

In this section, we present the basic idea and the details of the proposed algorithm. We first begin with an intuitive explanation in the next subsection and then provide a more detailed and formal explanation in subsequent subsections.
3.2.1 Basic idea

This section presents the basic ideas applied in our approach. We start by giving an explanation of the foremost feature that has been applied in our approach, the mutual contacts. These give us the basic idea to formulate a P2P botnets clustering problem into a graph community detection problem. Then, we provide a more detailed discussion about community behavior features, which contain numerical community features and structural community features, that have been applied to both botnet community detection and final bot candidates selection.

In order to illustrate the community behavior features and statistical traffic features, we conduct a list of preliminary experiments on a dataset obtained from [5] which we selected a sub dataset that contains 24 hours traffic trace of four popular legitimate P2P applications: eMule, FrostWire, uTorrent and Vuze, and two P2P botnets: Storm and Waledac. Table 3.1 shows the summary of our preliminary experiment dataset.

<table>
<thead>
<tr>
<th>Application</th>
<th>Number of Hosts</th>
<th>Hours</th>
<th>Average Number of Flows</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>eMule</td>
<td>2</td>
<td>24</td>
<td>175151</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>FrostWire</td>
<td>2</td>
<td>24</td>
<td>217330</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>uTorrent</td>
<td>2</td>
<td>24</td>
<td>965893</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>Vuze</td>
<td>2</td>
<td>24</td>
<td>652029</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>Storm</td>
<td>13</td>
<td>24</td>
<td>666238</td>
<td>UDP</td>
</tr>
<tr>
<td>Waledac</td>
<td>3</td>
<td>24</td>
<td>439455</td>
<td>TCP</td>
</tr>
</tbody>
</table>
3.2.2 Mutual contact graph

Mutual contact graphs are used in different perspectives (e.g., Coskun et al. [7]). A mutual contact between a pair of hosts is defined as a set of shared contacts or connections between the corresponding pair of hosts.

If we simply divide all of the hosts into legitimate hosts and different types of botnets, we can divide host pairs into four categories:

- legitimate-legitimate, a pair of legitimate hosts;
- legitimate-bot, a pair of hosts between legitimate host and bot;
- bot1-bot1, a pair of bots within the same botnet; and
- bot1-bot2, a pair of bots from different botnets.

Figure 3.1 Illustrate of mutual contact communities for a network (a) and its mutual contact graph (b), in which there are two mutual communities (C1: Host A, B, and C2: Host C, D, E).
To illustrate the mutual contacts patterns among different types of host pairs, we have conducted a preliminary experiment using the datasets in Table 1 that contains 8 legitimate hosts (running 4 P2P applications: eMule, FrostWire, uTorrent and Vuze), 13 Storm hosts and 3 Waledac hosts. As shown in Figure 2, different types of host pairs have different mutual contacts patterns. For instance, compared with legitimate-bot and bot1-bot2, host pairs within bot1-bot1 and legitimate-legitimate have a larger number of mutual contacts. Moreover, the host pairs within bot1-bot1 have the largest number of mutual contacts.

Compared with legitimate hosts, there is a much more significant probability that a pair of bots within the same botnet have a mutual contact [7] since bots within the same P2P botnet tend to receive or search for the same in the set of its IPs. First, P2P applications usually have a large number of distinct degrees because peer IPs are usually spreading across
a large number of networks. Moreover, in order to prevent peers from churning in a P2P botnet, the botmaster has to check each bot periodically, which translates into a convergence of contacts among peers within the same botnet. However, since bots from different botnets are controlled by different botmasters, they will not have a large number of mutual contacts. Legitimate host pairs may also have a small set of mutual contacts since nearly all hosts would communicate with a list of very popular servers such as google.com, facebook.com, etc. Furthermore, the host pairs running the same P2P applications may also result in a decent number of mutual contacts, if they are accessing the same resource from the same set of peers by coincidence. However, in reality, different legitimate P2P hosts usually will not search for the same set of peers intentionally. Therefore, we can utilize these different mutual contacts patterns among different types of host pairs to cluster bots within the same botnet into the same communities.

To utilize the mutual contacts feature, we begin with constructing a mutual contact graph. The basic idea is illustrated in Figure 1, in which HostA and HostB are linked together in Figure 1(b) since they have mutual contacts Host1 and Host2 in Figure 3.1(a). Similarly, HostC, HostD, and HostE are linked to each other (in Figure 3.1(b)) since every pair of them is sharing at least one mutual contacts (in Figure 3.1(a)). The detail of implementation is given in Section 3.3.

### 3.2.3 Dynamic community

In this section, we present a list of community behavior-based features that can be utilized to identify potential botnet communities. We assume that the botnet communities must have community-level behavior features that are distinguishable from legitimate hosts or legitimate communities since botnet always works as a group/community. However, since
the community behaviors of single bots are usually changing dynamically, it would be very
difficult to identify a single bot from a number of legitimate hosts, only based on the single
host's behaviors. Community behavior features can be roughly divided into two categories:
- numerical community features.
- structural community features.

