Algorithms for Network-Based Misuse Detection

Baris Coskun

June 10, 2010
ALGORITHMS FOR NETWORK-BASED MISUSE DETECTION

DISSERTATION

Submitted in Partial Fulfillment

of the Requirements for the

Degree of

DOCTOR OF PHILOSOPHY (Electrical Engineering)

at the

POLYTECHNIC INSTITUTE OF NEW YORK UNIVERSITY

by

Baris Coskun

June 2010

Approved :

______________________________
Department Head

Copy No. ______________________
Microfilm or other copies of this dissertation are obtainable from

UNIVERSITY MICROFILMS
300 N. Zeeb Road
Ann Arbor, Michigan 48106
Vita

Baris Coskun was born in Antalya, a beautiful city on the Mediterranean coast of Turkey. He received B.S. and M.S. degrees in Electrical Engineering from Bogazici University, Istanbul, Turkey in 2001 and 2004, respectively. During his M.S. studies, he worked as a Software Engineer at Oksijen Technologies and Argela Technologies, where he was involved in the design and development of several core network components for GSM networks.

In 2005, Baris Coskun started working towards his Ph.D. degree at Polytechnic Institute of NYU. In 2006 and 2007, he was a summer intern at Microsoft Research, where he worked on multimedia security and challenge-response human authentication, respectively. His research interests are Network and Internet Security, Efficient Algorithms for Large Data, Multimedia Security and Signal Processing for Security.
To Senem.
Acknowledgements

My journey at Polytechnic Institute of NYU has been long, mostly joyful but sometimes winding and dark. Along the way, I have been fortunate enough to have many great people not only supporting and guiding me but also sharing the joy with me.

I thank my advisor Nasir Memon for the invaluable support and guidance he has provided me since the very beginning of my graduate studies. He has been an excellent source of advice both in technical, professional and philosophical sense, for which I am extremely grateful.

I would like to thank the valuable members of my defense committee Shivendra Panwar and Yong Liu for their helpful feedback and being valuable members of my defense committee. I would also like to thank many other excellent professors at Polytechnic Institute of NYU, from whom I learned a lot.

School was not the only place that I learned something along the way. I thank Cormac Herley and M. Kivanc Mihcak for mentoring me during my internships at Microsoft Research and helping me to develop skills to be a better researcher.

I would like to mention my advisor at Bogazici University, Bulent Sankur and my colleagues at Argela Technologies and Oksijen Technologies; Okan Cimen, Seyhan Civanlar, Reha Civanlar, Bora Eliacik, Ayse Erinc, Bulent Kaytaz, Erhan Lokman, Umit Sair for guiding me in the early years of my career and encouraging me to pursue my doctorate degree. I am most thankful for their continuing support.

My life as a graduate student would undoubtedly have been a miserable one without many friends surrounding me. In particular, I would like to thank Huseyin Akcan and Utku Irmak not
only for being good friends, but also for their invaluable help when I first arrived at this country; Deniz Gunduz for the deep and entertaining conversations we had on numerous diverse topics; and Paul Giura for discussing deep research topics while casting spinners for trout. I would also like to thank my dear friends Ozgu Alay, Kagan Bakanoglu, Sevinc Bayram, Cagdas Bilen, Sinteyahu Dehnie, Emir Dirik, Liyun Li, Ozgur Ozkan, Vikram Padman, Onur Sahin, Taha Sencar, Kulesh Shanmugasundaram, Yagiz Sutcu, Umut Topkara and all the present and the past members of Information Systems and Internet Security (ISIS) Lab at Polytechnic Institute of NYU.

I would like say many thanks to my parents Sebahat Coskun and Hamza Coskun, and my sister Gizem Coskun for their endless support and encouragement which always gave me strength to get through this endeavor. Despite an ocean and a continent between us, I have always felt their warm presence by my side.

Finally, I am utterly grateful to my wife Senem Acet Coskun, for her indescribable support, patience, loving and caring. I am extremely fortunate to have her with me at every single moment of my life since before I was a graduate student. Without her, neither this thesis nor myself would ever be complete.

Baris Coskun, Polytechnic Institute of NYU

June 10, 2010
Abstract

Increasingly high volume and variety of Internet attacks are encountered every day due to attackers’ strong financial motivation. Most of such malicious activities, such as Spamming, Phishing, Distributed-Denial-of-Service (DDoS), etc., involve numerous compromised computers dispersed over geographically diverse networks. As a result, Internet attackers often have strong incentives to compromise as many computers as possible around the globe. Clearly, such compromised computers pose a significant threat both to the networks they reside in and to all Internet users. Therefore, it is in the best interest of an organization to detect compromised computers and malicious activities within its network.

In this thesis, we consider network-based misuse detection in an enterprise network, which is defined as the collection of all network components including computers, servers, routers, subnets, etc., under the jurisdiction of a single organization. We particularly focus on three different malicious activity detection problems; The first problem is detection of relay nodes, which are often used by attackers to conceal their identities. For this problem, we present an online algorithm, which efficiently detects relay nodes in a network. Second, we consider online detection of correlated network flows. The algorithm we propose for this problem can be used in attack source attribution when attackers hide behind a series of relay nodes. Finally, completely different from the first two, the third problem we consider in this thesis is the detection of the members of a Peer-to-Peer (P2P) botnet in an enterprise network. For this problem, we present a simple graph based algorithm which identifies additional P2P bots in a network once a single bot exposes itself by sending spam or perform a network scan, etc.
In general, to detect a malicious activity, a network-based misuse detection scheme monitors network traffic for certain network traffic patterns which emerge due to that particular malicious activity. One of the biggest challenges in this process is that, high volumes of network traffic corresponding to thousands of computers has to be monitored efficiently. Therefore, algorithms employed by network-based misuse detection schemes have to be simple and fast. Hence, in this thesis, we place a great emphasis on computational efficiency and scalability.
Contents

Vita iv

Acknowledgements vi

Abstract viii

1 Introduction 1

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.2 Intrusion Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2.1 Host-Based Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2.2 Network-Based Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Network-Based Intrusion Detection . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.3.1 Anomaly-Based Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.3.2 Misuse-Based Detection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.4 Thesis Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.5 Organization of the Thesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2 Network Patterns for Misuse Detection 9

2.1 Characterizing a Misuse . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
2.2 Taxonomy of Network Traffic Patterns . . . . . . . . . . . . . . . . . . . . . . . . 11
2.2.1 Packet-Level Patterns . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.2.2 Flow-Level Patterns . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
4.6 Limitations And Potential Solutions ........................................... 71
4.7 Extension to VoIP Traceback ....................................................... 72
  4.7.1 Related Work ...................................................................... 74
  4.7.2 Robust Hashing of VoIP Flows .............................................. 74
  4.7.3 Experiments ....................................................................... 79
4.8 Conclusion .............................................................................. 80

5 P2P Botnet Detection ................................................................. 82
  5.1 Introduction ........................................................................... 82
  5.2 Related Work ........................................................................ 86
  5.3 Finding Friends of An Enemy .................................................... 87
    5.3.1 Basic Idea ....................................................................... 87
    5.3.2 The Proposed Scheme ....................................................... 88
    5.3.3 The “Mutual-Contacts” Graph ............................................ 90
    5.3.4 The “Dye-Pumping” Algorithm ........................................... 91
  5.4 Experiments ........................................................................... 93
    5.4.1 Detecting Nugache Peers ................................................... 93
  5.5 Why “Dye-Pumping” Works: A Mathematical Analysis .............. 98
    5.5.1 Random Peer Selection Model .......................................... 99
    5.5.2 Friends Stay Closely Connected (Erdős-Rényi Subgraphs) .... 100
  5.6 Limitations and Potential Improvements .................................... 103
  5.7 Conclusion and Future Work ................................................... 105

6 Conclusions and Future Work .................................................... 107

A One Dimensional Random Walk ............................................... 120

B Probability of Accessing a Given Node in Erdős-Rényi Graphs .... 122
List of Figures

2.1 Taxonomy of network traffic patterns ........................................... 11

3.1 Network pattern of a relay node ................................................... 24
3.2 A typical relay traffic used in experiments ................................. 31
3.3 Performance under Mail Server traffic ......................................... 33
3.4 Performance under Mail Server traffic ......................................... 35
3.5 Effect of number of relays ......................................................... 37
3.6 Effect of relay duration ............................................................ 39
3.7 Effect of chaff packets ............................................................. 40
3.8 Estimation of $\Psi$ for $T = 100$ ............................................... 45
3.9 Estimation of $\Psi$ for $T = 1000$ ............................................... 46
3.10 Histogram of $\Psi$ with no relay activity ..................................... 47

4.1 Packet-count vectors ............................................................... 57
4.2 Bit-Error Probability upper bound ............................................ 61
4.3 Probability of finding a correlated flow-pair ................................ 65
4.4 The empirical probability distribution ....................................... 67
4.5 Detection rates ................................................................. 68
4.6 Effect of time-slot length ......................................................... 69
4.7 Effect of sketch length .......................................................... 70
4.8 Effect of correlated flow duration ............................................. 71
List of Tables

3.1 False positive rates .......................................................... 37
3.2 Detection performance for web server traffic .......................... 47
3.3 Detection performance for computer lab traffic ....................... 48

5.1 Summary of observed P2P botnet behavior ............................ 103
List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Calculate $S$</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Estimate $\sigma^2$</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Calculate $S_{scrambled}$</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>MaintainSketches</td>
<td>63</td>
</tr>
<tr>
<td>4.2</td>
<td>$h = \text{Compute}_Hash$</td>
<td>78</td>
</tr>
<tr>
<td>5.1</td>
<td>$Dye_{Pumping}$</td>
<td>93</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Financial gain has become a major motivation behind most of the malicious activities on the Internet, such as Spamming, Phishing, Distributed-Denial-of-Service (DDoS), etc. Hence, it is not surprising to encounter increasingly high volume and variety of Internet attacks each year [22]. As a result, detecting and preventing such malicious activities are gaining more significance everyday.

Most of the malicious activities observed in today’s cyber-world involve numerous compromised computers dispersed over geographically diverse networks. For instance, attackers often employ several compromised computers to circulate unsolicited emails, called Spam, or to host Phishing websites. Similarly, a DDoS attack generally requires a high number of compromised computers generating network traffic towards a victim server. In a different scenario, attackers use several compromised computers within different networks as stepping-stones in order to conceal their identities while intruding target networks or servers. In this strategy, often known as the stepping-stone attack, the traffic between an attacker and a victim is relayed through a chain of intermediate computers so that the victim can only deduce that the attack is coming from the last intermediate computer on the chain.

Due to such distributed malicious activities, attackers often have strong incentives to compromise as many computers as possible around the globe. Therefore, it is often the case to find several compromised computers in a given network.
Despite substantial efforts of an organization to protect its network, attackers can still compromise many of the computers in the organization’s network by exploiting countless software vulnerabilities and/or by tricking users into executing malicious programs [101] [65] [37] [75] [69] [74]. Obviously such compromised computers pose a significant threat both to the organization itself and to all Internet users. Therefore, it is in the best interest of an organization to detect compromised computers and malicious activities within its network.

Motivated by this, in this thesis we consider, at a high level, the problem of detecting compromised computers and malicious activities in an enterprise network, which is defined as the collection of all network components including computers, servers, routers, subnets, etc., under the jurisdiction of a single organization. We particularly consider three different malicious activity detection cases, namely relay node detection, correlated flow detection and peer-to-peer botnet\(^1\) detection.

In the remaining of this chapter, we first discuss different strategies proposed to address the general malicious activity detection problem. Following that, we elaborate the three problems we consider in this thesis and summarize our contributions. Finally we present the organization of the thesis.

### 1.2 Intrusion Detection

The practice of detecting compromised computers and malicious activities by monitoring network traffic and/or host activities is known as Intrusion Detection [78]. Depending on the type of their input data, Intrusion Detection Systems (IDS) are basically classified into two groups [30] [10]:

#### 1.2.1 Host-Based Systems

A Host-Based Intrusion Detection System (HIDS) monitors the internal activities and states of an individual computer (host) [97]. Typically an HIDS resides on the host computer that is being monitored, where it has access to low level operations such as disk operations, memory operations, key strokes, etc. Therefore, an HIDS can compile semantically rich infor-

\(^1\)A botnet is defined as an overlay network of compromised computers under the control of an attacker. Further details can be found in Chapter 5
information about the host which cannot be observed remotely from the network. On the other hand, running on the host computer makes an HIDS vulnerable to being disabled or tainted by an attacker after compromising the host. For instance, a version of Conficker Botnet blocks the access of compromised hosts to security related websites so that HIDS programs cannot receive security updates [40].

1.2.2 Network-Based Systems

A Network-Based Intrusion Detection System (NIDS) monitors network traffic for traces of malicious activities. In a simple architecture, a NIDS collects data from a single sensor placed at the network’s edge router, where all ingress and egress traffic of the network can be seen. For larger networks, several sensors may be placed at different points in the network (i.e. switches, routers, etc.) in order to collect more thorough and higher resolution information. For instance, all of the hosts behind a Network Address Translation (NAT) router appear as a single node from an edge router’s point of view, whereas an additional sensor behind or on the NAT router allows a NIDS to differentiate such hosts from each other.

While HIDSs have proved to be efficacious in many scenarios, network-based approach provides several benefits which cannot be achieved by host-based systems. Some of these benefits can be listed as follows:

- **Passive and Transparent Monitoring:** A NIDS passively monitors the hosts without interfering with the network traffic. Therefore, attackers cannot sense if and how their actions are being monitored. Furthermore in a NIDS, since there is no software running on individual hosts, attackers have no means of disabling or tainting the intrusion detection mechanism even if they gain complete control of a host.

- **Aggregation and Correlation:** A NIDS can collect information from all of the hosts in a network, thereby making various correlation analysis possible. For instance, as mentioned earlier, significant portion of the Internet attacks are carried out by networks of compromised computers called botnets. Members of a botnet, usually referred to as bots, often coordinate with each other and behave similarly. By aggregating information from all of the hosts in the network for an extended period of time, a NIDS can potentially
identify the members of a botnet by exploiting the similarities that they may exhibit in their network traffic.

• **Uniform Monitoring:** If each host in the network is monitored individually, some hosts in the network might be monitored more closely than others due to the precautions taken by their administrators. Similarly, careless administration may result in some hosts being not monitored at all. Network-based monitoring of hosts allows all the hosts in the network to be monitored uniformly at an equal level. Hence, no host in the network is left unattended due to lack of individual administrative efforts when a NIDS is deployed.

• **Easy-to-Manage:** A malicious activity detection scheme should be improved and updated frequently since Internet attacks are evolving constantly. Managing and updating a network-based scheme is extremely simple since, instead of being distributed over every host in the network, essentially a central point is responsible for executing the detection mechanism.

• **Security-as-a-Service:** The concept of Security-as-a-Service, where external parties offer network-based security services to enterprise networks, has been gaining significant attention recently. For instance, along with providing Internet access, Internet Service Providers can potentially provide network security measures as a service to create a new stream of revenue. On the other hand, by enrolling a network-based security service, an enterprise network can achieve high-level of network security without requiring an in-house network administrator carrying out sophisticated security procedures.

Similarly, the host-based approach has several advantages as well, such as being able to access detailed and reliable information about a host. In fact, a complete protection of an enterprise network should include both NIDSs and HIDSs operating simultaneously. In this regard, there has been several attempts to integrate the network-based and host-based approaches [103] [7] [35]. In this thesis, we concentrate on the network-based side of the intrusion detection problem and further details of the host-based approach is out of our scope.
1.3 Network-Based Intrusion Detection

Regarding how the input data is analyzed, network-based intrusion detection schemes can be basically divided into two categories [30] [10]:

1.3.1 Anomaly-Based Detection

In summary, anomaly-based NIDSs first estimate the “normal” behavior of the hosts in a network during a training phase [43]. Here, the “normal” behavior refers to the network behavior of the hosts when there is no malicious activity. Then, they monitor the network traffic for behaviors which significantly deviate from the estimated “normal”. Such anomalous behaviors are flagged as malicious and reported.

To estimate the normal behavior, various features are extracted from the training network data in an anomaly-based NIDS. Some examples of such features can be number of packets per unit time, number of bytes per unit time, number of flows per unit time, average flow duration, etc. [66]. Typically, the dynamics of these features under the normal traffic are characterized by statistical models and various machine learning techniques are employed to differentiate anomalous traffic from the normal.

Obviously, accurate estimation of the normal behavior is crucial for anomaly-based NIDSs. For this purpose a clean set of network data, free of malicious activity, is required. Otherwise, malicious activities, which are included in the training data, would be treated as “normal”, thereby causing anomaly-based NIDSs to overlook such activities.

It is often very hard to confirm that a set of training data is free of malicious activity. This is one of the major issues that challenge anomaly detection schemes. In this respect, some techniques have been proposed to sanitize the training data [27].