3.2.3.1 Numerical community features

Numerical community features contain a list of community-level numerical statistics
of each community such as the average number of mutual contacts between each pair of
hosts within the same community, the average number of degrees or the number of contacts
among hosts within the same community and the average traffic statistics or connection
characteristics.

In this work, we do not utilize any traffic statistics related features to detect botnets
since they can be randomized or changed dynamically without much influence on the
primary functions of a botnet community. Our work mainly focuses on two numerical
community features: the average degree and mutual contacts ratio.

Average degree: P2P bots within the same botnet tend to have a similar number of degrees.
These bots are directly controlled by the same machine without human involvement. As a
result, they receive the same C&C messages and conduct similar malicious activities. The
average degree of a P2P botnet community is much higher than a legitimate community.
Although an individual legitimate host (e.g. P2P hosts) may have a large degree, the other
legitimate hosts within the same community may not have a similarly large degree. For
example, even if a legitimate community contains several high degree hosts, the average degree of that community might not be that significant. Furthermore, legitimate P2P hosts usually have a larger degree in the same time period compared to P2P bots since botnets usually act stealthily that generate a low volume of traffic. Therefore, we consider the average degree among all hosts within the same community as the first community behavior features.

**Average mutual contact ratio:** Average mutual contact ratio is defined as the number of mutual contacts between pairs of hosts divided by the number of all contacts of those pair of hosts. This feature is based on three assumptions:

- P2P botnet community contains at least two bots since one member communities cannot have this feature.
- the mutual contact ratio between a pair of bots is much higher than that between a pair of legitimate hosts.
- each pair of bots within the same botnet has a similar mutual contact ratio.

Therefore, we consider the average mutual contacts ratio among all pairs of hosts within the same community as the other community behavior features.

### 3.2.3.1 Structural community feature

This feature captures the structural characteristics of a botnet community. As discussed in previous subsections, every pair of bots within the same botnet tends to have a considerable number of mutual contacts. Therefore, if we consider each host as a node and
draw an edge between two nodes if the pair of hosts represented by those nodes have a certain number of mutual contacts, then the bots within the same botnet should form a clique (a complete graph). In contrast, the contacts sets among legitimate P2P host usually tend to diverge into different networks which result in a relatively low probability of forming a clique. Hence, we can translate the P2P botnets detection problem into a complete graph detection problem that can detect complete graphs with certain requirements based on the node and edge weights. However, since the clique detection problem is NP-complete, it is not algorithmically feasible to apply a clique detection to detect botnets. Therefore, we need to combine both numerical and structural features to identify P2P botnets.

Detecting communities dynamically is an iterative computational process. When the graph size gets larger over time, it requires an efficient graph processing platform to find the communities within. Spark has received a lot of attention in the recent past as a platform to process large data sets in a relatively quick time. GraphX component under the Spark specially built to analyze huge graphs using fundamental operators like subgraph, joinVertices, aggregateMessages, etc. which have benefited from Resilient Distributed Datasets (RDDs) - highly optimized data abstractions perfectly tailored for iterative algorithms [19].

### 3.3 System Design

In this section, we present a detailed description of the system design to detect P2P botnets. The proposed system consists of three main components, that work synergistically to

- construct mutual contact graphs from network trace
- cluster bots into its botnet communities during dynamic community detection process
• identify potential botnets communities and further identify potential bots from each botnets community.

Figure 3.3 illustrates the system framework and the description of each component are given below.

3.3.1 Mutual contact graph component

The mutual contact graph is a weighted undirected graph where each node represents a host. An edge in mutual contact graph implies that the pair of hosts correspond to the two nodes of the edge is sharing at least one mutual contact. To utilize the mutual contacts feature mentioned in section 3.2, we start by building a mutual contact graph from the network trace. This mutual contact graph is updated during the online detection process as shown in Figure 3.4.
Consider applying our botnet detection procedure on the network boundary (e.g. firewall, backbone link) illustrated in Figure 1(a). The input for this component is a list of internal network hosts $V_{in}$, such as HostA, HostB, HostC, HostD and HostE in Figure 1(a), and a set of netflow trace $F = (S,D)$, where $S = \{s_1, s_2, \ldots, s_n\}$ is a set of hosts appeared in the netflow trace, including both internal network hosts $V_{in}$ and external network hosts, and $F = \{f_1, f_2, \ldots, f_{|F|}\}$ is a set of traditional 5-tuple flows. The output is the mutual contact graph $MCG(V,E)$, where each node $s_i \in V$ is an external contact corresponding to internal host and each edge $e_{uivj} \in E$ contains a weight attribute that shows the ratio of mutual contacts between nodes ui and vj. The weight attribute of an edge is the same as the Jaccard Similarity between the two nodes. The algorithm for this component is given in Algorithm 3.1. Below is a detailed description of the main steps in this component.

3.3.1.1 Generating contact sets

Assume we only monitor the network at the network boundary (e.g. firewall, backbone link). Then, the detection system only considers the bot/non-bot membership of the hosts belonging to the internal network. If an external host has communication with an
internal host, we call the external host as a contact of the corresponding internal host. In this step, for each internal host \( h_i \in V_{in} \), it will generate a contact set \( S_{hi} \) for each internal host. This process contains:

- initializing an empty contact set \( S_{hi} \) for each internal host \( h_i \in H_{in} \)
- adds new contacts into each contact set based on the source/destination hosts (IP) information of each flow.

### 3.3.1.2 Computing edge and vertex attributes

As mentioned earlier, both nodes and edges in a mutual contact graph have weight attributes. Vertex weight attribute \( d_i \) is the cardinality of that node's contact set, and edge weight attribute \( w_{ij} \) is the ratio of cardinalities of the common contact set and the total set that they own.
Dynamic community detection consists of two parts:

- update mutual contact graph
- community detection.
3.3.2.1 Update mutual contact graph

When a new node is connecting to an existing graph, the mutual contact graph $MCG(V, E)$ is also needed to be updated. To update the mutual contact graph, the adjacency list of incoming node $v_{in}$ has to be considered with the adjacency list of existing nodes in the graph. To keep this much data available, we used Redis [51] in-memory data structure along with RDD in Spark. The corresponding algorithm is given in Algorithm 2. Here, if $v_{in}$ is connected to $v_j$, then the contact set of $v_j$ will be updated. Therefore, the weights of edges already connected to $v_j$ need to be updated using $ADJ_{vj}$. This step takes linear time.

![Algorithm 3.2: Update Mutual Contact Graph.](image)

**Algorithm 3.2**  
Update Mutual Contact Graph.
3.3.2.2 Community detection

Many community detection methods have already been discussed in Holz et al [52]. We use the Girvan-Newman community detection algorithm [32] considering its' simplicity in adaptation to run parallelly on the Spark environment, which will discuss later in this section. Girvan-Newman algorithm runs in four steps:

(a) find the shortest path between pairs
(b) find the edge betweenness
(c) remove edge with highest betweenness
(d) repeat step (b) and (c) until no remaining edges.

The performance bottleneck of the Girvan-Newman algorithm in step (a) which always of complexity O(n²). By storing the shortest path in Redis memory structure we make dynamic community detection efficient at the entrance of a new node into existing community structure. Figure 3.5 illustrates these steps.

Figure 3.5 Implementation of the Girvan-Newman algorithm on Spark using Redis memory structure to store the shortest paths between node pairs.
At the end of each mutual contact graph update, the new community structure $CMM_G$ is derived by considering the shortest paths $SPATH_{CMM,G}$ of the previous community. This process generates a considerable amount of metadata. Again, to hold the data efficiently in memory, we use Redis data structures. The corresponding algorithm is given in Algorithm 3.3.

When the node joins the existing graph, it can either

(a) join the existing community or

(b) divide the existing community to smaller communities.

For case (a), updating the shortest paths $SPATH_{CMM,G}$ with a new node takes linear time since the new node always becomes a leaf node (an end node) in the resulting community. For case (b), the shortest paths between every pair of nodes need to be recalculated in the resulting community. Here the runtime depends on how well the graph is connected. However, by definition, a community is a tightly connected knot in Graph. Hence, the number of iterations in message-passing considerably reduced. For example, if we need to find a minimum node in a clique, it will only require a single round of messages passing in Spark GraphX.
To detect suspicious community in $CMM_G$, both average degree $AVG\_Deg$ and average mutual contact ratio $AVG\_Mut\_Cont\_Rat$ are considered. Let $AVG\_Deg_{ci}$ and $AVG\_Mut\_Cont\_Rat_{ci}$ be the average degree and average mutual contact ratio of community $c_i$ respectively. If the product of $AVG\_Deg_{ci}$ and $AVG\_Mut\_Cont\_Rat_{ci}$ is greater than the predefined threshold $T$, then that community $c_i$ declared as a suspicious community. The value of $T$ is decided at the training phase of the experiment. Initially, all the nodes in the community declared suspicious, but this leads to more false positives. Later in the experiment, a host is declared as suspicious based on the flow classification of hosts in a suspicious community. Algorithm 3.4 shows the algorithm of suspicious botnet detection.
3.4 Experimental Evaluation

3.4.1 Experimental environment

The experiments are conducted on a PC with a 12 core Intel i7-5820 Processor, 32GB RAM, 470GB SSD and 4TB HHD, and on 64-bit Ubuntu14.04 LTS operating system. Spark-1.4.1 is used with GraphX running in local mode.