Once an anomaly-based NIDS accurately characterizes the normal network behavior, it can potentially detect malicious activities even though they may not be previously known to exist. This alone is a huge motivation for network administrators to employ anomaly-based systems. However, various non-malicious activities are often classified as malicious as the training data may not contain such activities. Therefore, anomaly-based NIDSs often suffer from high false-positive rates.
1.3.2 Misuse-Based Detection

Unlike anomaly-based NIDSs, misuse-based NIDSs look for specific network patterns, which occur due to malicious behaviors. In order for a misuse-based based detection of a certain malicious activity, first the network-traffic pattern which characterizes that particular malicious activity has to be determined. For instance, a compromised host, which scans a network for a specific vulnerability, can be characterized by its several connection attempts to several hosts within the network using a specific port number [102]. Once a malicious activity is appropriately characterized by such a pattern (sometimes referred to as a signature or a rule), the misuse-based NIDS start monitoring the network traffic to see if any of the hosts exhibits that pattern.

The process of characterizing a malicious activity often requires human knowledge and manual analysis. Therefore misuse-based detection schemes are sometimes referred to as expert systems.

Although it is obvious that a misuse-based approach can only detect previously known malicious activities, they are often very efficacious as they usually provide high detection accuracy. In fact, many of the NIDS which are widely used in practice today, such as Snort [6] and Bro [3], are all misuse-based systems. Nevertheless, the accuracy of a misuse-based NIDS ultimately boils down to how accurately the network patterns characterize malicious activities as discussed in Chapter 2.

1.4 Thesis Contributions

In this thesis, we consider three particular network-based malicious activity detection problems in an enterprise network setting; The first problem is relay node detection, which is tightly related to the detection of stepping-stone attacks mentioned earlier since intermediate hosts on a stepping-stone chain are essentially some kind of a relay node. Second we concentrate on the detection of correlated network flows within an enterprise network. Here, a network flow is defined as the collection of packets flowing from the same source to the same destination using the same source and destination port numbers and the same transport layer protocol. Since the flows between the intermediate hosts on a stepping-stone chain has to be correlated in
some sense, a correlated flow detection scheme would potentially enable stepping-stone attack attribution. Finally, completely different from the first two, the third problem we consider in this thesis is the detection of the members of a Peer-to-Peer (P2P) botnet.

To address these problems, we first determine the network traffic pattern that accurately characterizes the corresponding malicious activity for each problem. Then we design an algorithm, which monitors a given enterprise network traffic and reports when the corresponding pattern is detected. Due to these steps, our approach in this thesis can be classified as network-based misuse detection.

Network-based misuse detection systems are often required to monitor high volumes of network traffic created by thousands of hosts. Furthermore, online detection of some malicious activities are crucial for stopping the underlying attacks before causing any harm. For that reason, we place a great emphasis on computational efficiency in this thesis. Consequently, major contributions of this thesis are in terms of simplicity and computational efficiency. In summary, the contributions of this thesis are:

- We propose a relay node detection algorithm, which is asymptotically faster than previously proposed algorithms, thereby allowing online detection of relay nodes in large networks.

- Similarly, we propose a correlated flow detection algorithm, which is asymptotically faster than previously proposed algorithms, thereby potentially enabling online attribution of stepping-stone attacks.

- We propose a P2P botnet detection scheme, which allows network administrators to quickly identify additional P2P bots in their networks once a single P2P bot exposes itself by sending spam or perform a port scan, etc.

1.5 Organization of the Thesis

In Chapter 2, we elaborate on network patterns which are used in network-based misuse detection schemes. More precisely, we first present few properties and a taxonomy of network patterns. Then we discuss different pattern detection strategies in a network traffic.
In Chapters 3, 4 and 5, we present our proposed solutions to the relay node detection, correlated flow detection and P2P botnet detection, respectively. In each of these chapters, we evaluate the efficacy of the proposed scheme both experimentally and theoretically.

Finally in Chapter 6, we present concluding remarks and future work.
Chapter 2

Network Patterns for Misuse Detection

Misuse-based detection of a certain malicious activity in a network comprises two basic steps; First a network traffic pattern, which characterizes that particular malicious activity, has to be determined. Following that, an algorithm which efficiently detects that pattern in the monitored network traffic has to be designed. In this chapter we elaborate on these steps. In Section 2.1, we begin with discussing several conditions required in order for a network pattern to properly characterize a malicious activity and therefore improve the overall misuse detection accuracy. Following that we present a taxonomy of network patterns in Section 2.2. Finally in Section 2.3, we present different approaches to detect these network patterns.

2.1 Characterizing a Misuse

In a misuse-based malicious activity detection scheme, determining a proper network pattern is crucial for the overall detection accuracy. In order to achieve good false-alarm and true-positive rates, the following issues has to be carefully considered:

Generalizability: The determined pattern should capture the essence of a generic class of malicious activity rather than representing a specific implementation. For example, if a misuse-based NIDS was to characterize compromised hosts based on the port numbers they use to communicate with each other, it would only be able to detect a particular instance of the culprit malware, as the malware could easily use different ports on different compromised hosts,
thereby evading detection. On the other hand, characterizing spammers in a network, based on their extensive usage of Simple Mail Transport Protocol (SMTP) would be a good example of a generalizable pattern, since spammers have to communicate with other mail servers over SMTP to deliver spam emails [36].

**Essentiality:** The determined pattern should be essential for an attacker to carry out the target malicious activity. That is, it should be infeasible for attackers or malware to avoid exhibiting that pattern in their network traffic while performing the target malicious activity. Otherwise, attackers would simply choose not to exhibit the pattern and hence trivially evade detection.

**Exclusivity:** The determined pattern should be exclusive to the target malicious activity. In other words, non-malicious activities should not exhibit that particular pattern. Otherwise such non-malicious activities would also be picked up by the detection scheme, thereby increasing the false-positive rate. For instance, even though Waledac botnet uses HyperText Transfer Protocol (HTTP) as its main communication protocol [82] [94], it is obvious that selecting “HTTP usage” as the network traffic pattern to detect Waledac bots is not a good idea at all, since a significant portion of the non-malicious Internet traffic is carried over HTTP.

Notice that if a misuse-based malicious activity detection scheme managed to completely meet the above requirements, then it would achieve perfect detection rate, such as 100% true-positive rate and 0% false-negative rate. That is, if a network pattern is completely exclusive and completely essential for a malicious activity, then the detection of that network pattern immediately and undoubtedly implies the presence of the corresponding malicious activity. However, in most cases, it is impossible to determine such network patterns which are completely exclusive and completely essential for a malicious activity, thereby preventing perfect detection rate. In those cases, the detection performance of a malicious activity detection scheme depends on how much general, how much essential and how much exclusive the used network patterns actually are.
2.2 Taxonomy of Network Traffic Patterns

Malicious activities may exhibit patterns on various network features at various levels. For instance, some activities may exhibit a certain binary pattern on the payload of individual Internet Protocol (IP) packets, whereas some malicious activities can be characterized by the number of Transmission Control Protocol (TCP) connections made by a host in unit time. Based on the level of underlying features, network traffic patterns used in misuse detection schemes can be classified into three basic groups, namely packet-level patterns, flow-level patterns and host-level patterns. In addition, network patterns can be further divided into subgroups at each level, depending on the type of the underlying network features. In the remaining of this section, we discuss each branch of this categorization, which is depicted briefly in Figure 2.1.

2.2.1 Packet-Level Patterns

At the lowest level, malicious activities may exhibit patterns on the features extracted from individual network packets. These features can be extracted from the header or the payload of a packet. Based on these features, packet-level patterns can be categorized into two classes:

Header-Based Patterns

Header-based patterns are the patterns observed in the header of a single packet. For instance, attackers sometimes fragment Internet Protocol (IP) packets into smaller pieces to
conduct a set of attacks, called IP Fragmentation Attacks, or to bypass some security measures [8]. This type of malicious activity often creates a certain pattern in the packet header information. More specifically, in the case of an IP fragmentation, "more fragments" flag in the IP header is set to 1 or there is an offset indicated in the offset field of the header [4]. Therefore, a misuse-based NIDS, which monitors the network for this specific pattern, can easily detect IP fragmentations.

**Payload-Based Patterns**

Payload-based patterns, on the other hand, are the patterns observed in the payload of a single packet. For instance, in some cases attackers has to send a certain message or provide a certain input to a server in order to exploit a particular vulnerability. Such messages appear as a specific sequence of bytes in the payload data. Hence, that specific byte pattern can be used to detect that particular exploitation attempt. This type of payload analysis is often known as Deep Packet Inspection (DPI). DPI is often computationally expensive, since the payload of every packet has to be compared against a database of known byte patterns corresponding to numerous malicious activities. To address this problem several, data structures and hardware-based solutions have been proposed [83] [12] [9].

Payload-based patterns can be extremely efficacious in characterizing malicious activities. However, it is obvious that payload-based patterns are limited only to malicious activities conducted in plain-text. Consequently, to reduce the chances of being detected, attackers are increasingly employing various encryption techniques. For instance, most of the recent botnets, such as Storm [48,54], Waldeac [82], Nugache [88], etc., transmit their command and control traffic under heavy encryption.

### 2.2.2 Flow-Level Patterns

One level up from network packets, there are network flows. Basically, a network flow is a group of network packets which share the same five-tuple of source IP address, source port number, destination IP address, destination port number and layer 3 protocol type (i.e, UDP,TCP). For instance, all the packets transmitted from a particular source to a particular destination during a single TCP session constitute a network flow.
Several features can be extracted from network flows. Certain patterns on these features are often used to characterize some kinds of malicious activities. We refer to such patterns as flow-level patterns. Basically there are two types of flow-level patterns:

**Statistical Patterns**

Several flow features can be extracted by computing various statistics from a network flow. In the simplest case, counting various events during a lifetime of a flow yields some interesting features such as, number of packets transmitted, number of bytes transmitted, etc. On the other hand, more complex features can be obtained by applying basic statistical operations such as, average number of packets, maximum packet length observed, etc. In some cases, even more complex features can be computed using higher level moments, order statistics, entropy, etc. The patterns on these statistical flow features often play an important role in identifying malicious activities. Therefore, several commercial products [1] [2] have been designed to compute and collect such flow-level features.

**Evolutionary Patterns**

In some situations, statistical features are not enough to capture the characteristics of a malicious activity within a flow. In those cases, the evolution of flow features over time might help. For instance, a compromised host participating in a stepping-stone attack has to relay the packets coming from the preceding intermediate node towards the succeeding one on the stepping stone chain. As a result, a pair of network flows (i.e. one incoming flow and one outgoing flow) in that host’s traffic has to transmit packets at similar times. To detect this kind of activity, a misuse detection scheme has to compare the packet timings of incoming flows to outgoing flows to see if there is any pair of flows which have similar packet timing patterns. Obviously, to achieve this, the misuse detection scheme has to keep track of the patterns and variations of each flow’s packet timings over a period of time. We refer to this kind of patterns, which are often in the form of time-series, as evolutionary patterns.
2.2.3 Host-Level Patterns

At the highest level, there are hosts, where a host’s traffic is composed of the network flows transmitting packets from and to that host. Some malicious activities can be characterized by patterns observed on this host-level network information. We refer to such patterns as host-level patterns.

Host-level patterns can be divided into two categories:

Statistical Patterns

Several host-level statistics can be extracted from a host’s network traffic. Similar to flow-level statistics, computation of host-level statistics varies from basic counting to complex statistical operations, such as high order moments, entropy, order statistics, etc. Some examples of host-based statistics can be; average number of flows per unit time, average number of bytes transmitted per unit time, distribution of remote port numbers that the host connects, number of bytes that the host transfers over a specific port, etc. Some malicious activities exhibit certain patterns on such host-level statistics, thereby getting caught by misuse detection systems. For instance, a spammer often transmits large amount of data over port 25, which is reserved for SMTP. Hence, a misuse detection scheme can identify spammers in a network by looking at the volume of each host’s outgoing traffic volume on port 25.

Conversational Patterns

Conversational patterns on a host’s traffic are essentially related to whom the host communicates with. Certain malicious activities, especially botnet activities, exhibit distinctive conversational patterns. Conversational patterns can be categorized into two groups:

i. Individual Conversational Patterns: Individual conversational patterns are observed in the network traffic of individual hosts without requiring any correlation or aggregation over several different hosts. For instance, some infected hosts communicate with other infected hosts which are previously identified and listed in a publicly available blacklist, such as DShield [5]. Hence, one can easily identify such infected hosts by looking at whether they have communicated with a blacklisted host. As another example, a network-based
misuse detection scheme can detect peer-to-peer activity of a host by exploiting another
distinctive conversational pattern, which is communicating with several remote hosts at
geographically diverse locations within a short amount of time.

ii. Group-based Conversational Patterns: On the other hand, group-based conversa-
tional patterns are collectively exhibited by a group of hosts. Therefore, in order to ob-
serve and detect group-based conversational patterns, one has to aggregate the network
traffic of several hosts. For instance, members of a botnet with centralized command and
control architecture communicate with a common set of control servers [11]. As a result,
a network-based misuse detection scheme can aggregate the conversational information
from all hosts in the network and identify members of such a botnet by exploiting the
fact that they collectively communicate with the same set of servers. As another example,
members of a P2P botnet exhibit some distinctive patterns on a graph where the nodes
represent hosts and an edge between two nodes implies that the two corresponding hosts
have communicated with each other [70]. More specifically, members of a P2P botnet
form various structured subgraphs on such communication graphs depending on the peer
selection mechanism they use. As a result, one can identify the members of a P2P botnet
by detecting such structured subgraphs.

2.3 Detecting Network Patterns

Once an appropriate pattern that characterizes a certain malicious activity is determined,
a misuse-based NIDS can begin monitoring network traffic. In order to detect occurrences of
a pattern in network traffic, a misuse-based NIDS has to implement some sort of a pattern
detection algorithm. Depending on the nature of the target pattern, various approaches can be
used in designing such pattern detection algorithms.

2.3.1 Fixed Pattern Matching

Some network patterns used in characterizing malicious activities have well defined and
deterministic properties. We refer to such patterns as fixed patterns. Byte patterns on packet
payloads or specific set of flags on packet headers are few examples of fixed patterns. In fact,
most packet-level patterns currently used in commercial misuse detection systems are all fixed patterns. Similarly, communicating with a blacklisted IP addresses can be given as an example of a host-level fixed pattern.

Detecting fixed patterns is often a very simple operation. Usually a basic string matching, masking or a template matching algorithm is sufficient to accomplish the task with very high accuracy. However, a significant challenge arises when gigabits of network traffic, that has to be inspected at every second, is considered. Nevertheless, deterministic and well-defined nature of fixed patterns allows extremely fast hardware-based solutions [83] [12] [9], which can process high-speed network traffic, to some extent.

2.3.2 Model-Based Pattern Matching

Some malicious activities exhibit network patterns which don’t have any particular deterministic properties but rather generated by a model. To detect such patterns in network traffic, a misuse detection scheme first determines the model that governs the target pattern. Then, it monitors the network traffic to detect any activity that fits to the model. For instance, a successful infection of a host is actually a result of a series of network events, such as; first the host is subject to an inbound scan, then it downloads a binary executable, and finally it performs an outbound scan, etc. [50]. This chain of events might be slightly different in different hosts, such as the chain may begin by clicking on a link in the browser for some cases. Furthermore, a whole set of unrelated events usually occur simultaneously and therefore it is not easy to recognize the chain of events, which lead to the infection, in the crowd. In such cases, a state transition model can be used to represent a successful infection [50]. Following that, a finite state machine-based analysis can be performed to detect target chain of events. Similarly, graph-based models can also be used to detect such malicious activities, which can be represented by a chain of events [84].

As another example, the information of who communicates with whom can be represented by a graph, where nodes represent hosts and an edge between two nodes means the corresponding two hosts have communicated with each other [56]. On such graphs, botnet communications often exhibit a distinctive pattern, thereby allowing a model-based analysis to discover several bots in a network, as discussed in Chapter 5.
A model-based detection scheme outputs probabilistic results since the underlying network pattern cannot be defined deterministically. That is, although it is highly probable, one can never be sure whether a malicious activity reported by a model-based detection scheme is indeed a genuine malicious activity.

2.3.3 Clustering and Correlation

In some misuse detection problems, especially in botnet detection, the network pattern that characterizes a malicious activity cannot be precisely determined. What is known instead is that, multiple instances of the same malicious activity will exhibit similar patterns on some network features. For instance, members of a botnet often exhibit similar flow-level patterns as they are essentially different instances of the same malware [49] [107]. In those cases, a clustering analysis can potentially exploit such similarities in the network traffic by grouping the malicious hosts together.

For another example, as mentioned earlier and as elaborated in Chapter 3, intermediate nodes on a stepping-stone chain have at least one pair of incoming and outgoing flows which have correlated packet timings. A misuse-based NIDS can exploit such evolutional correlations to potentially identify and stop stepping-stone attacks.
Chapter 3

Relay Node Detection

3.1 Introduction

In a typical enterprise network, there can be various situations where a node receives data from another node and forwards it to some others. Such nodes are often called “relay nodes”. Relay nodes can be employed for many different purposes which can be either legitimate or malicious. Routers and switches are clearly examples of usually legitimate relay nodes. But there are other more ambiguous situations. For instance, an enterprise may be running a legitimate peer-to-peer video streaming application for the benefit of its employees. On the other hand an employee could be violating company policy by running a peer-to-peer application to watch live television on the desktop. Similarly, a system administrator may be connected from home to a server and logs in from that server using ssh to one of the internal machines in order to check its status. This would be an example of a legitimate stepping stone connection. Stepping stones, however, are commonly used by attackers to make attack traceback difficult. In general, regardless of the original intention, relay nodes are a potential threat to networks since they are used in many malicious situations like stepping stone attacks, botnet communication, illegal peer-to-peer file sharing etc. Hence, quick and accurate detection of relay nodes in a network can significantly improve security policy enforcement.