3.4.2 Datasets and analysis tool

To evaluate our system, we utilize botnet dataset DB and legitimate dataset DL. DB is obtained from the University of Georgia [5], which contains 24 hours real network trace from 13 hosts infected with Storm and 3 hosts infected with Waledac. The Storm dataset contains 666,238 flows and 145,972 unique IPs other than 13 Storm hosts. The Waledac

```
Data: CMM_G: Community graph such that CMM_G = \bigcup c_i, where c_i is the i^{th} community, T: Threshold.
Result: SP_CMM: Suspicious community.

SSP_CMM = \emptyset

for each community c_i in CMM_G do
    Node_{c_i} = Set of nodes in c_i
    Edge_{c_i} = Set of edges in c_i
    AVG_Deg_{c_i} = Average degree of c_i = \frac{\sum_{p \in Node_{c_i}} d_p}{|Node_{c_i}|}, where d_p is the degree of p^{th} node of c_i.
    AVG_Mut_Cont_Rat_{c_i} = Average mutual contact ratio of c_i = \frac{\sum_{(p,q) \in Edge_{c_i}} w_{pq}}{|Edge_{c_i}|}, where w_{pq} is the mutual contact ratio of (p,q) edge of c_i.
    if AVG_Deg_{c_i} \times AVG_Mut_Cont_Rat_{c_i} > T then
        SSP_CMM = SSP_CMM \cup c_i
    end
end
```

Algorithm 3.4  Suspicious Botnet Detection.
dataset contains 439,455 flows and 29,973 unique IPs other than 3 Waledac hosts. No malicious activities can be observed in this botnet dataset.

DL is downloaded from the MAWI Working Group Traffic Archive (MAWI Working Group Traffic Archive [53]) which contains 24 hours anonymized network trace at the transit link of WIDE (150Mbps) to the upstream ISP on 2014/12/10 (sample point F). DL contains approximate 407,523,221 flows and 48,607,304 unique IPs. 79.3% flows are TCP flows and the rest are UDP flows.

We utilize ARGUS (ARGUS [54]) to process and cluster network trace into 5-tuple TCP/UDP flows.

### 3.4.3 Experimental dataset generation

In order to evaluate our system, we generate three main datasets by mixing the network trace from DB and the DL. Table 3.2 illustrates the summary of three main datasets. Each DS and DW contains 20 sub-datasets and each sub-dataset includes the network trace from 5000 internal hosts sampled from DL. Among each sub-dataset’s 5000 internal hosts in DS, 13 hosts are mixed with Storm network trace. Similarly, among each sub-dataset's 5000 internal hosts in DW, 3 hosts are mixed with the Waledac network trace. DS+W contains 40 sub-datasets and each sub-dataset includes the network trace from 5000 internal hosts sampled from DL where 3 hosts are mixed with Waledac network trace and the other 13 hosts are mixed with Storm network trace. A sub-dataset is the basic unit for one test. In the rest of this section, the term "graph" represents the graph generated using a sub-dataset.
Table 3.2 Experiment Dataset Summary

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Number of Graphs</th>
<th>Bots/Internal Hosts</th>
<th>The average number of Totals Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>20</td>
<td>13/5000</td>
<td>786,494</td>
</tr>
<tr>
<td>DW</td>
<td>20</td>
<td>3/5000</td>
<td>543,913</td>
</tr>
<tr>
<td>DS+W</td>
<td>40</td>
<td>16/5000</td>
<td>1,209,112</td>
</tr>
</tbody>
</table>

The first step of generating experimental sub-datasets is to sample a list of background hosts from DL. As discussed in Section 2, the design of our system is to deploy at a network boundary (e.g. firewall, gateway, etc.) where the network forms a bipartite structure so that we can only capture the connection between internal hosts and external hosts. Therefore, we need to sample a list of internal hosts such that any pair of them should not have any connections to each other.

To maintain a bipartite network structure of botnets network trace, we eliminate all of the flow between every two bots in DB. Further to make sure after mixing both botnet and legitimate network trace, each graph still maintains a bipartite structure. To select 5000 nodes, we used two-coloring of the graph.

To mix the botnet trace with sampled legitimate trace, for DS, each time we randomly select 13 legitimate hosts out of the 5000 hosts, map 13 Storm hosts IPs to the 13 legitimate hosts IPs, and merge the corresponding network trace. DS and DS+W are generated using the same procedure.

This experimental dataset generation process is repeated for a total of 80 times (each time we start from a different random host; 40 times for DS+W, 20 times for DS and 20 times for DW). Finally, we obtained DS, DW, and DS+W to evaluate our system.
respectively. From the generated graphs, 40 graphs are used to find the threshold and rest used for testing the online detection of communities.

3.4.4 Results

Figure 3.6 illustrates the comparison of time taken to detect communities in the dynamic version of original Girvan Newman that runs natively and in Spark. According to the figure, there is a significant improvement of execution time in Spark by restricting recalculations at each step only to the portion of the network that affected by modifications and keeping most of the metadata required for recalculations in memory.

![Graph of time taken to detect dynamic communities natively and in Spark](image)

Figure 3.6 *Comparison of time taken to detect dynamic communities natively and in Spark.*

By using Redis in-memory data components we can further reduce writing to disk by Spark in between the jobs. Hence the performance is increased as shown in Figure 3.7.
Figure 3.7 Comparison of time taken to detect dynamic communities in Spark with and without using Redis.

We utilize precision and recall as our evaluation criterion:

\[ \text{Precision} = \frac{TP}{TP + FP} \]
\[ \text{Recall} = \frac{TP}{TP + FN} \]

where,

\( TP \) = True Positive
\( FP \) = False Positive
\( FN \) = False Negative.