Relay nodes can be divided into two main categories:

1) **Store & Forward Relay Nodes:** These type of relay nodes often store data before forwarding. Peer to peer file sharing and email relaying are some examples of store and forward
relays. The time elapsed before the data is forwarded depends on application requirements. For instance email relays forward received emails after few minutes whereas in peer to peer file sharing applications data is forwarded only when another user requests it.

2) Delay-Constrained Relays Nodes: These type of relay nodes forward the received data within a maximum tolerable delay constraint. The delay requirement is inherent in the underlying application. Delay-constrained relaying can be done by applications which are either interactive or machine driven. For instance, stepping stones and IM message routing nodes are some examples of delay-Constrained relays with interactive sessions. On the other hand, peer to peer live broadcast and Skype super-nodes are examples of machine-driven delay-constrained relays.

Detection of store and forward type of relays is generally done by identifying protocol features. Usually a target protocol is selected and its distinctive characteristics are identified. Subsequently, a node that exhibits such characteristics is declared as a relay node. Some such protocol features used by researchers include connections to known ports, some specific signatures in the payload, concurrent use of both UDP and TCP etc. This work will mainly focus on delay-constrained relays. Interested readers can find further details on store and forward relay node detection schemes in [44,61,68,71,81].

To the best of our knowledge, there is no prior work that focusses on the delay-constrained relay node detection problem. However, there has been much work on the closely related delay-constrained correlated flow detection problem [15,34,85,108,110]. The basic detection methodology in the proposed correlated flow detection schemes is to search for network flow pairs which exhibit strong mutual correlation. This correlation is determined based on various attributes of the flow, including packet content (payload), packet arrival times, packet lengths etc.

Although detection of correlated flows implies identification of relay nodes as well, correlated flow detection is computationally harder than relay node detection. In fact, a correlated flow detection scheme compares each incoming flow to each outgoing one, usually on a node by node basis. Therefore, a correlated flow detection scheme requires quadratic time operation in the number of flows for each node, which may be prohibitive for medium to large scale networks with tens of thousands of nodes and thousands of active connections in many nodes.
For many scenarios, however, instead of identifying correlated flows, identifying relay nodes could be sufficient to take appropriate action. In addition, a lightweight and scalable solution to the problem of delay-constrained relay node detection can serve as a first step in a scalable correlated flow detection solution that performs one of the known quadratic time analysis techniques for correlated flow detection only on nodes that have been determined to be relay nodes. This strategy brings significant computational efficiency to existing schemes since the quadratic flow detection algorithm is now applied to a few selected nodes rather than every node in the network. Hence, in this chapter we focus on network-based delay-constrained relay node detection. We reformulate this slightly relaxed problem as a variance estimation-based misuse detection. We employ a statistical approach to solve the problem in linear time, which makes it viable to incorporate the proposed scheme in large scale networks. Also the proposed scheme is robust, to some extent, against adversarial manipulations that change the time structure of the flows such as intentional delays or chaff packets. Part of this chapter has been published in [23].

The rest of the chapter is organized as follows. After discussing related work in the following section, the basic idea, formulation and the implementation of the proposed scheme is given in Section 3.3. Section 3.4 presents some experiments in order to evaluate the performance and Section 3.7 contains discussions and conclusions.

### 3.2 Related Work

Research on delay constrained relay detection has mostly focused on stepping stones due to their obvious potential malicious intention. Perhaps the first such technique was proposed by Staniford and Heberlein [85]. They proposed a content correlation based scheme where flow pairs are compared in terms of thumb-prints of their content. However, content based schemes have limited applicability since flows are usually encrypted and their contents are inaccessible. This fact motivated researchers to focus on layer 3 information which mostly consists of originating and destination IP addresses, packet arrival times etc. In the first work that incorporates layer 3 information [110], Zhang and Paxson detect stepping stones by correlating flows in terms of their ON and OFF periods. The assumption is that correlated
flows switch from OFF state to ON state at similar times. In [108], Yoda and Etoh propose a similar timing based algorithm where correlation is defined over sequence number vs. time curves of the flows. Another timing based algorithm is proposed by He and Tong in [53], where authors formulate the flow correlation problem as a nonparametric hypothesis testing. Other than stepping stones, in [90], Suh et al. proposed a similar timing based technique for detecting Skype related relay traffic.

Timing based methods usually fall short when the time structure of the relaying flows is perturbed by an attacker. This perturbation may be performed by means of introducing artificial delays before relaying the received packet or by adding chaff packets into the stream. In [34], Donoho et al. shows that if there is a maximum tolerable delay constraint, instead of using raw packet timing information, applying wavelet decomposition and analyzing packet timings in different (lower) resolutions will make the effect of the adversarial changes in time structure insignificant. Similarly under a maximum tolerable delay constraint, Blum et al. present confidence bounds on the stepping stone detection problem in [15]. As a completely different approach, in [99], Wang and Reeves propose a watermarking based approach where selected packet timings are slightly adjusted on all incoming flows. In order to identify a relaying flow, a watermark detection procedure is applied to all outgoing flows. Although this technique is a form of timing based flow correlation algorithm, it is shown to be robust against timing perturbations introduced by adversaries.

As we have pointed out before, flow correlation based techniques solve the problem in quadratic time. They need to compare each incoming flow to each outgoing flow. Therefore it is not easy to employ these methods in large networks. One could adopt filtering techniques to alleviate this problem to some extent. For instance, in [110] specific flow pairs are filtered out based on packet size, inconsistent source and destination ports, inconsistent packet direction, inconsistent packet timing etc. However, discarding information usually brings a potential threat to detection performance since the real relaying flows could be filtered out or adversaries could manipulate flow characteristics to get filtered out. Therefore, a more scalable solution for relay detection problem would be of potential value in many situations.
3.3 Detecting Relay Nodes in Linear Time

In this section we explain the proposed algorithm to detect relay nodes in liner time. We begin with giving few definitions in the next subsection.

3.3.1 Definitions

**Flow:** A flow is the collection of packets which share the common the five-tuple of source IP, source port, destination IP, destination port and layer 3 protocol type (UDP or TCP).

**Incoming or Outgoing Flows:** For a particular node, if the destination IP of a flow is equal to the IP of that node, then that flow is considered as an incoming flow. Conversely, if the source IP of the flow is equal to the IP of that node, then the flow is regarded as an outgoing flow.

**Time Slot:** In this work, time axis is considered as a sequence of equal length time intervals which are called time slots.

**Active Flow:** If a flow has at least one packet which are transmitted within a given time slot, then that flow is regarded as being active within that particular time slot.

3.3.2 Basic Idea

The basic idea of the proposed technique relies on the fact that the incoming and outgoing components of relay flow have to transmit at least some of the packets (i.e. non-chaff packets) at similar times (i.e. within the maximum tolerable delay duration). That is to say, if the time axis is considered as a sequence of time slots, the corresponding incoming and outgoing flows of the relay would be simultaneously active within some of the time slots. This network pattern is illustrated in Figure 3.1, where the incoming flow $R$ and outgoing flow $F$ are acting as a relaying flow pair. It is observed that, in order to forward the received information through flow $R$, flow $F$ is also active in the same time slots as flow $R$.

In order to capture this pattern exhibited by relaying flows, we assign to each flow a random number drawn from a zero-mean known distribution. Then, for each time slot, the random numbers assigned to active incoming flows are summed and multiplied to the sum of random numbers assigned to the active outgoing flows. Finally the results of each time slot are added.
together and an overall sum value is obtained. Calculation of this overall sum \( S \) is summarized below for the flows shown in Figure 3.1. Note that here the letters \( A, B, C \) etc. represent the assigned random numbers to the corresponding flows shown Figure 3.1.

\[
S = (R + A)(F + B) + C(D + B) + (A + R)F + AD + RF \\
= RF + RB + AF + AB + CD + CB + AF + RF + AD + RF \\
= 3 \times RF + (RB + AF + AB + CD + CB + AF + AD) \\
= 3 \times RF + \text{sum(Random Numbers)}
\]

It is observed above that, thanks to distributive property of multiplication over addition, this calculation is effectively equivalent to assigning a random number to each active flow pair and then summing the numbers up over all active flows. Therefore, the random number assigned to the correlated flow pair appears multiple times in the summation and somewhat can be considered as a constant term in the summation. In other words, if there were no correlated relaying flows, “\( S \)” would be the sum of random numbers coming from a zero-mean known distribution. Consequently “\( S \)” itself would be a random number coming from another zero-mean known distribution. However, the presence of a correlated flow pair acts as a constant term in the summation and therefore changes the governing distribution of “\( S \)”. Hence, based on a simple statistical test applied on calculated “\( S \)” values, one can classify a node as being a relay or not. The mathematical details of this scheme are presented in Section 3.3.3.

At this point, an acute reader might have observed that request-response based protocols (i.e. TCP) pose a problem against the proposed technique. This is because flows which carry requests and corresponding flows which carry responses are often active within the same time slots. Consequently, network nodes which use such protocols would be automatically declared as relay nodes by the scheme we have described above. But there is a simple fix to this problem. Essentially, only one flow of such flow pairs should contribute to the summation process we have outlined above and the other flow should simply be ignored. This solution can easily be implemented by checking if a random number has already been accumulated for a flow which has the exact reverse direction (i.e source and destination IP addresses are swapped, source and destination ports are swapped and layer 3 protocol is the same) of a given flow. If the answer is yes, that flow is simply ignored since its counterpart flow has already been taken into account.
3.3.3 Problem Formulation and Solution

Our goal is to differentiate between network traffic data which contains relay activity and that which does not. The basic difference between these two situations is the number of time slots in which the same flow pair is simultaneously active. That is because, although uncorrelated flows may share few time slots but it is very unlikely that the number of such time slots is high. Otherwise these flows shouldn’t be considered as uncorrelated because in that case whenever one flow was active the other flow would have been active as well. On the other hand, for a flow pair relaying information with a maximum tolerable delay constraint, if one flow is active for a given time slot, with high probability the other flow is active in order to relay information before the time constraint elapses.

We denote the number of time slots shared by the same flow pair as $\beta$. Clearly, high values of $\beta$ indicate a correlation between these two flows and hence is indicative of relay activity. The rest of this section explains the details of differentiating these relay and non-relay situations based on $\beta$.

For a given node in the network, let $I_i$ denote the incoming flows and $O_j$ denote the outgoing flows, where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. For each incoming flow and each outgoing flow, a random number is generated and assigned to the corresponding flow. Let $RI_i$ and $RO_j$ denote the random numbers assigned to the incoming flow $I_i$ and the outgoing flow $O_j$ respectively. Here $RI_i$ and $RO_j$ are assigned such that they are independently drawn from the probability mass function $P(n)$, which is:
The reason \( P(n) \) is chosen as a bipolar symmetric PMF is that our detection algorithm requires the distribution of \(RI_i \times RO_j\) to be bipolar symmetric. This distribution turns out to be bipolar symmetric as long as \( P(n) \) itself is chosen as bipolar symmetric.

Meanwhile, let \( \hat{i}_t \) be the indices of the \( m \) active incoming flows within a given time slot \( t \) where \( \hat{i}_t \in \{1, 2, 3, ..., m\} \). Similarly let \( \hat{j}_t \) be the indices of \( n \) active outgoing flows in time slot \( t \) where \( \hat{j}_t \in \{1, 2, 3...n\}\). Then for each time slot \( t \), the corresponding random numbers for active incoming flows within \( t \) are summed and multiplied to the sum of the random numbers assigned to active outgoing flows. This step is repeated for each time slot and result of each step is accumulated. More formally, the following summation \( S \) is calculated for time slots \( t = 1, 2, 3, ..., T \):

\[
S = \sum_{t=1}^{T} \left[ \sum_{i \in \hat{i}_t} RI_i \times \sum_{j \in \hat{j}_t} RO_j \right] \tag{3.2}
\]

This \( S \) value serves as the feature which is used to classify whether the network traffic of a node contains relay activity or not. The reason for this becomes more clear if we rewrite Equation (3.2) as:

\[
S = \sum_{t=1}^{T} \left[ \sum_{i \in \hat{i}_t} \sum_{j \in \hat{j}_t} RI_i \times RO_j \right] \tag{3.3}
\]

Notice that \( S \) is effectively the summation of a new set of random numbers which are assigned to each active incoming-outgoing flow pair for each time slot. Therefore, Equation (3.3) can be written as:

\[
S = \sum_{a=1}^{A} M_a \tag{3.4}
\]

where \( A \) is the number of terms such that “\( A = \sum_{t=1}^{T} \sum_{i \in \hat{i}_t} \sum_{j \in \hat{j}_t} 1 \)” and “\( M_a = (RI_i \times RO_j) \)”,
which are random numbers drawn from probability mass function $\hat{P}(n)$:

$$\hat{P}(n) = \begin{cases} 
\frac{1}{2}, & \text{if } n = +\gamma^2 \\
\frac{1}{2}, & \text{if } n = -\gamma^2 \\
0, & \text{elsewhere}
\end{cases}, \quad \text{where } \gamma \in \mathbb{R} \quad (3.5)$$

Rigorously speaking, the $M_a$ values may not be independent since a single $RI_i$ or a single $RO_j$ may, and probably will, contribute to multiple $M_a$ values. However, for practical purposes $M_a$ values are considered as i.i.d. random variables with probability mass function $\hat{P}(n)$. As a matter of fact, this is not a very crude assumption$^1$.

Assuming independent $M_a$ values, if there is no correlation between any pair of incoming and outgoing flows, in other words if there is no relaying flow pairs, then Equation (3.4) holds and $S$ will simply be interpreted as the sum of independent random variables. On the other hand if $y^{th}$ outgoing flow ($RO_y$) relays the packets from $x^{th}$ incoming flow ($RI_x$), both $RI_x$ and $RO_y$ will be active in a number of time slots ($\beta$), then the sum in Equation (3.4) can be written as:

$$S = \beta (RI_x \times RO_y) + \sum_{a=1}^{A-\beta} M_a, \quad \text{where } \beta \geq 0 \quad (3.6)$$

More generally if there are $F$ such incoming-outgoing flow pairs, namely $I_{f_i}$ and $O_{f_o}$, where $f_i$ and $f_o$ indicate incoming and outgoing flow indices of $f^{th}$ pair respectively ($f = 1, 2, ..F$), and each of these pairs are simultaneously active within $\beta_f$ time slots, then the summation $S$ can be written as:

$$S = \sum_{f=1}^{F} \beta_f M_f + \sum_{a=1}^{A-\beta} M_a \quad (3.7)$$

$^1$Suppose $M_1 = RI_i \times RO_j$ and $M_2 = RI_i \times RO_k$. Since $P(n)$ is bipolar symmetric, knowing $RI_i$ doesn’t give any information about $RI_i \times RO_j$. In other words $P(M_1 \mid M_2) = P(M_1)$ and therefore $M_1$ and $M_2$ are independent. However, suppose the following values were also given or known: $M_3 = RI_m \times RO_j$ and $M_4 = RI_m \times RO_k$. Then, by using $M_3$ and $M_4$ one could determine the polarity relation of $RO_j$ and $RO_k$ such that, if $M_3 = M_4$ then $RO_j = RO_k$ as well and if $M_3 \neq M_4$ then $RO_j \neq RO_k$. This information could later be used to determine $M_1$ such that if $RO_j = RO_k$ then $M_1 = M_2$ and if $RO_j \neq RO_k$ then $M_1 = -M_2$. So, $M_1$ is not independent from the collection ($M_2, M_3, M_4$). However, in order this to happen, flow $I_i$ has to be active in the same time slot with $O_j$ and in another or the same time slot with $O_k$. Also $I_m$ should be active in the same time slot with $O_j$ and in another or the same time slot with $O_k$. And all this has to happen in a short period of time before “$A$” terms are collected. The probability of such event is not very high and therefore independent $M_a$ assumption is quite realistic.
where \( M_f = (RI_f \times RO_f) \), \( \beta = \sum_{f=1}^{F} \beta_f \) and \( \beta_f \geq 0 \) for all \( f \). Notice that Equation (3.7) reduces to Equation (3.6) when \( F = 1 \) indicating a single relaying input-output flow pair and further reduces to Equation (3.4) when \( F = 0 \) indicating no relay.