Figure 3.8 and Figure 3.9 show the precision and recall of 40 graphs tested using our model respectively. The graphs having bot count of 16 with both Waldec and Storm have the highest average precision of 0.923. Graphs with only 3 Waldec bots have an average precision of 0.63. From this result, we can see graphs with higher bot count gives less false
positive. We will discuss possible reasons in Section 3.5. Average recall for all graphs is 1 that implies higher detection from our model.

Figure 3.8 Precision on 40 graphs tested using the proposed model.

Figure 3.9 Recall on 40 graphs tested using the proposed model.
3.5 Discussion

As shown in the experimental results, the proposed system achieves higher average precision and higher average recall. Moreover, the comparison with the dynamic version of the Girvan-Newman algorithm shows a relatively efficient runtime.

Despite these significant improvements, the proposed system is not without limitations.

Although the proposed system is capable of detecting all the bots in the network, the existence of false positives is not fully eliminated. By analyzing each trace, we have identified the following reasons may lead to the existence of false positives.

(a) if one-degree legitimate hosts only connection is a bot, then it will be clustered into the same community as the corresponding bot. When selecting botnet communities this one-degree host will also be considered as bot.

(b) the best community selected in the current timestamp to join the incoming node might not be the ideal community based on the connections that node will have in the future time stamps.

The first reason can be overcome by eliminating very low degree nodes while selecting bot candidates from suspicious communities. For the second one, we can keep all the connection properties of the incoming node with the help of the distributed storage and consider all this information when a connection appears with the existing node.
CHAPTER 4. PEER-TO-PEER (P2P) BOTNET FORECASTING

This chapter described the proposed P2P botnet forecasting approach.

4.1 Time series forecasting

Data are available in different formats and structures. Temporal data is a type of data that varies over time. These data are represented using time stamps. The process of extracting or discovering patterns of data from temporal databases is temporal data mining. The major process involved in temporal data mining is the analysis of temporal data and finding useful patterns. Time series analysis and forecasting [55] are part of the temporal data mining. The collection of a large number of data values within a uniform time interval is termed as Time series data. The time can be represented as a year, month, week, day, etc. The time series is analyzed to predict the changes that happen within the given data and to predict the changes that will happen in the future.

Time series prediction is done in many applications that deal with numerical data. The prediction can be done on the basis of three different time-spaces as the Short-Term period, Mid-Term period and Long-Term period. The process of time series forecasting is applied in a wide range of areas such as business, stock market, and exchange, weather, electricity demand, cost and usage of products like fuels, electricity, etc. and in any kind of place that has specific seasonal or trendy changes with time where the forecasted values or conditions are helpful in many ways to make useful decisions. Here we present how we can use time series analysis to forecast P2P botnet attacks.

For most machine learning models, steps involve in the process is training a model, test it, retrain it if necessary until we get satisfactory results, and finally evaluate it on a hold
out data set. If the model’s performance is satisfactory, then we deploy it to production. Once in production, model is run against new data as they come in. We may have to update model if there is significant changes in the data that feed into model over some period of time. Model training is a one-time activity, or done at most at periodic intervals to maintain the model’s performance to take into account new information. For time series models, this is not the case. Instead we have to retrain our model every time we want to generate a new forecast. For P2P botnet forecasting problem, this retraining of the model makes sense since attacker varying its attack pattern over the time to keep a low profile to go undetected.

### 4.2 Background concepts

#### 4.2.1 Deep neural networks (DNN)

Deep Neural Network (DNN) can be considered general, flexible, nonlinear statistical techniques capable of learning complex relationships between variables in a multitude of fields of study. This technique has a series of advantages compared to classical statistical models. First of all, DNN does not depend on the fulfillment of statistical assumptions such as, for instance, the type of relationship between variables or the type of data distribution. Secondly, as universal function approximates, they are capable of fitting linear and nonlinear functions without the need for knowing the shape of the underlying function.

The basic architecture upon which DNN models are built is the perceptron. A perceptron has three distinct sets of nodes: (1) a set of nodes representing model inputs, (2) a set of computational nodes, and (3) a set of nodes representing model outputs. We refer to each set as a layer. The defining feature of the perceptron is that only one layer of computational neurons is used to transform inputs to target outputs.
Recent developments have led to the widespread application of neural networks in many fields. A combination of several factors is responsible for this. Biological advancements in our understanding of neurons, and particularly the functioning of the visual cortex, inspired innovations in the modeling and structure of contemporary neural networks. Advancements in computing technology such as the increased availability of massive data stores, increases in access to massive compute power via cloud service providers, and general advancements in the speed of computer hardware have made training complex neural networks computationally feasible.

4.2.2 Recurrent neural networks (RNN)

Recurrent Neural Network (RNN) [56] models developed along with research on the application of neural networks to language parsing and translation. RNN architecture draws information from the temporal structure of the input data. Specifically, these types of networks draw upon the sequence in which the input data is presented to the model. Recurrent neural networks do this by accepting input not only from the current input in a sequence but from the state of the network that arose when considering previous inputs in that sequence. There are many variants of RNN architectures. In this paper, we focus on a particular type called a long short term memory (LSTM) networks [57]. This type of RNN architecture is notable because it has the capacity to store a long-running memory about the sequence along with short-run memory of the most recent network outputs. Consequently, this allows the network to draw upon broad contextual features in the data as well as information provided by only the most recent elements in a sequence.
4.2.3 Long short term memory (LSTM)

As illustrated in Figure 4.1, each input $x$ is fed into a computational node (we might think of this as an a1-node layer), which also accepts inputs from the output of the preceding layer (denoted $s_i$, $h_i$). The term $s_i$ represents the state of the network at the $i$th member of the sequence. The state is the long-running memory of the sequence, as informed by the elements of the sequence to which the network has been exposed. The term $h_i$ is the prediction of the layer that corresponds to a given element $(i)$ in the sequence.

![Simple long short term memory (LSTM) network](image)

Figure 4.1 Simple long short term memory (LSTM) network

Note that the architecture is consisting of many layers and having the following properties:

1. Each layer corresponds to a particular element in the sequence
2. Each layer receives the network's long-run understanding of the sequence so far
3. Each layer receives the output generated from the previous element in the sequence. Hence, more concise representation of the recurrent architecture is provided in Figure 4.2.

![Figure 4.2 More concise representation of LSTM](image)

4.3 Forecasting peer to peer functionality

As described earlier in Chapter 1, the initial infection involves a Trojan installing some files onto a vulnerable machine. However, a compromised machine is not part of the botnet until it joins the P2P network that controls the botnet.

After the Trojan installs the initial infection files and inserts the malware into the services process, the malware begins its bootstrap process. The infection files include a list of nodes that are already part of the botnet, so the newly compromised computer attempts to
connect to those nodes and become a peer. After successfully joining the network, the new bot updates its list of peers to find other close peers. The new bot also searches the network to find an encrypted URL that points to the secondary injection payload. The bot decrypts this URL and downloads the secondary injection code. This secondary injection code then executes on the bot. The secondary injection code is the code that sends the spam emails, participates in DDoS attacks, or performs other malicious botnet activities. The bot can also be programmed to periodically search for updates using the P2P network. These continuous scanning of the network can be seen in the form of intensity(byte/sec) increase or simply the number of communication increases between the host. These communications can be quantified and use as a feature to forecast P2P bots.

To illustrate these statistical features, we conduct a list of preliminary experiments on a dataset obtained from [5], from which we selected a sub-dataset that contains 24 hours traffic trace of infamous P2P botnets Zeus, Storm, and Waledac. Table 4.1 shows a summary of our preliminary experiment dataset.

Table 4.1 Preliminary experiment dataset summary

<table>
<thead>
<tr>
<th>Application</th>
<th># of Hosts</th>
<th>Hours</th>
<th>Average # of Flows</th>
<th>Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeus</td>
<td>2</td>
<td>24</td>
<td>652029</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>Storm</td>
<td>13</td>
<td>24</td>
<td>666238</td>
<td>UDP</td>
</tr>
<tr>
<td>Waledac</td>
<td>3</td>
<td>24</td>
<td>439455</td>
<td>TCP</td>
</tr>
</tbody>
</table>

Figures 4.3 and 4.4 demonstrates how intensity and number of hosts connected over the time for 24hr Zeus bot. We have used bin size 30 minutes instead of the original seconds scale to show the variation of the above mentioned features over time.
System Overview

The main components of our proposed approach are depicted in Figure 4.5. The first forecasting model is rendered by extracting session flows from P2P botnet traffic. Subsequently, feature-based time series are created using different time intervals which commonly known as lags. Differencing techniques are applied to time series data to get rid of
seasonality and trends. Finally, forecasting technique is applied on time series data. The proposed approach is detailed next.

4.4.1 Extracting Session Flows

In order to filter out the scanning activities, we split the connections into separate session flows, where each session consists of a unique source and destination IP/port pair. The rationale for this is that P2P bots attempts possess a much greater number of packets sent to one destination (i.e., flood) whereas port sweeps scanners have one or few attempts towards one destination (i.e., probe).
4.4.2 Create Feature-based Time Series

We are characterizing and forecasting attacks’ impact features, namely, intensity and size. In this step, we create time-series data for each of the features. Here lags are created by shifting data points in time series. Lags are very useful in time series analysis because of a phenomenon called autocorrelation, which is a tendency for the values within a time series to be correlated with previous copies of itself. One benefit of autocorrelation is that we can identify patterns within the time series, which helps in determining seasonality, the tendency for patterns to repeat at periodic frequencies. Algorithm 4.1 shows how we create a lagged time series for forecasting.