The value \( S \) is a random variable in both Equations (3.4) and (3.7) and variance of \( S \) \( (\sigma_S^2) \) can be used to identify relay activity. In order to show this, let \( \sigma_{S,\text{noRelay}}^2 \) represents the variance of \( S \) where there’s no relay activity. We can write \( \sigma_{S,\text{noRelay}}^2 = \gamma^4 A \) from Equation (3.4), since \( M_\alpha \) values are i.i.d random variables drawn from probability distribution given in Equation (3.5) whose variance is \( \gamma^4 \). On the other hand, \( \sigma_{S,\text{withRelay}}^2 \), which represents the variance of \( S \) under relay traffic, can be similarly calculated from Equation (3.7) as:

\[
\sigma_{S,\text{withRelay}}^2 = \sum_{f=1}^{F} (\beta_f)^2 \gamma^4 + (A - \beta) \gamma^4 \\
= \gamma^4 \left( \sum_{f=1}^{F} (\beta_f)^2 - \beta \right) + \gamma^4 A \\
= \gamma^4 \left( \sum_{f=1}^{F} (\beta_f)^2 - \beta \right) + \sigma_{S,\text{noRelay}}^2 \tag{3.8}
\]

Since \( \beta = \sum_{f=1}^{F} \beta_f \), we can write

\[
\sigma_{S,\text{withRelay}}^2 = \gamma^4 \sum_{f=1}^{F} ((\beta_f)^2 - \beta_f) + \sigma_{S,\text{noRelay}}^2 \\
or \\
= \gamma^4 \sum_{f=1}^{F} (\beta_f(\beta_f - 1)) + \sigma_{S,\text{noRelay}}^2 \tag{3.9}
\]

Here we observe \( \sigma_{S,\text{withRelay}}^2 > \sigma_{S,\text{noRelay}}^2 \) so long as \( \sum_{f=1}^{F} (\beta_f)^2 > \sum_{f=1}^{F} \beta_f \). This constraint is satisfied as long as \( \beta_f \) values are greater than one. Therefore one can identify relay nodes by looking at \( \sigma_S^2 \) values as long as the relay flows are simultaneously active for more than one time slot. This is a purely theoretic constraint and in practice a relay has to be active for sufficiently enough number of time slots in order to be detected. Fortunately, most of the relay scenarios have to satisfy this constraint in order to serve their purpose. Another interesting observation is that the above constraint is independent of \( \gamma \). Therefore, the system performance doesn’t change for different values of \( \gamma \). Therefore, in the experiments this value is simply set to \( \gamma = 1 \).

In practice, estimating \( \sigma_S^2 \) is the first task to be performed. Then the system can declare a node as being a relay if the estimated \( \sigma_S^2 \) is sufficiently larger than the anticipated \( \sigma_S^2 \) if the node is not relaying at all. For this purpose, the system computes a number of \( S \) values
simultaneously but independently in order to estimate the $\sigma^2_S$ value. Details of the algorithm and its analysis are explained in the following sub-section.

### 3.3.4 Detection Algorithm and the Analysis

In this section we first discuss the proposed algorithms which compute $S$ values in order to estimate $\sigma^2_S$ and consequently performs relay detection. Following that, space and time requirements of these algorithms are discussed.

The $S$ values are calculated by the algorithm, $Calculate\_S$ listed below, which basically performs the operation defined in Equation (3.2). The algorithm takes the parameter $A$, which is introduced in Equation (3.4) and indicates the number of terms added together. Here $I$ and $O$ indicate the list of incoming and outgoing flows respectively. Also the function “Reverse()” returns the flow identification which has the complete reverse direction of a given flow. The algorithm then checks if this reverse flow has been assigned a random number previously in order to deal with the problem of request-response based protocols mentioned earlier in Section 3.3.2.

Notice that $Calculate\_S$ computes a single $S$ value. In order to accurately estimate $\sigma^2_S$, a number of $S$ values have to be collected. This can be done by executing multiple $Calculate\_S$ instances simultaneously. The algorithm, $Estimate\_\sigma^2_S$ listed below, implements the estimation procedure by employing the algorithm $Calculate\_S$. The input parameter $T$ indicates the number of simultaneous $Calculate\_S$ executions.

It should be noted that $Estimate\_\sigma^2_S$ outputs a single $\sigma^2_S$ value whenever $A$ terms are collected. The time elapsed until $A$ terms are collected totally depends on the input parameters and the characteristics of the underlying network traffic. In a typical scenario the parameters can be chosen so that the elapsed time to collect $A$ terms is around few seconds. Therefore, each estimated $\sigma^2_S$ value would correspond to a few seconds of network traffic. For continuous operation, $Estimate\_\sigma^2_S$ should be executed repeatedly.

The final decision is given based on the difference between estimated $\sigma^2_S$ values and the anticipated $\sigma^2_S$ value when the node is not relaying at all. This value can be written more formally as:
Algorithm 3.1 Calculate $S(A, I = \{I_1, \ldots, I_m\}, O = \{O_1, O_n\})$

$S \leftarrow 0$

$\text{noOfTerms} \leftarrow 0$

$timeSlot \leftarrow \text{currentTimeSlot}$

\textbf{while} $\text{noOfTerms} \leq A$ \textbf{do}

\hspace{1em} $\text{incomingSum} \leftarrow 0$

\hspace{1em} $\text{actInFl} \leftarrow 0$

\hspace{2em} \textbf{for all} active incoming flow $I_i$ within $\text{timeSlot}$ \textbf{do}

\hspace{3em} if $RO_i$ is assigned to $\text{Reverse}(I_i)$ \textbf{then}

\hspace{4em} continue; // In order to avoid request-response problem ignore this flow

\hspace{3em} end if

\hspace{3em} if $RI_i$ is not assigned to $I_i$ \textbf{then}

\hspace{4em} assign $RI_i$ randomly \hspace{1em} [as in Eq(3.1)]

\hspace{3em} end if

\hspace{3em} $\text{incomingSum} \leftarrow \text{incomingSum} + RI_i$

\hspace{3em} actInFl $\leftarrow$ actInFl $+$ 1

\hspace{2em} \textbf{end for}

\hspace{1em} $\text{outgoingSum} \leftarrow 0$

\hspace{1em} $\text{actOutFl} \leftarrow 0$

\hspace{2em} \textbf{for all} active incoming flow $O_j$ within $\text{timeSlot}$ \textbf{do}

\hspace{3em} if $RI_j$ is assigned to $\text{Reverse}(O_j)$ \textbf{then}

\hspace{4em} continue; // In order to avoid request-response problem ignore this flow

\hspace{3em} end if

\hspace{3em} if $RO_j$ is not assigned to $O_j$ \textbf{then}

\hspace{4em} assign $RO_j$ randomly \hspace{1em} [as in Eq(3.1)]

\hspace{3em} end if

\hspace{3em} $\text{outgoingSum} \leftarrow \text{outgoingSum} + RO_i$

\hspace{3em} actOutFl $\leftarrow$ actOutFl $+$ 1

\hspace{2em} \textbf{end for}

\hspace{1em} $\text{noOfTerms} \leftarrow \text{noOfTerms} + \text{actInFl} \times \text{actOutFl}$

\hspace{1em} $\text{timeSlot} \leftarrow \text{nextTimeSlot}$

\hspace{1em} $S \leftarrow S + \text{incomingSum} \times \text{outgoingSum}$

\textbf{end while}

\hspace{1em} output $S$
\[ \Delta = \sigma_S^2 - \gamma^4 A \quad (3.10) \]

In the experiments, a recent few \( \Delta \) values are incorporated in the decision process such that if the sum of \( \text{“d”} \) most recently calculated \( \Delta \) values exceeds a threshold \( \text{“}th\text{”} \) then the corresponding host is declared as performing relay activity within the corresponding time slice.

Before we begin to analyze the algorithm, it should be emphasized that the parameters \( A \) and \( T \) are constant values which are in the order of few hundreds and they do not vary with the input size. As for the investigation of time requirements, the algorithm \( \text{Calculate}_S \) loops over the active incoming flows and active outgoing flows separately. It repeats these loops a constant number of times until \( A \) terms are collected for summation. Therefore \( \text{Calculate}_S \) is \( O(m + n) \) time algorithm where \( m \) and \( n \) are the number of incoming and outgoing flows respectively of the node being analyzed. Practically speaking, the algorithm may actually loop fewer times than \( m + n \) since only a fraction of incoming and outgoing flows are active for a given time slot.

For the second algorithm, \( \text{Estimate}_\sigma^2 \) calls \( \text{Calculate}_S \) exactly \( T \) times which is indeed a constant parameter. Therefore, \( \text{Estimate}_\sigma^2 \) too runs in \( O(m + n) \) time. However, the time requirements of the whole system actually depends on how many times the algorithm \( \text{Estimate}_\sigma^2 \) is executed. But again for a given fixed time interval, \( \text{Estimate}_\sigma^2 \) is called repeatedly for a constant number of times. Therefore for a given fixed time interval, which is typically the duration of a typical relay activity, the decision is given in linear time.

The space required by the algorithm, on the other hand, is mainly the table which keeps assigned random values of the incoming and outgoing flows. Hence, it can be concluded that the algorithm requires linear space as well.

**Algorithm 3.2** \( \text{Estimate}_\sigma^2(A, T, I, O) \)

\[
\begin{align*}
&\sigma_S^2 \leftarrow 0 \\
&\text{for } i = 1 \text{ to } T \text{ do} \\
&\quad S_i \leftarrow \text{Calculate}_S(A, I, O) \\
&\quad \sigma_S^2 \leftarrow \sigma_S^2 + \frac{(S_i)^2}{T} \\
&\text{end for} \\
&\text{output } \sigma_S^2
\end{align*}
\]
3.4 Experiments and Results

In order to verify the practical efficacy of the proposed scheme, the algorithms discussed in Section 3.3.4 were implemented and executed for real network traffic. This section presents the experimental setup and their results in order to demonstrate the performance of the proposed scheme.

3.4.1 Experimental Setup

The ultimate goal of the proposed scheme is to identify network nodes that perform relay activity, or in other words, “relay nodes”. Aside from relay traffic, in almost every case, relay nodes also receive and transmit legitimate non-relay traffic. Therefore, in our experiments, traffic for relay nodes was constructed such that pure relay traffic is blended into non-relay host traffic. For the non-relay traffic, two different types of traffic data were captured from real network. The first type was the traffic of our university’s web server which basically consists of http flows. The second type was captured from a mail server, which provides mail client connections and ssh/telnet interactive sessions. Both traffic data were captured on a typical day for a few hours. The Web server’s traffic had average packet rate of 70 packets/second and average bit rate of 416 kbps whereas mail server’s traffic had higher average packet rate of 76 packets/second but lower average bit rate of 257 kbps.

The relay traffic on the other hand was artificially generated where the packet inter-arrival times were determined by the following Gaussian mixture model:
\[
P(i) = p \mathcal{N}(\mu_{\text{short}}, \sigma_{\text{short}}^2) + (1 - p) \mathcal{N}(\mu_{\text{long}}, \sigma_{\text{long}}^2)
\]  
(3.11)

where, \( \mathcal{N}(\mu, \sigma^2) \) indicates normal distribution with mean \( \mu \) and variance \( \sigma^2 \).

This model generates bursty traffic such that the packets are mostly sent back to back without waiting too much (with probability \( p \)) and a pause period occurs once in a while (with probability \( 1 - p \)). The reason this model was used is that bursty traffic captures the behavior of most applications more accurately. Notice that this model generates only the incoming portion of the relay traffic. Each received packet has to be forwarded in order to obtain a complete relay activity. Rather than forwarding packets immediately, a certain amount of delay was introduced for each packet in order to represent the packet processing time and/or intentional adversarial delays. For each packet, delay values were chosen randomly from a normal distribution with \( \mu_{\text{delay}} \) and \( \sigma_{\text{delay}}^2 \). About 30 second portion of an example relay traffic generated by this model is given in Figure 3.2, which shows the incoming packets, inter-arrival times, and introduced delay between incoming and outgoing packets. The parameters used for the relay traffic in this figure are \( p = 0.8, \mu_{\text{short}} = 100 \) milliseconds, \( \sigma_{\text{short}} = 10 \), \( \mu_{\text{long}} = 3000 \) milliseconds and \( \sigma_{\text{long}} = 500 \). The introduced delay parameters are \( \mu_{\text{delay}} = 400 \) milliseconds and \( \sigma_{\text{delay}} = 50 \). Also in this figure, positive bars indicate incoming packets where negative ones indicate corresponding forwarded (relayed) packets.

Given network traffic data, the system was required to decide if there is relay activity or not. The decision is given based on the \( \Delta \) value described in Section 3.3.4. As discussed in that section, the system calculates a \( \Delta \) value each time \( A \) terms are collected. In our experiments, the decision in favor of the presence of a relay activity was made if the sum of the most recent two \( \Delta \) values exceeded the threshold value \( th = 1000 \). The numbers 2 and 1000 are selected experimentally and they are tuned to detect shorter relay activities which last only for a few hundred packets. However, it should be noted that further reducing \( th \) value would enable detecting even shorter relay activity but would incur the cost of increased false positive rates. On the other hand, large \( th \) values could reduce false positive rates to arbitrarily small values but the system can detect only long enough relay activity.

In order to measure detection performance, the generated relay traffic was blended into real
network traffic and the overall traffic was fed to the system for analysis. If the system was able to detect the relay activity by the time all relay packets are forwarded, then the number of true positives was incremented. This process was repeated several times and at the end, the ratio of the true positives to the number of all generated relay traffic was declared as the true positive rate. The same experiment was then repeated once more but this time without blending the relay traffic into the real traffic. Hence, if the system declares a relay activity then it means a false positive and the false positive counter was incremented. Finally the ratio of the number of false positives to the number of all generated but not blended relay traffic gives the false positive rate.
3.4.2 Results

In this section the results of the experiments described in the previous section are discussed. The experiments were conducted using various time slot lengths \((L)\), and various \((A)\) values which indicates the number of terms incorporated as discussed in Section 3.3.4 in Equation (3.4). The performance was also investigated for various average delay values that relay packets encounter. In the first set of experiments we used only one pair of relaying flows (i.e. \(F = 1\) in Equation (3.7)) and relay traffic consisted of 200 incoming packets and the corresponding 200 outgoing relayed packets. The inter-arrival times between incoming relay packets were generated according to the model in Equation (3.11) where \(p = 0.8\), \(\mu_{short} = 100\) milliseconds, \(\sigma_{short} = 10\), \(\mu_{long} = 3000\) milliseconds and \(\sigma_{long} = 500\). A delay value was drawn from \(N(\mu_{delay}, \sigma_{delay}^2)\) between each incoming packet and the corresponding forwarded packet. Here \(\mu_{delay}\) was changed throughout the experiments but \(\sigma_{delay} = 50\) was kept fixed. Finally the resulting relay traffic, which typically lasts for about 150-160 seconds, was blended into the captured real traffic and true positive rate was calculated.

Figure 3.3 and Figure 3.4 plot the true positive rates vs. average delay \((\mu_{delay})\) of the relay traffic where the relay traffic is blended into web server traffic and mail server traffic respectively. The experiment was repeated for parameter values \(A = 500, 1000, 1500, 2000\) and \(L = 100, 200, 400, 600\) milliseconds. Each of the four sub-figures corresponds to a different \(A\) value whereas each sub-figure contains four different curves each represents a different \(L\) value.

In all experiments, regardless of the \(A\) and \(L\) values, it is observed that the true positive rate decreases as the average delay of the relay is increased. This behavior is expected since increased delay reduces the probability of relaying flows being active simultaneously in the same time slot. In other words, as the delay increases, it is more likely that the time slot during which the incoming packet is received, elapses before the node forwards that packet. However, another interesting behavior was also observed in all experiments. Even though the average delay value was way greater than the length of the time slots \((L)\), some of the relay activities were still successfully detected. For instance when the time slot length \((L)\) was 100 milliseconds and the average delay was 600 milliseconds, there is no way that the incoming packets are relayed within the same time slot. Therefore, at first sight it can be said that the system cannot
detect these relays although it certainly can as shown in Figures 3.3 and 3.4. The reason for this behavior is that sometimes relay packets corresponding to previous incoming packets are transmitted within the same time slot that a new packet arrives. Therefore, although the same content isn’t being relayed at the same time slot, both incoming and outgoing relay flows still may simultaneously be active. This event contributes to the increase in calculated $\Delta$ value and enables the system to detect some of these relays.

It is also observed that in all experiments, detection performance decreases as the length of the time slots increases. This is little bit counterintuitive since one may expect to capture relay activity more accurately with longer time slots. However, in reality longer time slots means more flows (relay or non-relay) are active within the same time slot and therefore more random number terms added to the overall sum as in the algorithm $Calculate_S$. Therefore, the system
collects “A” terms more rapidly, and hence is forced to make a decision earlier. This sometimes prevents relay flows from injecting sufficient number of packets in order to be detected before the system makes decision. Consequently some of the relay activities are left undetected with larger $L$ values.

On the other hand, increasing the “A” value certainly delays the time that the system has to make a decision. Therefore, relay flows will have enough time to inject sufficient number of packets to be detected. This behavior can be observed as an increased detection performance as we go from Figure 3.8(a) to Figure 3.3(d) and from Figure 3.4(a) to Figure 3.4(d). However, this increase in the detection performance comes at the cost of increased false positive rates as can be seen in Table 3.1. This is primarily due to the fact that the number of time slots which any two arbitrary flows are simultaneously active within, increases as $A$ is increased. This is a simple fact from probability theory that if the size of the universal set increases, the number of event occurrences increases as well, as long as the frequency of that event remains constant.

When we look at the results presented in Figure 3.3, Figure 3.4, Table 3.1, we observe that some settings of the parameters lead to quite unsatisfactory results. For instance, in the case when $A = 500$ and $L = 600$, the true positive rate is too low. Therefore, the parameters $A$ and $L$ should not be set to these values although they are included in the figures to demonstrate the effect of changing parameters. Similarly when $A = 2000$, even though true positive rates are significantly higher, this setting shouldn’t be used due to high false positive rates. However, if the proposed algorithm were to be used as an initial step for relay flow detection as discussed earlier, setting $A = 2000$ may not be a bad idea. Because, in that case the nodes that are flagged by the proposed algorithm would be further analyzed by a relay flow detection algorithm in order to identify relay flows. High false positive rates do increase the number of flagged nodes but this computational inefficiency may still be better as compared to the case where the proposed scheme is not used at all as the initial step. In the rest of the experiments, we set our algorithm parameters as $A = 1000$ and $L = 100$ as they lead to reasonable true positive and false positive rates. However, for real deployments, some care will need to be taken before setting algorithm parameters. Finally, it was observed for all parameter values that the performance for the mail server traffic was slightly worse than the performance for web server traffic. The reason is that the mail server traffic had higher average packet rates than the web server traffic. Similar to
the previous observations, the more the packet rate, the traffic obtains the more active flows fall within a time slot and hence the earlier the “$A$” terms are collected. Therefore, the system has to make a decision earlier under heavier traffic and consequently it may miss some relay activities. In order to minimize this effect, the system parameters should be carefully chosen according to the expected traffic characteristics. In the experiments identical parameter settings are used for both web server and mail server traffic for comparison purposes.

<table>
<thead>
<tr>
<th>$L$</th>
<th>UNDER WEB SERVER TRAFFIC</th>
<th>UNDER MAIL SERVER TRAFFIC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A = 500$</td>
<td>$A = 500$</td>
</tr>
<tr>
<td>$L = 100$</td>
<td>0.0088</td>
<td>0.0352</td>
</tr>
<tr>
<td>$L = 200$</td>
<td>0.1257</td>
<td>0.1337</td>
</tr>
<tr>
<td>$L = 400$</td>
<td>0.1890</td>
<td>0.1886</td>
</tr>
<tr>
<td>$L = 600$</td>
<td>0.2842</td>
<td>0.2873</td>
</tr>
</tbody>
</table>

### Effect of Multiple Flow Pairs

In the experiments described above, a single pair of flows perform the relaying activity. However, it is possible that a relay node could host multiple relay activities simultaneously. The number of such relay activities is denoted by “$F$” in Section 3.3.3. In this section the
performance of the system was investigated for different values of $F$. In order to demonstrate the performance, for a specific set of parameters such as $A = 1000$ and $L = 100$, the system is required to detect the presence of a relay activity when there are multiple relaying flow pairs. As shown in Figure 3.5, the true positive rate increases drastically as the number of relaying flow pairs increases. In fact the relay activity is detected almost with 100% accuracy when there are more than four relaying flow pairs. This result is not surprising because higher values of $F$ also increases the variance ($\sigma^2_S$) as observed in Equation (3.8).

**Effect Relay Duration**

In section 3.3.3, it was mentioned that theoretically relay activity is detected if corresponding flows are simultaneously active within more than one time slot. It was also mentioned that this was a purely theoretical conclusion and in practice sufficiently many packets have to be relayed in order to be detected. This section investigates the detection performance for various relay durations and gives insight about the minimum detectable relay duration. Here the duration of a relay is measured as the number of relayed packets. In the experiment, where the parameters are chosen as $A = 1000$ and $L = 100$, relay packets were generated by the same model described in Section 3.4.1. The true positive rate vs. average delay curves for different number of relayed packets are presented in Figure 3.6. As expected, it is observed that the detection performance increases as the number of relayed packets increases. This is because, as the number of relayed packets increases the number of time slots, within which the corresponding flows are active, also increases. This exactly corresponds to the increase in $\beta_f$ values mentioned in Equation (3.7) and consequently increase in $\sigma^2_S$ and $\Delta$.

On the other hand, it is also observed in Figure 3.6 that the proposed scheme has low accuracy in detecting relays that last for less than about 50-60 packets for the given underlying traffic characteristics. Here it should be noted that the underlying web server traffic and mail server traffic used in the experiments have average packet rates of 70 packets/sec and 76 packets/sec respectively. Since the average packet rate of the mail server traffic is higher, when the curves for mail server and web server traffic are compared, it is expected that, the minimum detectable relay duration gets lower and lower as the non-relay traffic of the node gets lighter.
Effect of Chaff Packets

In order to disturb the correlation structure between relaying incoming and outgoing flows, adversaries often blend chaff packets into the relay stream. This enables an adversary to relay information without being detected by flow-correlation based relay detection algorithms. The chaff packets usually carry no useful information. They can be placed in the incoming flow and not relayed through the outgoing flow. They can also be generated by the relay node and placed in the outgoing flow. In both cases the purpose is to generate a packet in a flow which has no counterpart in the corresponding flow. However, regardless of the chaff packets, some of the packets (actual relay packets) still have to be relayed within a certain time period in order the relay node to serve its purpose. Those packets will still make the incoming and outgoing flows simultaneously active within a number of time slots. Hence, the proposed scheme will be able to detect the relay activity. Therefore it can be said that chaff packets have virtually no effect on the proposed method. This property is clearly observed in the experiments whose results are presented in Figure 3.7. In the experiments, the system parameters are set to $A = 1000$ and $L = 100$. After each relayed packet a chaff packet is generated with probability $P(\text{chaff}) = 0, 0.2, 0.4, 0.6,$ and $0.8$. These chaff packets take place only in incoming flow and are not relayed through the outgoing flow. The inter-arrival times for these chaff packets are determined by the same model which determines the inter-arrival times of regular incoming relay packets as described in Section 3.4.1. It is observed that none of these $P(\text{chaff})$ values...
have decreased the detection performance at all. On the contrary, as the number of chaff packets is increased, the detection performance is slightly improved especially for larger average delay values. This is because these extra chaff packets in the incoming flow sometimes coincides with other packets in the outgoing flow and therefore increase the number of time slots that both flows are simultaneously active.

3.5 Problems with High-Volume Traffic and Potential Solutions

According to Equation (3.9), if there’s an incoming-outgoing flow pair which are simultaneously active within substantial number of time slots (i.e. high $\beta_f$ value), then the variance of $S$ will be significantly higher than the variance when there’s no relay activity. Hence one can identify relay nodes as the nodes which produce large $\sigma^2_S$ values. However, $\sigma^2_S$ can also be large if there are many flow pairs which are simultaneously active within just a few time slots (i.e. large $F$ and small $\beta_f$). This situation is in fact observed when the node has significantly high network traffic even if there’s no relaying flow pair. In such a case, there will be a high probability of finding an outgoing flow sharing more than one common time slots with any given incoming flow even though they are totally uncorrelated. Consequently, a detection scheme based on $\sigma^2_S$ would mistakenly consider such nodes as relay nodes. In order to differentiate such case from genuine relay activities we can split the genuine and accidental cases in Equation (3.9) as:

Figure 3.7: True positive rate vs. average delay curves for different chaff packet probabilities.
\[ \sigma^2_{S, \text{with Relay}} = \gamma^4 \sum_{f=1}^{F-F_g} \left( \dot{\beta}_f(\ddot{\beta}_f - 1) \right) + \gamma^4 \sum_{f=1}^{F_g} \left( \ddot{\beta}_f(\dddot{\beta}_f - 1) \right) + \sigma^2_{S, \text{no Relay}} \] (3.12)

where \( \beta_f \) values are sorted in ascending order without loss of generality so that, \( \dot{\beta}_f \) represents smaller \( \beta_f \) values accounting for accidental cases and \( \ddot{\beta}_f \) represents larger \( \beta_f \) values accounting for genuine relay flow pairs. Also in above equation, \( F_g \) indicates the number of genuine relaying flow pairs. Consecutively, we can consider \( \sigma^2_{S, \text{no Genuine Relay}} \) as the variance value when there’s no genuine relay activity but some flow pairs accidentally happen to be simultaneously active for few time slots, such that:

\[ \sigma^2_{S, \text{no Genuine Relay}} = \gamma^4 \sum_{f=1}^{F-F_g} \left( \dot{\beta}_f(\ddot{\beta}_f - 1) \right) + \sigma^2_{S, \text{no Relay}} \] (3.13)

Now suppose we start over calculations all the way from Equation (3.2) but with introducing a simple perturbation factor such that instead of doing the multiplication in the same time slot, the sum of the random numbers assigned to incoming flows of each time slot are now multiplied by the sum of random numbers assigned to outgoing flows of a randomly selected time slot. This perturbation can be expressed by plugging the random mapping function “\( RM() \)” into Equation (3.2) as:

\[ S_{\text{scrambled}} = \sum_{t=1}^{T} \left[ \sum_{i \in i_t} RI_i \times \sum_{j \in j_{RM(t)}} RO_j \right] \] (3.14)

where \( RM(t) \) maps the \( t^{th} \) time slot to a randomly selected time slot such that \( RM(t) \in \{1, 2, ..., T\} \). Since no extra flows or packets are introduced to the summation, the probability of finding another flow which accidentally shares few time slots with any given flow is expected to remain the same. However, disturbing the alignment of input and output time slots destroys any correlation between genuine incoming-outgoing relay flow pairs. Consequently, after scrambling the order of incoming and outgoing time slots, \( \dot{\beta}_f \) values are expected to remain similar whereas \( \ddot{\beta}_f \) values get very small or even become zero. Therefore, when the variance of \( S_{\text{scrambled}} \) is calculated, the second term in Equation (3.9) approaches to zero and \( \sigma^2_{S, \text{scrambled}} \) becomes very close to \( \sigma^2_{S, \text{no Genuine Relay}} \) such that:
\[ \sigma_{S_{scrambled}}^2 \approx \gamma^4 \sum_{f=1}^{F-F_g} \left( \beta_f (\hat{\beta}_f - 1) \right) + \sigma_{S_{noRelay}}^2 \] (3.15)

Consequently, we can identify the genuine relay activity by calculating the difference between \( \sigma_S^2 \) and \( \sigma_{S_{scrambled}}^2 \) such that:

\[ \Psi = \sigma_S^2 - \sigma_{S_{scrambled}}^2 \] (3.16)

Here, if there were no genuine relay activity \( \sigma_S^2 \) would converge to \( \sigma_{S_{noGenuineRelay}}^2 \) and therefore \( \Psi \) would be close zero since \( \sigma_{S_{noGenuineRelay}}^2 \approx \sigma_{S_{scrambled}}^2 \). On the other hand, if there were a genuine relay activity \( \sigma_{S_{withRelay}}^2 \) would converge to \( \sigma_{S_{scrambled}}^2 \) and \( \Psi \) would be much higher as indicated by Equations (3.12) and (3.15) such that:

\[ \Psi = \sigma_{S_{withRelay}}^2 - \sigma_{S_{scrambled}}^2 = \gamma^4 \sum_{f=1}^{F_g} \left( \beta_f (\hat{\beta}_f - 1) \right) \] (3.17)

As a result, the relay nodes can be identified by a simple comparison of \( \Psi \) values with a threshold. Therefore, an efficient calculation of \( \Psi \) is crucial for the system. A linear time algorithm, called Calculate_\( S \), which is used in estimating \( \sigma_S^2 \) was described in the previous section. A similar algorithm, which additionally incorporates the scrambling process, can be used to estimate \( \sigma_{S_{scrambled}}^2 \). We call this algorithm Calculate_\( S_{scrambled} \) and we present its details in the next page.

### 3.6 Experiments With A Real Relay Traffic

To evaluate the proposed scheme under a real world scenario, we utilized a real relay traffic, which was previously captured during an actual attack where one of the hosts in our network was compromised and used as a relay node for an entire day. In the attack, a chat session between two individuals were relayed through a compromised host in our network. Therefore, the captured relay traffic consists of two different hosts sending IRC packets back and forth using the host in our network as a relay.
Algorithm 3.3 Calculate $S_{scrambled}(A, I = \{I_1, .., I_m\}, O = \{O_1, .., O_n\})$

1: $S \leftarrow 0$
2: $\text{noOfTerms} \leftarrow 0$
3: $\text{timeSlot} \leftarrow \text{currentTimeSlot}$
4: while $\text{noOfTerms} \leq A$ do
5:   $\text{incomingSum} \leftarrow 0$
6:   $\text{actInFl} \leftarrow 0$
7:   for all active incoming flow $I_i$ within $\text{timeSlot}$ do
8:       if $\text{RO}_i$ for $\text{Reverse}(I_i)$ is already used in $\text{timeSlot}$ then
9:           continue;  //In order to avoid request-response problem ignore this flow
10:      end if
11:      if $\text{RI}_i$ is not assigned to $I_i$ then
12:         assign $\text{RI}_i$ randomly [as in Eq(3.1)]
13:      end if
14:      $\text{incomingSum} \leftarrow \text{incomingSum} + \text{RI}_i$
15:      $\text{actInFl} \leftarrow \text{actInFl} + 1$
16:   end for
17:   $\text{outgoingSum} \leftarrow 0$
18:   $\text{actOutFl} \leftarrow 0$
19:   $\text{scrambledTimeSlot} \leftarrow \text{RM}(\text{timeSlot})$ //Selects a random time slot
20:   for all active incoming flow $O_j$ within $\text{scrambledTimeSlot}$ do
21:       if $\text{RI}_j$ for $\text{Reverse}(O_j)$ is already used in $\text{scrambledTimeSlot}$ then
22:           continue;  //In order to avoid request-response problem ignore this flow
23:       end if
24:       if $\text{RO}_j$ is not assigned to $O_j$ then
25:          assign $\text{RO}_j$ randomly [as in Eq(3.1)]
26:       end if
27:       $\text{outgoingSum} \leftarrow \text{outgoingSum} + \text{RO}_i$
28:       $\text{actOutFl} \leftarrow \text{actOutFl} + 1$
29:   end for
30:   $\text{noOfTerms} \leftarrow \text{noOfTerms} + \text{actInFl} \times \text{actOutFl}$
31:   $\text{timeSlot} \leftarrow \text{nextTimeSlot}$
32:   $S \leftarrow S + \text{incomingSum} \times \text{outgoingSum}$
33: end while
34: output $S$
3.6.1 Accuracy of Estimated $\Psi$

The real relay traffic consists of two different hosts (host A and host B) sending packets back and forth to each other through the compromised node. Therefore, from our algorithm’s perspective, there were two relaying flow pairs, one from host A to B and one from host B to A. However, it should be noted that, since these two flows have exact reverse directions with respect to each other, at any given time slot only one pair of relaying flows could be active as the other pair would have been discarded by the algorithm due to eliminate request-response drawback as explained in Section 3.3.2.

In order to analyze the algorithm behavior, we blended this relay traffic into normal traffic of some selected nodes in our network. Then we estimated $\Psi$ values with the proposed scheme. The algorithm parameters during the experiments were set such that, the length of each time slot was 100 milliseconds, the number of collected terms ($A$ in Equation (3.4)) was 500, and finally the number of successive $S$ and $S_{scrambled}$ calculations ($T$) was 100. Again we set $\gamma = 1$ as different $\gamma$ values don’t affect the performance of the scheme in the experiments.

In this section, we present $\Psi$ values calculated for two specific type of nodes, which represent high traffic and low traffic cases. The first node is our university’s web server which contains fairly high amount of traffic. For the low traffic case, we select a group of computers in a computer lab, where students usually surf the Internet and check their mails. In Figures 3.8(a) and 3.8(b), calculated $\Psi$ values along with average observed $\beta_f$ values are plotted for both web server and computer lab traffic. Here recall that $\beta_f$ indicates the number of time slots that the relaying flow pairs are simultaneously active. One can also consider $\beta_f$ as the number of terms representing the corresponding relay activity when calculating $S$. In both Figures 3.8(a) and 3.8(b), it was observed that $\Psi$ values gets higher as $\beta_f$ gets higher. This eventually allows us to differentiate between relay and non-relay cases.

To show the accuracy of computed $\Psi$ values, in Figures 3.8(a) and 3.8(b), we also plotted the theoretical values of $\Psi$ calculated from Equation (3.17) for various $\beta_f$. We set $F_g = 2S$ in Equation (3.17) since there were two relaying flows in the blended traffic. It was observed that, the experimental values follows the theoretical curve especially for smaller values of $\beta_f$. In order to further increase the accuracy, we increased the number of successive $S$ and $S_{scrambled}$
calculations such that $T = 1000$. Although we observed a slight improvement as plotted in Figures 3.9(a) and 3.9(b), we continued to use $T = 100$ in the rest of our experiments as it is computationally less costly.