**Algorithm 4.1** Generate lagged time series for forecasting.

```plaintext
Data: DS: Input data frame, unit: Forecasting time. Set to 1.
Result: x: Training data set, y: Target data set.

x = DS
lags = Duration in number of hours in training data to be selected to do the forecasting.
i = 0

for each i > -24 do
  lags[i] = x.shift(i)
i = i - 1
end

y = DS data frame including x’s indexes.
y[0] = x.shift(-(unit + 24))

msk = Mask which includes null indexes of both x and lags
x = Apply msk to lags.
y = Apply msk to y.
```

4.4.3 Apply Transformation

Time series datasets may contain trends and seasonality as in Figure 4.3, which may need to be removed prior to modeling. Trends can result in a varying mean over time,
whereas seasonality can result in a changing variance over time, both of which define a time series as being non-stationary. Stationary datasets are those that have a stable mean and variance and are in turn much easier to model. To make time-series stationary we apply commonly used differencing methods. Figure 4.6 shows the intensity of Zeus botnet after making series stationary.

![Intensity of Zeus bot with stationary series](image)

**Figure 4.6 Intensity of Zeus bot with stationary series**

### 4.4.4 Validate Results

Finally, to perform the forecasting, we apply different types of forecasting techniques, namely, moving average, weighted moving average, and RNN. We have selected to leverage these techniques instead of other complex well-known models such as ARIMA since the latter require long-term (weekly, monthly, yearly, etc.) time series [58] which is not the case with our data set.
**Moving Average (MA):** The single parameter of the model is estimated as the average of the previous $x$ data points at time $t$ in the time series. The MA is given by:

$$
\hat{x}_{t+1} = \frac{1}{k} \star (x_t + x_{t-1} + \cdots + x_{t-k})
$$

where $k$ is the smoothing window or period.

**Weighted Moving Average (WMA):** This technique is based on a numeric value known as the weight. In general, a WMA is more responsive to change in the time series data than a simple MA. The computation of the WMA estimated temporal average is given by [59].

$$
\hat{x}_{t+1} = \frac{w_{t-k}x_{t-k} + \cdots + w_t x_t}{h}
$$

where $k$ is the chosen window size and $h$ is the sum of the temporal weight, $h = w_{t-k} + \cdots + w_t$. In general, to obtain better results, the highest weight is given to the most recent periods.
4.5 Experimental evaluation

4.5.1 Experimental Environment

The experiments are conducted on an Amazon cloud with core with 4 vCPU, 16 GB RAM and on Linux operating system. S3 storage is used to store data series in JSON format. Hyperparameter tuning jobs and model deployment were done using Amazon SageMaker [60].

4.5.2 Experimental Dataset Generation:

We construct a training dataset from 24 hours of P2P traffic [5]. The evaluation of the RNN model on the training dataset would result in a biased performance. Therefore the model is evaluated on the held-out sample to give an unbiased estimate of the model. This extra dataset (validation dataset) which is held back from training data used to give an estimate of model performance while tuning model’s hyperparameters like the number of layers, number of nodes in each layer, learning rate, etc. From the training dataset, every tenth observation is sequestered into a validation dataset. We use the validation dataset to evaluate the performance and accuracy of the model over the course of the training process. For training data, we consider the 18hrs of data with a 1-minute sampling size. The remainder of the data is sequestered into a testing dataset.

The initial weights for each model network are randomly distributed. Additionally, we implement a regularization technique called `dropout' in which, for each step in the training process, the output of a randomly selected subset of nodes are ignored. This limits the over-dependence of the model on any one node and thereby reduces the potential for overfitting.
As a consequence of the stochasticity inherent to the model training process, repeated runs of the same model will yield trained networks that vary in their weights and, consequently, in forecasts. To accommodate this variance, we train 20 instances of each model. This allows us to assess expected model performance as well as assess the variance in performance across repeated runs of the same model.

We target 1, 2, 4, and 6-hour forecast horizons for the intensity and size of the attack. For each forecast horizon, we train each of the three models presented above. To evaluate the performance of the prediction methods, we compute the mean absolute error (MAE).

\[
MAE = \frac{1}{n} \sum_{i=1}^{n} |x_i - x|
\]

where,

\(n = \) the number of errors

\(|x_i - x| = \) the absolute errors

This error metric is defined as the absolute difference of the predicted value from the actual value divided by the actual value. Table 4.2 shows the MAE for the attack’s intensity for different forecasting horizons. Table 4.3 shows the MAE for the attack’s size for different forecasting horizons.
Table 4.2 MAE for the attack’s intensity for different forecasting horizons

<table>
<thead>
<tr>
<th>Horizon</th>
<th>MA</th>
<th>WMA</th>
<th>RNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1hr</td>
<td>.27</td>
<td>.25</td>
<td>.17</td>
</tr>
<tr>
<td>2hr</td>
<td>.48</td>
<td>.45</td>
<td>.30</td>
</tr>
<tr>
<td>4hr</td>
<td>.74</td>
<td>.63</td>
<td>.42</td>
</tr>
<tr>
<td>6hr</td>
<td>1.01</td>
<td>.87</td>
<td>.63</td>
</tr>
</tbody>
</table>

Table 4.3 MAE for the attack’s size for different forecasting horizons

<table>
<thead>
<tr>
<th>Horizon</th>
<th>MA</th>
<th>WMA</th>
<th>RNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1hr</td>
<td>.23</td>
<td>.18</td>
<td>.20</td>
</tr>
<tr>
<td>2hr</td>
<td>.39</td>
<td>.35</td>
<td>.34</td>
</tr>
<tr>
<td>4hr</td>
<td>.81</td>
<td>.52</td>
<td>.48</td>
</tr>
<tr>
<td>6hr</td>
<td>1.12</td>
<td>.83</td>
<td>.67</td>
</tr>
</tbody>
</table>

We can notice that the RNN model is most effective for the forecasting attack intensity and the size of the attack. The RNN model continues to demonstrate a substantial reduction in mean absolute error, compared to MA and WMA.

Figure 4.7-4.10 illustrates the attack’s intensity and prediction distributions for forecasting horizons. In order to preserve the clarity, we only present the first hour of prediction from a 6-hour prediction length. According to figure 4.7-4.10 attack peaks around 350 intensity at the 30th minute. The predicted values of such distribution show insights into an increase in the intensity of the attacks. From the figures, we can see that RNN predicts close the actual intensity time series for all forecasting horizons.
Figure 4.7 Attack’s intensity and prediction distributions for forecasting horizon 1

Figure 4.8 Attack’s intensity and prediction distributions for forecasting horizon 2
Figure 4.9 Attack’s intensity and prediction distributions for forecasting horizon 2

Figure 4.10 Attack’s intensity and prediction distributions for forecasting horizon 4
Figure 4.11-4.14 illustrates the attack’s size and prediction distributions for forecasting horizons. According to the figures, attack peaks around 19 hosts at the 19th minute. At forecasting horizon 1, MA and WMA models perform similarly. All three models remain competitive with the forecasting at the second forecasting horizon. Beyond the second forecasting horizon, the RNN model continues to outperform MA and WMA models.

Figure 4.11 Attack’s size and prediction distributions for forecasting horizon 1
Figure 4.12 Attack’s size and prediction distributions for forecasting horizon2

Figure 4.13 Attack’s size and prediction distributions for forecasting horizon3
It should be noted that the generated inferences from the above case studies aim to better understand the scale and rate of DDoS attacks that could be adopted by organizations for immediate response and hence mitigation as well as accumulated by security operators, emergency response teams. Also identifying any patterns of the attack helps to block the requests with firewall.
CHAPTER 5. CONCLUSION

This chapter concludes the dissertation and explains our future research road-map.

5.1 Conclusion

In botnet detection, we propose a novel way of detecting communities in dynamic networks on Spark. Our major contributions in this paper include (a) the improvement of the community detection algorithm efficiency and (b) the implementation of a scalable dynamic community model in a distributed environment. The proposed system is capable of detecting P2P botnets through a mutual contact graph-based botnet community detection approach. We introduce an algorithm for dynamic community detection by extending the original Girvan-Newman algorithm and benchmarking our results with the results of the original algorithm. It is important to note that we are not aiming to improve the original algorithm in terms of accuracy.

In botnet forecasting, the aim is to provide the organization under attack the capability to comprehend the situation and hence adaptively respond to the threat. We characterize and forecast the attacks’ impact features, namely, intensity/rate (packets/sec), and size (number of used compromised machines). Our proposed approach leverages various time series analysis and forecasting methods, including, moving average, weighted moving average, and Recurrent Neural Networks. We evaluate the proposed approach using real P2P botnet traffic. Furthermore, we provide the ability of forecasting botnet activities in both long term and short term time periods.
5.2 Future work

Our plans for extending the current works lie in several directions as follows.

- With respect to botnet detection, there are limitations to our current approach that we hope to resolve in our future work. First, we recognize that our detection technique is based on the availability of existing malicious data and that in order for a detector to be truly robust we must develop a mechanism to evolve the classifiers to adapt to new threats. Our botnet detection works mainly by identifying the mutual connection between the bots and legitimate hosts. We are aware that it is possible for a malicious botnet designer to obfuscate the network flow behavior of a bot in order to evade detection, even if such evasion would come at the expense of the effectiveness of a bot. To address these concerns, we are looking into the development of hybrid detectors that utilize evolving classifiers along with our current approach.

- The run time of our algorithm could be an issue when we dealing with large data sets. Therefore we are looking into more efficient clustering techniques along with group behavior correlation as our possible future direction. Testing on more datasets would be also beneficial in this aspect going forward.

- In relevant to the forecasting model, there are a few ways that we expect to improve model performance as we continue to develop. First, we anticipate that the inclusion of additional information as model inputs will improve the model considerably. In this work, we have deliberately restricted ourselves to the use of a single series as the basis for model inputs. The addition of other features which are correlated to features currently we forecasting should provide more information for model forecasts.
Furthermore, we expect that employing more advanced architectures will provide increased performance. Moreover, deep learning models are undergoing rapid development, with new techniques published every week. As new techniques emerge, we expect that we will be able to integrate them into the networks presented above to further improve performance by implementing our proposed approach in a real-time fashion.
REFERENCES