### 3.6.2 Detection Performance

As suggested by Equation (3.17) and also confirmed by the experiments, $\Psi$ values are higher when there’s a relay activity. When there’s no relay, the $\Psi$ values are concentrated around zero as presented in Figure 3.10. Therefore, one can compare the calculated $\Psi$ values with a specified threshold to identify relay nodes.

In order to specify a such threshold ($\Psi_{TH}$), we first have to specify $\beta_{tolerance}$, which indicates the minimum number of time slot that a relaying flow pair has to be active in order to be considered as a genuine relay. Once the $\beta_{tolerance}$ is specified, we can calculate the threshold from Equation (3.17) such that:

$$\Psi_{TH} = \gamma^4 \sum_{f=1}^{F_g} (\beta_{tolerance} (\beta_{tolerance} - 1))$$  \hspace{1cm} (3.18)

where $\gamma = 1$ as explained earlier and $F_g = 1$ since we want to detect even a single relaying flow pair. Having specified $\beta_{tolerance}$ and $\Psi_{TH}$ in order to measure the performance we introduce the following definitions:
Figure 3.9: Theoretical and Experimental Values of $\Psi$ for various $\beta_f$ and for $T = 1000$.

- **Non-Relaying Time Window**: is the time window corresponding to a single $\Psi$ calculation which contains less than $\beta_{tolerance}$ time slots which contain relay activity.

- **Relaying Time Windows**: is the time window corresponding to a single $\Psi$ calculation which contains more than $\beta_{tolerance}$ time slots which contain relay activity.

The detection performance can be measured by true positive rate and false positive rate which are defined as follows:

- **True Positive Rate**: is the the ratio of the number of relaying time windows having $\Psi \geq \Psi_{TH}$ to the number of all relaying time windows during the experiment.

- **False Positive Rate**: is the the ratio of the number of non-relaying time windows having $\Psi \geq \Psi_{TH}$ to the number of all non-relaying time windows during the experiment.

$\Psi_{TH}$, true positive rate and false positive rate for different $\beta_{tolerance}$ are tabulated in Table 3.2 and Table 3.3 for web server traffic and computer lab traffic respectively. In these tables, it is observed that the detection performance gets better for high $\beta_{tolerance}$ values as true positive rates increase and false positive rates decrease. However, using high $\beta_{tolerance}$ values usually is not desired since many genuine relays may be considered as non-relays by definition. This is because, by definition, the algorithm considers flow pairs which are active for less than $\beta_{tolerance}$ time slots as non-relays even though they are genuine relays. Therefore, keeping
$\beta_{\text{tolerance}}$ around 30 seems to be a reasonable choice as long as Table 3.2 and Table 3.3 are concerned.

Table 3.2: Detection performance for web server traffic

<table>
<thead>
<tr>
<th>$\beta_{\text{tolerance}}$</th>
<th>$\Psi_{TH}$</th>
<th>True Pos. Rate</th>
<th>False Pos. Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>0.94783</td>
<td>0.68548</td>
</tr>
<tr>
<td>15</td>
<td>210</td>
<td>0.9898</td>
<td>0.37589</td>
</tr>
<tr>
<td>25</td>
<td>600</td>
<td>1</td>
<td>0.18788</td>
</tr>
<tr>
<td>35</td>
<td>1190</td>
<td>0.98113</td>
<td>0.16667</td>
</tr>
<tr>
<td>45</td>
<td>1980</td>
<td>1</td>
<td>0.1934</td>
</tr>
<tr>
<td>55</td>
<td>2970</td>
<td>1</td>
<td>0.11818</td>
</tr>
<tr>
<td>65</td>
<td>4160</td>
<td>1</td>
<td>0.07489</td>
</tr>
<tr>
<td>75</td>
<td>5550</td>
<td>1</td>
<td>0.063559</td>
</tr>
<tr>
<td>85</td>
<td>7140</td>
<td>1</td>
<td>0.063025</td>
</tr>
</tbody>
</table>

3.7 Conclusion

Due to their potential harmful effects, identifying relay nodes in the network can improve security policy enforcement. In this work, the delay constrained relay node detection problem is investigated. A statistical solution, which has linear time and space complexity, is proposed. The proposed algorithm is lightweight and simple, therefore it is scalable and can be used in large scale implementations which may require real time detection.

For some applications identifying relay nodes may be sufficient. If an application requires...
Table 3.3: Detection performance for computer lab traffic

<table>
<thead>
<tr>
<th>$\beta$ tolerance</th>
<th>$\Psi_{TH}$</th>
<th>True Pos. Rate</th>
<th>False Pos. Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
<td>0.99716</td>
<td>0.65842</td>
</tr>
<tr>
<td>15</td>
<td>210</td>
<td>0.99902</td>
<td>0.21702</td>
</tr>
<tr>
<td>25</td>
<td>600</td>
<td>0.99901</td>
<td>0.065574</td>
</tr>
<tr>
<td>35</td>
<td>1190</td>
<td>0.99901</td>
<td>0.047619</td>
</tr>
<tr>
<td>45</td>
<td>1980</td>
<td>0.99979</td>
<td>0.14336</td>
</tr>
<tr>
<td>55</td>
<td>2970</td>
<td>0.99897</td>
<td>0.16558</td>
</tr>
<tr>
<td>65</td>
<td>4160</td>
<td>0.99785</td>
<td>0.12232</td>
</tr>
<tr>
<td>75</td>
<td>5550</td>
<td>0.99891</td>
<td>0.1137</td>
</tr>
<tr>
<td>85</td>
<td>7140</td>
<td>0.9989</td>
<td>0.10256</td>
</tr>
</tbody>
</table>

flow level relay identification, one of the existing relay flow detection techniques can be sub-
sequently applied to the relay nodes which have been flagged by the proposed method. The
contribution of this work is then in terms of computational complexity, as quadratic time relay
flow detection algorithms now have to be executed only for flagged nodes rather than every
node in the network.

Experimental results show that the proposed scheme is quite robust against various possible
adversarial or non-adversarial modifications on the underlying network traffic. In summary, the
experiments reveal that the proposed scheme can withstand, up to some extent, packet delays
which could be introduced due to packet processing in the relay node or for adversarial purposes.
Also the algorithm is shown to be able to detect relay activity even if the flows contain chaff
packets intended to defeat relay detection systems.

There are few limitations of the proposed scheme. First of all relay nodes should be delay-
constrained. That is to say, if incoming packets were buffered long enough before they are
forwarded, proposed algorithm would not be able to detect. Also it is assumed that the flows
are relatively sparse such that they are active for some of the time slots and inactive for others.
Otherwise, if a flow were continuously active within all observed time slots, then it would appear
as a relaying flow since it is simultaneously active with all the other flows. In the current setup
of the proposed scheme, if a node contains such flow, then it will be detected as a relay node.

There is a lot of work that still needs to be done. As part of our future work effort, we
plan to focus on methods which can increase the detection performance such that higher true
positives rates and lower false positive rates can be achieved. One possible way to achieve this
could be using a different alignment of time slots for each of the simultaneous $S$ calculations.
(i.e. time slot boundaries are not aligned for each simultaneous $Calculate_S$ execution). This could enable some of the calculated $S$ values to catch relay activity which the others might have missed. Also an adaptive selection of algorithm parameters, which adjust the parameters according to encountered traffic characteristics, might be very useful especially when the traffic characteristics tend to change over time. This could be done by sensing and analyzing the ongoing traffic and reacting accordingly.
Chapter 4

Correlated Flow Detection

4.1 Introduction

In this chapter, we consider real-time detection of correlated flows in network traffic. Although it can be used in various applications, our basic goal is to mitigate stepping-stone attacks. Our approach is based on data sketches, which are widely used in the context of streaming algorithms [21] [19] [46]. In general, these methods maintain short sketches of data streams, which are used to efficiently answer various queries about the data stream, such as rangesum, heavy hitters, quantiles etc. Adopting a similar philosophy, we propose a novel stepping-stone detection scheme based on an online algorithm, which continuously maintains sketches of network flows from a stream of captured packets at the border of a network. Using these sketches, the proposed scheme identifies correlated flows, and consequently stepping-stones, faster than the existing methods, without compromising robustness to timing perturbations, such as jitter and chaff.

As explained in previous chapter, stepping-stones are one of the effective strategies adopted by network perpetrators to maintain anonymity of an attack. In a stepping-stone strategy, instead of direct communication, an attacker uses a series of intermediate nodes, called stepping-stones, to relay her commands to a victim. Consequently, if the victim detects that he is under attack, he will only know that the attack packets are coming from the closest intermediate node.

An intermediate node of a stepping-stone chain essentially relays information from one of its
ingress flows to one of its egress flows. In general, it is possible to observe a certain correlation between relaying ingress-egress flow pairs, such as identical payload or similar packet timings. Therefore, one can detect stepping-stones in a network by searching for such correlations between ingress and egress flows at the network boundary. This kind of stepping-stone detection at network borders can be utilized in two major ways:

- If a pair of flows is detected to be a part of a stepping-stone chain, they can be blocked immediately to stop the attack, thereby preventing further harm.

- If the records of correlated flow pairs are collected from different networks, one can compile them to potentially traceback stepping-stone paths and identify the source of an attack. To give an example, “Cooperative Intrusion Traceback and Response Architecture (CITRA)” [80] is a framework that potentially benefits from a flow correlation based stepping-stone detection running at network borders. CITRA enables firewalls, routers, intrusion detection systems etc. from different networks to collaborate and exchange information using the “Intrusion Detection and Isolation Protocol (IDIP)” [79]. It’s objective is to trace intrusions across networks boundaries, as close as possible to the true origin, and to automatically generate immediate responses in order to prevent intrusions from causing any further damage. Hence, incorporating a flow correlation based stepping-stone detection in CITRA potentially enables both prevention and attribution of stepping-stone attacks.

A network-based stepping-stone detection scheme has to possess two key properties in order to be reliably employed:

i. **Efficiency/Scalability:** To detect and block ongoing attacks, a stepping-stone detection scheme should be able to identify correlated flows in real time. For this purpose, it has to process a dense packet stream, composed of numerous concurrent ingress and egress flows, very efficiently both in terms of computation and memory.

ii. **Robustness:** A stepping-stone detection scheme should be resistant to network imperfections (jitter, packet drops) and to various evasion techniques (chaff, random packet delays) often employed by attackers. In general, such perturbations disrupt the correlation
between flows of a stepping-stone chain, thereby potentially preventing a stepping-stone
detection scheme from detecting underlying timing correlations.

There are several stepping-stone detection schemes proposed in the literature. In ear-
lier, [110] [15] [108] [15] authors proposed several flow-correlation algorithms which can quickly
identify correlated flows based on simple timing features, such as packet counts, inter-packet
time difference etc. However, they provide very limited or no resistance to some of the afore-
mentioned timing perturbations, especially packet drops/retransmissions and chaff. On the
other hand, the schemes which are designed to resist chaff [109] [106] are relatively slower.
That is, in order to decide if given a pair of flows are correlated, they essentially try to find
a matching packet on one flow for each packet on the other one. However, comparing a pair
of flows in linear time in the number of packets and doing it for every pair of ingress-egress
flows does not scale to moderate to large networks. Another potential scalability issue is that
existing schemes are not designed to work directly on packet streams. Basically they need to
continuously reconstruct flows from a packet stream and frequently compare each of the active
m ingress flows with each of the active n egress flows in $O(nm)$ time, which potentially raises
several memory and computation issues.

In this work, we aim to design an efficient stepping-stone detection scheme without signif-
icantly compromising resistance against timing perturbations. For this purpose, we propose a
stepping-stone detection scheme based on flow packet-timing sketches. A packet-timing sketch
of a flow is a short, constant-length integer array, which summarizes the flow’s packet-timing
information. The proposed stepping-stone detection scheme continuously maintains succinct,
constant-length sketches of active flows’ packet-timing information from a stream of captured
packets at a network border. These sketches are then used to efficiently identify correlated
flows. The proposed flow sketches are maintained very efficiently by a streaming algorithm.
The algorithm performs a few arithmetic operations for each packet, thereby allowing simulta-
neous sketching of thousands of concurrent active flows from a packet stream. In addition, the
sketches of a pair of correlated flows remain similar, even if the flows encounter various timing
perturbations. Hence, the proposed scheme is able to detect the correlation between flows of
a stepping-stone chain under the presence of random delays and chaff packets. Part of this
chapter has been published in [24] and [25].

To demonstrate the efficacy of the proposed scheme, in this chapter we present various experimental results where we used real network traces. In addition, we also mathematically explain and analyze the proposed stepping stone detection scheme, where we make the following contributions:

- We prove that, given chaff rate and maximum packet delay values, the difference between sketches of a pair of correlated flows has an upper-bound in the expectation sense. This upper-bound justifies that the proposed scheme is expected to be able to identify correlated flows with high probability as long as the introduced packet delays and chaff are within acceptable limits.

- Exploiting the fact that the sketches of correlated flows are similar, we show that the proposed stepping-stone detection scheme can find correlated flows, with high probability, among \( m \) ingress and \( n \) egress flows in \( O(n + \sqrt{mn}) \) time.

The remaining of this chapter is organized as follows: We begin with presenting related work in Section 4.2. In Section 4.3, we present preliminaries and define the problem formally. Then, in Section 4.4, we explain the proposed stepping-stone detection scheme in greater detail. Following that, to demonstrate the efficacy of the proposed scheme, we present our experiments and results in Section 4.7.3. We present the limitations of the proposed scheme and possible solutions in Section 4.6. In Section 4.7, we present a possible extension of the proposed algorithm on Voice over IP traceback applications. Finally, we present conclusions and future work in Section 4.8.

### 4.2 Related Work

**Stepping-Stone Detection:** The literature on stepping-stone detection was summarized in the previous chapter. Interested readers should refer to Chapter 3.

**Data Stream Sketching:** In general, data sketching techniques can be viewed as linear projections of an input stream on appropriate basis functions [21]. They are widely used to
answer efficient queries on streaming data such as rangesum, heavy hitters, quantiles, inner product etc., [19] [45] [46].

**Robust Multimedia Hashing:** The proposed technique shows a resemblance to robust hashing schemes. They both represent an input signal by a short array (robust hash), which is resistant to small perturbations on the input. In the context of multimedia signal processing, robust hash functions are often used to identify and authenticate multimedia contents (audio, video and image) in the presence of perceptually preserving modifications such as compression, minor filtering etc. [41,52,95].

### 4.3 Preliminaries And Problem Definition

We begin with defining few concepts. Some of these concepts have already been defined in previous chapters but for the sake of self completeness we repeat them in this chapter once more.

**Definitions:**

- **A network flow** is defined as the collection of packets having the common five-tuples of source IP, source port, destination IP destination port and protocol (UDP or TCP).
- **An active flow** is defined as a flow which transmits its most recent packet not more than $T_{MaxIdle}$ seconds ago. $T_{MaxIdle}$ is the maximum idle threshold and we set $T_{MaxIdle} = 60$ seconds in this work.
- **An ingress (or egress) flow** with respect to a network is a flow which comes into (or goes out of) the network. More specifically, the destination IP of the packets of an ingress flow is within the network whereas that of egress flows is out of the network.

**Stepping-Stone Attacks:**

A stepping-stone attack incorporates several consecutive flows established between intermediate nodes, as illustrated below, where the attacker ($A$) first makes a connection to an intermediate node ($Nd_1$); then from $Nd_1$ he makes another connection to another intermediate node ($Nd_2$) and so on all the way to the victim.

$$A \rightarrow Nd_1 \rightarrow Nd_2 \rightarrow \ldots \rightarrow Nd_{h-1} \rightarrow Nd_h \rightarrow V$$
This chain of connections is referred as a **stepping-stone chain** and the intermediate nodes are often called **stepping-stones**. Each connection between two successive nodes on a stepping-stone chain is a separate network flow. These flows relay information from $A$ to $V$ through intermediate nodes. However, one cannot observe this directly from their payloads, as each flow is assumed to be encrypted with a different key in most cases, such as an SSH tunnel based stepping-stone chain. The response of $V$ to $A$’s packets might be relayed back to $A$ through a series of flows with reverse directions, such as “$A \leftarrow Nd_1 \ldots \leftarrow Nd_h \leftarrow V$”, or might be relayed through a completely different path.

Throughout this work we assume the attacker is constrained by a **maximum tolerable delay** [34], since he needs to interactively communicate with the victim. In other words, packets cannot be delayed more that the maximum tolerable delay along a stepping stone chain. As a result, the flows on a stepping-stone chain have similar packet timing characteristics, which allows timing based stepping-stone detection systems to identify stepping-stones in a network.

**Packet Delays, Jitter, Retransmissions and Chaff:**

There are various factors that potentially perturb the similar packet-timing characteristics among the flows of a stepping-stone chain, thereby making their discovery difficult. For instance, networks might introduce jitter on packet timings or packets might be delayed at intermediate nodes since they might need to first process each packet (i.e. re-encryption) before relaying. In addition, some packets might be dropped in the network and retransmitted. On the other hand, attackers might introduce intentional random packet delays to disrupt the correlation between the flows of a stepping-stone chain in the hope of evading detection. Attackers might also introduce superfluous packets, called **chaff**, which contain no valuable information and are not relayed to the succeeding flow on the chain. A packet-timing based stepping-stone detection scheme should be resistant to such perturbations.

**Real-Time Stepping-Stone Detection Problem Definition:**

A stepping-stone detection system, monitoring network traffic at the network border, observes numerous active ingress and egress flows at any given time. Let $I_t^i$ and $E_t^j$ denote these active ingress and egress flows at time $t$, respectively, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$. If there’s an active stepping-stone attack passing through the network, there should be at least one pair of ingress/egress flows, which have similar timing characteristics. More formally, there should
be at least one flow pair, \( \{ I^i_t, E^j_t \} \), such that \( \text{Diff}(I^i_t, E^j_t) < T \), where \( \text{Diff}(\ldots) \) is some packet-timing difference measure based on the packets observed so far and \( T \) is a threshold value. We refer the flows of a such ingress/egress flow pair having similar packet-timings as **correlated flows**.

Notice that, \( I^i_t \) and \( E^j_t \) don’t necessarily have a common endpoint, since two successive intermediate nodes might be in the same network and hence the flow between them is not observable at the network border. Hence, one needs to consider that every pair of ingress/egress flows is a potential correlated flow pair, thereby forcing an \( O(nm) \) time search. As a result, the stepping-stone detection problem at time \( t \) can be defined as: *given the flows \( I^i_t \) and \( E^j_t \), find all ingress/egress flow pairs \( \{ I^i_t, E^j_t \} \), such that \( \text{Diff}(I^i_t, E^j_t) < T \).* We refer to the procedure of solving this problem as **correlated flow search**. Finally, since the network has to be continuously monitored, the correlated flow search has to be periodically repeated at times \( t, t + \Delta, t + 2\Delta, t + 3\Delta, \ldots \text{etc} \), where \( \Delta \) should be selected small enough such that, a stepping-stone attack shouldn’t be able to start and finish between two search procedures and evade detection.

Solving the real-time stepping-stone detection problem, especially for large networks, is a challenging task as the solution has to be:

- **Memory Efficient:** A stepping-stone detection scheme should consume minimal amount of memory for each flow in order to be able to process hundreds of thousands of active connections.

- **Computationally Efficient:** A stepping-stone detection scheme should complete each correlated flow search procedure within \( \Delta \), before the subsequent search begins. Therefore, it should employ a correlated flow search algorithm faster than \( O(mn) \).

### 4.4 Sketch-Based Stepping Stone Detection in Real-Time

In this section, we present a real-time stepping stone detection scheme. In summary, the proposed scheme employs an online algorithm to continuously maintain packet-timing sketches of active flows from a stream of packets. As a result, the scheme is **memory efficient**, since,
4.4.1 Flow Packet-Timing Sketch

Overview and Basic Idea

In order for the proposed scheme to be effective, sketches of correlated flows should be as similar to each other as possible, whereas those of uncorrelated flows should be as different from each other as possible. In addition, for efficiency, these sketches are required to be succinct.

Sketch Computation: One way to obtain a sketch with such properties is to compress packet-timing characteristics of a flow using a linear transformation. A linear transformation maps an input vector from one space to another space by projecting input vector onto a set of basis vectors. The projection values, which are called the transform coefficients, highly depend on the dynamics of the input vector. Therefore, transform coefficients of similar input vectors will be also similar to each other. As a result, one can select a small set of linear transform coefficients for each flow as its sketch, and use it to distinguish between correlated
and uncorrelated flows. Based on this strategy, in this work, we compute the sketch of a flow as follows: i) We first convert the packet-timing information of the flow to a standard vector representation, which we call the packet-count vector. Elements of a packet-count vector, which will be explained in the next subsection, are basically the number of packets that the flow transmits in consecutive time-slots. It is clear that, correlated flows have similar packet-count vectors whereas uncorrelated flows have different. An example case is illustrated in Figure 4.1, where the packet-count vectors of two correlated flows and the difference between them is shown in Figure 4.1(a). On the other hand, the difference between packet-vectors of two uncorrelated flows is shown in Figure 4.1(b). ii) Then we apply a random linear transformation, whose basis vectors are composed of random integers, to the packet-count vector and obtain a small set of coefficients as the sketch of the flow. The resulting sketch is an integer array and we refer this sketch as the integer-array sketch of the flow.

Efficient Search for Correlated Flows: When it comes to the correlated flow search, we use binarized sketches. The binarized (or binary) sketch of a flow is nothing but the signs of the elements of its integer-array sketch. They enable us to efficiently search for the correlated flows using the Hamming Distance. The basic idea of the proposed efficient search is that although the binary sketches of two correlated flows are not exactly the same, a short random subset of their sketch bits match exactly with high probability. Therefore, instead of comparing the sketches of every ingress flow with that of every egress flow, one can compare only the pairs whose certain bits match with each other and thus end up comparing pairs of correlated flows, with high probability. The details of the algorithm is given in Section 4.4.2.

“Packet-Count Vector” Representation

To employ a linear transformation, we first need to represent the flow’s packet-timing information as a vector. For this purpose, we consider the time axis as a series of non-overlapping consecutive time slots. Then, using these time-slots, we define the packet-count vector of a flow as the number packets that the flow transmits at each time-slot. More formally, let $L_{TS}$ denote the length of these time-slots forming the time axis. Then time slot $t$ is defined as the $t^{th}$ time interval after an epoch ($T_{epoch}$) such that $[T_{epoch} + (t - 1)L_{TS}, T_{epoch} + (t)L_{TS}]$. Based on these time-slots, we can specify the packet-count vector of flow $F$ as $V_F$, such that $V_F(t)$ is
equal to the number of packets that flow $F$ transmits during time-slot $t$.

Conceptually $V_F$ is an infinite length vector ranging from $t = -\infty$ to $t = \infty$. However, it is obvious that $V_F(t)$ can get non-zero values only during the lifetime of flow $F$, as $F$ transmits no packet before it begins or after it ends.

**Random Linear Transformation and Integer-Array Sketch**

After computing the packet-count vector of a flow, we apply a random linear transformation to obtain the integer-array sketch, by projecting the packet-count vector $V_F$ onto the $k$ random basis vectors $B_{i=1,2,...,k}$, as follows:

$$C_F(i) = \sum_{t=-\infty}^{\infty} B_i(t)V_F(t)$$

(4.1)

where the elements of random bases are random numbers drawn from a Bernoulli distribution with $p = 0.5$ such that:

$$Pr(B_i(t) = 1) = Pr(B_i(t) = -1) = \frac{1}{2}$$

(4.2)

Notice that, although the summation is from $t = -\infty$ to $t = \infty$, we only need to sum over the time slots during which flow $F$ transmitted at least one packet (i.e. when $V_F(t) \neq 0$). Therefore, the coefficients can be computed in real time as the packets arrive, as explained in Section 4.4.1.

**Binarizing the Integer-Array Sketch**

While continuously maintaining the integer-array sketch of a flow, we perform correlated flow search based on binary sketches, which allow us to design faster search algorithms. For this purpose, we simply compute the binary sketch from the signs of these coefficients when needed. More specifically the $i^{th}$ bit of the sketch of flow $F$ is computed as:

$$S_F(i) = \begin{cases} 
1, & \text{if } C_F(i) > 0 \\
0, & \text{if } C_F(i) \leq 0 
\end{cases}$$

(4.3)
Resistance of Binary Sketch to Timing Perturbations

It is important for the proposed scheme that a binary sketch stays similar even though the flow encounters timing perturbations, such as packet delays and chaff. To mathematically investigate this, we consider a pair of correlated flows, namely $F$ and $F'$, representing two flows on the same stepping-stone chain. Since they are correlated, the Hamming Distance between the binary sketches of $F$ and $F'$ is required to be low, such that \( \text{HamDist}(S_F, S_{F'}) < \text{threshold} \). Or equivalently, the bit-error probability between the sketches of $F$ and $F'$ should be low. To represent bit-error probability between $S_F$ and $S_{F'}$, we define the error probability of $i$th bit location, $P_{e_i}^{[F,F']}$, as the probability of $i$th bit of $S_F$ and $i$th bit of $S_{F'}$ being different than each other. Combining this with Equation (4.3) we can write:

\[
P_{e_i}^{[F,F']} = P_F [S_F(i) \neq S_{F'}(i)] = P_F [\text{sign}(C_F(i)) \neq \text{sign}(C_{F'}(i))] \tag{4.4}
\]

To determine this probability with respect to the difference between $F$ and $F'$, we denote the difference (or error) between the packet-count vectors of $F$ and $F'$ by $\mathcal{E}$ as, such that:

\[
\mathcal{E}(t) = V_{F'}(t) - V_F(t) \tag{4.5}
\]

Then, combining this with Equation (4.1), we can write $i$th linear transform coefficient of $F'$ as:

\[
C_{F'}(i) = \sum_{t=-\infty}^{\infty} B_i(t) [V_F(t) + \mathcal{E}(t)] = C_F(i) + \sum_{t=-\infty}^{\infty} B_i(t) \mathcal{E}(t) \tag{4.6}
\]

Recall that, a bit-error occurs only when $\text{sign}(C_F(i)) \neq \text{sign}(C_{F'}(i))$. And this only happens when $|\sum_{t=-\infty}^{\infty} B_i(t) \mathcal{E}(t)| > |C_F(i)|$ and $\text{sign}(\sum_{t=-\infty}^{\infty} B_i(t) \mathcal{E}(t)) \neq \text{sign}(C_F(i))$. Therefore, we can write:

\[
P_{e_i}^{[F,F']} \leq P_F \left[ \left| \sum_{t=-\infty}^{\infty} B_i(t) \mathcal{E}(t) \right| > |C_F(i)| \right] \tag{4.7}
\]

Since $B_i(t)$ values are i.i.d. Bernoulli random variables with $\mu = 0$ and $\sigma^2 = 1$ as presented
in Equation (4.2), one can derive that $\sum_{t=-\infty}^{\infty} B_i(t) \mathcal{E}(t)$ is also a random variable with $\mu = 0$ and $\sigma^2 = \sum_{t=-\infty}^{\infty} \mathcal{E}(t)^2$. Hence, from Equation (4.7) and Chebyshev Inequality, we can write:

$$P_{e_i}^{[F,F']} \leq \frac{\sum_{t=-\infty}^{\infty} \mathcal{E}(t)^2}{|C_F(i)|^2} \quad (4.8)$$

The above bound indicates that, as a pair of correlated flows become more different than each other, the bit-error probability between their binary sketches increases. It also shows that as the magnitude of linear transform coefficients increases, which essentially means that the flows transmit more packets, their sketches become more resistant to perturbations.

To visualize how this bound changes with respect to delay and chaff, we computed $\sum_{t=-\infty}^{\infty} \mathcal{E}(t)^2$ and $|C_F(i)|^2$ for 1000 correlated flow pairs. We used 512-bit sketches, therefore we have $i = 1,2,...,512$. We obtained correlated flows by modifying 1000 original flows, which are 60-second long SSH flows captured at our network border, with uniformly distributed packet delays and random chaff insertions. Recall that, the bound is different for each bit position depending on the magnitude of the corresponding transform coefficient. Hence, for each correlated flow pair we computed the bit-error probability in expectation sense by using average coefficient magnitude, such that $P_{e_{exp}} = \frac{\sum_{t=-\infty}^{\infty} \mathcal{E}(t)^2}{\sum_{i=1}^{512} |C_F(i)|^2}$. Figure 4.2 plots the average $P_{e_{exp}}$. 

Figure 4.2: The upper-bound for average Bit-Error Probability of a sketch for different maxDelay and chaffRate.
of all 1000 flow pairs for different maxDelay and chaffRate values. As expected, the average bit-error probability bound increases with both increasing max delay and increasing chaff rate. It is also observed that, the Hamming Distance between the sketches of a pair of correlated flows is expected to be less than a threshold as long as maxDelay and chaffRate don’t exceed a certain value.

**Online Computation of Packet-Timing Sketches**

In this subsection we present an efficient algorithm to simultaneously compute packet-timing sketches of all observed flows. The algorithm computes sketches continuously in a cumulative way. Therefore, the sketch of a flow with respect to the packets received so far is available at any desired time even if the flow is still active. The essence of the algorithm is to cumulatively compute Equation (4.1) by updating linear transform coefficients ($C_F(i)$) for each captured packet. Since $V_F(t)$ indicates the number of packets that flow $F$ has in time-slot $t$, Equation (4.1) essentially accumulates the random basis vector values for the time-slots that the observed packets are transmitted. More formally, for each packet $p$ of flow $F$, the algorithm updates transform coefficients as follows:

$$C_F(i) \leftarrow C_F(i) + B_i(t_p)$$  \hspace{1cm} (4.9)

where $t_p$ indicates the time-slot which packet $p$ is transmitted. The pseudo-code for the algorithm performing this procedure for all flows is given below. The algorithm runs on the packet stream $\Phi$ and take three inputs namely the epoch ($T_{Epoch}$), length of the time-slots ($L_{TS}$), and the number of transform coefficients—equivalently the length of binary sketches—($k$) for each flow. The subroutines getFlow() and getTimeStamp() extracts the flow information that the packet belongs to and the time stamp from a a packet respectively. The algorithm might compute the random basis vector elements “$B_i(t)$” online whenever needed. However, since packets arrive roughly in chronological order, $B_i(t)$ values can be pre-computed for a sliding window of time and stored in a cache repository for practical purposes.

The MaintainSketches algorithm continuously maintains an integer-array sketch for each active flow. If the binary sketch of a flow is required at any time, the algorithm simply computes


Algorithm 4.1 MaintainSketches($T_{Epoch}$, $L_{TS}$, $k$)

\begin{algorithm}
\For{all packet $P$ captured on stream $\Phi$} {
\begin{algorithmic}
\State $F \leftarrow \text{getFlow}(P)$ \{Determine the flow of $P$\}
\State $t = \left\lfloor \frac{\text{getTimestamp}(P) - T_{Epoch}}{L_{TS}} \right\rfloor$ \{Determine current time slot\}
\For{$i = 1$ to $k$} \{ 
\State $C_F(i) \leftarrow C_F(i) + B_i(t)$
\EndFor
\EndFor
\end{algorithmic}
\end{algorithm}

From the signs of the current value of integer-array sketch of that flow as explained in the previous subsection. The above algorithm simply updates $k$ coefficients of the corresponding flow for each packet. Since $k$ is constant (typically $k = 512$), the algorithm runs in linear time in the number of packets. Also, the algorithm requires linear space in the number of active flows since a constant length integer array is maintained for each active flow.

4.4.2 Efficient Search for Correlated Flows

In this subsection, we present an approximate search algorithm based on binary sketches, which finds all correlated flows in $O(n + \sqrt{nm})$ time, with high probability. The algorithm is similar to the algorithm proposed in [52], where the authors search a large audio database using robust audio hashes. The algorithm steps are given below:

1. We first pick $\alpha$ random bit locations on binary sketches, where $2^{\alpha} = \sqrt{nm}$. We refer the bits on these random locations of a flow’s sketch as the subsketch of that flow. More formally, let $b_1, b_2, \ldots, b_\alpha$ be these random bit locations, where $k$ is the sketch length and $\alpha \leq k$. Then $[S_F(b_1), S_F(b_2), \ldots, S_F(b_\alpha)]$ is the subsketch of flow $F$.

2. Then for $n$ ingress flows, we construct a lookup table, which is used to access an ingress flow from its subsketch. Note that the size of the lookup tables is $2^{\alpha} = \sqrt{nm}$, since a subsketch can have $2^{\alpha}$ possible values.

3. Following that, for each egress flow $F_e$, we compute the Hamming Distance between the binary sketch of $F_e$ and the binary sketches of ingress flows whose subsketch is the same as the subsketch of $F_e$. Notice that, we use the lookup table constructed in the previous step to find the ingress flows with the matching subsketches. If any of the Hamming Distances is smaller than the threshold $th$, $F_e$ and that particular ingress flow are declared as correlated.
Finally, in case we miss comparing some of the correlated ingress/egress flow pairs, we repeat the above steps $\beta$ times, each time with a different subsketch bit locations.

Recall that $n$ and $m$ are the number of ingress and egress flows respectively. Hence, the proposed search algorithm computes the lookup table in $O(n)$ time. After that, it compares each of the $m$ egress flows with on average $n/2^\alpha$ flows, since on average $1/2^{\alpha \text{th}}$ of the ingress flows matches the subsketch of a given egress flow. And the algorithm repeats this $\beta$ times. As a result, the proposed search algorithm runs in $O((n + mn/2^\alpha)\beta)$ time in the expectation sense. Since we set $2^\alpha = \sqrt{nm}$, and $\beta$ is typically a small constant, the proposed algorithm’s run time is $O(n + \sqrt{nm})$.

The proposed search algorithm can find correlated flow pairs only if their subsketches match in at least one of the $\beta$ repetitions. To find the probability of that event, let $P_{\text{match}}$ denote the probability of two flows, whose sketches are closer than $th$, having the same subsketch. Then we can write that, $P_{\text{match}} \geq \prod_{i=0}^{\alpha-1} \frac{k-th-i}{k-i}$, where $k$ is the sketch length. Using this, we can write the probability of the proposed search algorithm finding an ingress/egress flow pair, the Hamming Distance between whose sketches are smaller than $th$, in $\beta$ repetitions as $P_{\text{find}} = 1 - (1 - P_{\text{match}})^{\beta} \geq 1 - \left(1 - \prod_{i=0}^{\alpha-1} \frac{k-th-i}{k-i}\right)^{\beta}$. It is clear that $P_{\text{find}}$ approaches to 1 as $\beta$ increases. To give an example, in Figure 4.3, we plot $P_{\text{find}}$, for different $mn$ and $\beta$, where we set $2^\alpha = \sqrt{mn}$, where the sketch length is $k = 512$ and the threshold is $th = 71$. Threshold was set to allow $10^{-4}$ false positive rate as will be explained in Section 4.7.3. It is observed that $P_{\text{find}}$ climbs rapidly with $\beta$, and when $\beta \approx 50$, the proposed search algorithm is expected to find almost all flows whose sketches are closer than $th$.

4.5 Experiments and Results

To demonstrate the efficacy of the proposed stepping-stone detection method we set up an experiment, which measures how successfully the correlated flows can be detected under a fixed false alarm probability.

Obtaining Correlated Flows: In most cases, stepping-stone attacks are carried over an interactive protocol, such as SSH. Therefore, to obtain correlated flows, we first captured 100 real SSH flows, at our network’s border. We refer these flows as the original flows. We
observed that the original flows transmitted 2.4 packets per second on average. Then for each of these original flows, we obtained a **perturbed flow**, by delaying packets and introducing chaff. Despite the delay and chaff, a perturbed flow is considered to be correlated to the flow it originated from. To obtain a perturbed flow from an original flow, first we delayed each packet of the original flow by a random amount chosen uniformly from the interval \([0, \text{maxDelay}]\). Then, we introduced chaff packets to both the original flow and the perturbed flow at random times, where the ratio of the number of introduced chaff packets to the number of original packets was determined by \( \text{chaffRate} \).

**Searching for Correlated Flows:** Once we obtained these 100 correlated flow pairs, we blended them into the real network network trace, which was previously captured at our network’s border during a typical weekday. Then, we ran the proposed stepping-stone detection technique on the blended trace as if it was a real-time network traffic. The proposed technique continuously maintained the sketches of the active flows in the network trace as packets were captured. To deal with terminated flows, we checked whether the flows were still active once in every minute. A flow and it’s sketch were erased if it was idle for more than 60 seconds. Finally, to identify the correlated flows in the trace, we performed the proposed correlated flow
search at every $\Delta = 10$ seconds. We believe $\Delta = 10$ is reasonable because, it is very hard for an attacker to start and finish her attack within 10 seconds. During the search, we used 16-bit subsketches and 50 lookup tables, such that $\alpha = 16$ and $\beta = 50$ as explained in Section 4.4.2. With those specific $\alpha$ and $\beta$ values, we were able to perform the search in $O(n + \sqrt{nm})$ time for up to $n = 10^5$ active ingress and $m = 10^5$ active egress flows, since $(2^{16})^2 \approx 10^{10}$.

We declared a pair of flows as correlated, when we found out in any of these periodic search processes that the Hamming Distance between the binary sketches of those flows were below a threshold $th$. Finally, to quantify the performance of the proposed scheme, we measured the detection rate as the ratio of the number of correctly identified correlated flows to the number of all correlated flows blended into the trace.

**Selecting the Threshold:** A proper setting of the detection threshold $th$ was crucial for our experiments. Like all detection methods based on thresholds, the lower the threshold the higher the false alarm rate and the higher the threshold the lower the detection rate. In our experiments, we picked the detection threshold, which yielded the fixed $10^{-4}$ false alarm rate. We believe that, $10^{-4}$ false alarm rate is acceptable in most cases. To determine a such threshold, one needs to estimate the probability distribution of the Hamming Distances between binary sketches uncorrelated flows. For this purpose, we first empirically computed the probability distribution of Hamming Distances between the binary sketches of 5000 uncorrelated SSH flow pairs captured at our network’s border. During that process, we made sure that the flows were concurrent, by shifting them in time. This enabled us to produce a more realistic estimate, since a stepping-stone detection scheme searches concurrent flows for correlated pairs.

Once we computed the empirical distribution, we fitted a Gaussian on the empirical distribution. Then we picked the threshold value $th$ such that, the integral of the fitted Gaussian from $-\infty$ to $th$ was equal to $10^{-4}$, such that $\int_{-\infty}^{th} f_G(h) \, dh = 10^{-4}$. We plot the computed empirical distribution, fitted Gaussian and the selected threshold value for 512-bit sketches in Figure 4.4, where we used 500 millisecond time-slots.

Notice that, the distribution of Hamming Distances between the sketches of uncorrelated flows should ideally be a Binomial distribution with $p = 0.5$. However, the empirical distribution deviated from the ideal as observed in Figure 4.4. The reason is that, even though the flows were uncorrelated, their packet-count vector representations show some coarse similarities. For
Figure 4.4: The empirical probability distribution of the Hamming Distances between the binary sketches of uncorrelated flows (durations between 10 and 40 seconds) and the Gaussian probability distribution function approximating the empirical distribution. The vertical line marks the threshold value (i.e. \( th = 71 \)) for false alarm rate of \( 10^{-4} \).

instance, since we used quite long time slots (i.e. 500 milliseconds), there’s a good chance that a pair of uncorrelated flows share many common time slots that they are both active in, although they usually have different number of packets within a given time-slot.

**Real-Time Exclusions in the Correlated Flow Search:** In our experiments we excluded some flows in the correlated search process, since they had no potential to be a part of a stepping-stone attack. Our concern was not about the computational efficiency but rather about the detection performance, since such flows might introduce extra false positives. In order to be real-time, we identified and excluded such flows in real time as well. In our experiments, we excluded the following cases:

- **Flows With Reverse Directions:** In most cases, two hosts communicate with each other through a pair of flows (one for each direction), where one flow’s source IP and port is other flow’s destination IP and port. Although, such a pair of flows usually have similar packet timings to each other (i.e. TCP packets and their ACK’s), they are not part of a stepping stone attack.

- **Too Short Flows:** A successful stepping-stone attack is expected to last long enough to
allow the attacker exchange sufficient information with the victim over an interactive protocol. Hence, too short flows are highly unlikely to be a part of a stepping-stone attack and therefore excluded in the comparison process. To identify short flows, we used the number of time-slots in which the flow was active, such that, we excluded a flow in the search process if it was active in less than 20 time-slots.

- **Flows With Insufficient Packet-Timing Information:** In some cases, a flow might have no distinctive packet-timing information, such as when downloading a file or streaming a video etc. Usually such flows all have constant number of packets at each time-slot and therefore appear to be correlated with each other. Hence, such flows should be excluded in the search process and treated separately. To detect such flows in real-time, we simply checked the fraction of the time-slots in which a flow is active during its lifetime. We observed that, most of the flows having insufficient packet-timing information were active more than 90% of the time-slots.

**Detection Rate Results:** In our experiments, we computed the detection rate of the proposed scheme under various `maxDelay` and `chaffRate`. Figure 4.5 plots these measured detection rates, where we used 500 millisecond long time-slots and 512-bit sketches. The duration of the
correlated flows were 60 seconds. It is observed that the proposed scheme resists delays and chaff packets to some extent. For instance, the algorithm detected 95% of the correlated flow pairs when the \textit{maxDelay} = 100 milliseconds and \textit{chaffRate} = 0.1. However, further increasing chaff rates and delays decreases the detection rate, as expected since, the packet-count vectors of correlated flows deviate from each other as chaff rate and max delay increase.

**Effect of Time-Slot Length:** Time-slot length is an important parameter, which sets a trade-off between resistance to delays and ability to distinguish between uncorrelated flows. If the time-slot length was set too short, then even very slight packet delays would result in packets drifting into subsequent time-slots. As a result, packet-count vectors, and therefore flow sketches, would change drastically even under a slight packet delays. On the other hand, making the time-slots too long would result in the packet-count vectors of some uncorrelated flows being similar to each other. Our experiments suggest that, 500 millisecond long time-slots are an appropriate choice for our setting. Figures 4.5 and 4.6(a) depict that, using shorter time-slots (i.e. 300 milliseconds) weakens the resistance to the packet delays since detection rate is observed to decrease more rapidly for increased \textit{maxDelay} in Figure 4.6(a) than in Figure 4.5. On the other hand, using longer time-slots i.e. 1000 milliseconds) makes the sketches of uncorrelated flows similar to each other. As a result, the $10^{-4}$ false alarm rate threshold becomes smaller and therefore the detection performance decreases again, as observed in Figure 4.6.
4.6(b).

**Effect of Sketch Length:** Using short sketches is very important for the proposed scheme to be scalable. However, using too short sketches makes it harder to distinguish between correlated and uncorrelated flows. The reason is that, shorter sketches contain less information about the packet-timings of flows, thereby yielding increased probability of the sketches of two uncorrelated flows being similar to each other. More formally, when shorter sketches are used, the distribution of the Hamming Distances between the binary sketches of uncorrelated flows (Figure 4.4) will have higher variance, thereby pushing the $10^{-4}$ false alarm rate threshold towards zero and hence decreasing the detection performance. The improvement of the detection rate with the increased sketch length is observed in Figures 4.7(a), 4.5 and 4.7(b), where we used 256-bit, 512-bit and 1024-bit sketches, respectively.

**Effect of Correlated Flows’ Duration:** It is expected that, the detection rate increases as the duration of the correlated flows increases. That is because, longer flows have more packets and therefore their linear transform coefficients have higher magnitudes. As a result, same amount of distortion on packet-count vectors has less impact on the integer-array sketches of longer flows, as discussed in Section 4.4.1. Hence, sketches of longer flows are more resistant to timing perturbations. This can be observed in Figures 4.8(a), 4.5 and 4.8(b), where we present the detection rates for correlated flow durations 30, 60 and 90 seconds, respectively.
Figure 4.8: Effect of correlated flow duration for $= 500$ msec long time-slots and 512 bit binary sketches. It is observed that, increasing flow duration increases the detection rate.

Consequently one can say that, the longer an attacker stays on-line, the more likely the proposed scheme detects her attack.

4.6 Limitations And Potential Solutions

**High Chaff Rates:** Although, stepping-stone attacks are constrained by a maximum tolerable delay, in theory there is no such constraint on chaff. That is, an attacker can introduce as many chaff packets as she would like (as long as the network is able to handle) in order to disrupt the observed correlation between her flows. However, in most cases, sending packets at an unusually high-rate might expose the attack instead of concealing it. Nevertheless, such cases are potential problems, not only for the proposed scheme but for all timing-based flow correlation schemes (i.e. link padding algorithms in the context of anonymity networks [42] [96] [98]). To mitigate such cases, one needs to identify flows with unusually high packet-rates and treat them separately. If these cases are relatively rare, one can afford to employ very complex algorithms to process them. One possible solution is that, flows with unusually high packet-rates can be tested by an active flow correlation scheme [99] [76] [55], which marks an ingress flow by perturbing packet timings in a certain manner, and checks if any of the egress flows contain the embedded mark.

**Flow Splitting/Merging:** To evade detection, an attacker might employ more complex strategies, such as flow splitting or flow merging. In flow splitting, an attacker creates multiple egress
flows sharing outgoing attack packets, instead of using only one egress flow. Or similarly, in
flow merging, an attacker sends attack packets to the monitored intermediate node over multi-
ple ingress flows and merge them into one egress flow. To detect such cases, a stepping-stone
detection scheme should compare superpositions of several ingress flows with superpositions of
several egress flows for possible correlations. Unfortunately, such kind of search will be very
costly since every possible combination of the flows have to be tested. However, due to the linear
transformation employed by the proposed scheme, the integer-array sketch of the superposition
of a set of flows is equal to the summation of the sketches of those flows. This linearity property
enables quick and efficient computation of the binary sketch of any given combination of flows.
Hence, one can potentially combine this linearity property with an appropriate optimization
technique, such as dynamic programming, gradient search etc., to efficiently search for a small
set of ingress flows and egress flows which are collectively correlated with each other. We leave
the exploration of this issue as a future work.

4.7 Extension to VoIP Traceback

Aside from stepping-stone detection, the proposed flow correlation scheme can used in var-
ious source attribution or traceback problems. Tracing anonymized Voice over IP calls is one
such problem. In this section, we present a variant of the proposed flow-correlation scheme
which can be used on VoIP call traceback applications.

VoIP calls are often anonymized using cryptographic tools and various proprietary protocols
for privacy purposes. Furthermore most VoIP networks have peer-to-peer (P2P) architecture
(i.e. Skype) and VoIP packets pass through several peers before reaching their destinations.
Therefore, it is very hard to determine who is talking to whom on a VoIP network. However,
such information can be very crucial for many scenarios, especially in law enforcement. In such
cases, one could determine if two parties were talking to each other if he could identify audio
contents of VoIP calls by an identifier. For instance, suppose that the traffic of a number of
suspected networks are monitored. More specifically, some sort of content identifiers of VoIP
flows are stored for every host in those networks. Then, one can simply search the stored data
for pairs of concurrent VoIP flows having similar content identifiers, which suggest that the
corresponding parties are communicating with each other with high probability.

In this section we present a robust sketching scheme for VoIP flows to identify their audio content by short binary strings. Notice that, to identify the contents of a VoIP flow, one cannot use traditional audio hashing schemes [72,92], since in most cases VoIP flows are encrypted. Nevertheless, since the design purpose of the proposed sketching scheme lies along the same lines with robust audio hashing schemes, in this thesis we refer to the proposed VoIP sketching scheme as some type of a robust hashing scheme.

The proposed robust hash is based on VoIP flows’ packet timings and payload sizes, which are typically invariant under encryption and reflect the content information of VoIP flows. The basic idea is that, variable bitrate (VBR) audio codecs employed in most VoIP applications result in the packet timings and the payload sizes to be highly dependent on the contents of the input VoIP flow. This is because vowels typically require more bandwidth than consonants or fricative sounds [104]. Therefore, depending on the ordering of these sounds in an input audio signal, a VBR codec determines what size packets are sent on what times. Furthermore, this information is often invariant under encryption, since most VoIP applications employ length preserving ciphers.

In summary, the proposed robust hashing scheme takes the timestamps and payload sizes of the packets of a VoIP flow as the input and computes a short constant-length binary string, called the hash value, as the flow’s content identifier. However, in order this scheme to be employed in VoIP call tracking applications, it has to possess the following crucial properties:

- **Robustness:** Network flows often encounter various network impairments such as packet delays, jitter and packet drops. Therefore, the packet timings and variations on the bitrate slightly differs as a VoIP flow transmitted through networks. The proposed scheme should output the same or similar hash value even if a VoIP flow undergoes such impairments, in order to still be able to identify a VoIP flow at the other end of the network.

- **Low Collision-Probability:** Two different and uncorrelated VoIP flows should have completely different robust hash values.

- **Efficient Computation:** In order for the proposed scheme to be deployed in very large networks, where hundreds of thousands of VoIP have to be monitored, it has to compute
hash values very efficiently in real-time as the packets are captured. Furthermore, resulting hash values has to be short enough to avoid any scalability issues during storage or search.

In this section, we formally explain the proposed VoIP hashing scheme and demonstrate that it possesses the above properties. The remaining of this section is organized as follows: After presenting related work in Section 4.7.1, we formally discuss the details of the proposed robust hashing scheme in Section 4.7.2. Also in Section 4.7.2, we give an efficient algorithm which computes the proposed hash values in real-time. Then we present our experiments and results in Section 4.7.3.

4.7.1 Related Work

Wright et. al. exploited the information leakage through variable bitrate codecs in [105] to identify the spoken language in encrypted VoIP calls. In a more ambitious work [104], Wright et. al. were able to identify spoken phrases from a standard speech corpus in a VoIP call with on average %50 accuracy. In that work, authors trained a Hidden Markov Model using packet sizes of VoIP segments containing phrases in the corpus. These two works strongly suggests that audio contents of a VoIP call can be identified from the resulting packet sizes. Similar to robust hashing, in [24], Coskun and Memon propose a flow sketch to identify packet-timing characteristics of network flows for real time stepping-stone detection. In [100], Wang et. al. uses watermarking techniques to identify two communicating parties over VoIP. Their technique actively modifies packet timings of a VoIP flow and try the detect the same modification pattern on the other end of the network. However, real-time modification of packet-timings of all VoIP flows poses potential scalability issues for large networks.

4.7.2 Robust Hashing of VoIP Flows

In this section, we explain the details of the proposed robust hash function and a real-time algorithm, which efficiently computes robust hash of a VoIP flow. We begin with presenting a brief background.

**Background:**

In a typical VoIP call, call setup and voice transmission are handled by different network
flows. A standard protocol, such as Session Initiation Protocol (SIP), Extensible Messaging and Presence Protocol (XMPP), or a proprietary protocol (i.e. Skype) is often used for call setup. The actual voice data is typically transmitted using Real-time Transport Protocol (RTP) over UDP packets. In this work, we refer these UDP flows carrying actual voice data as **VoIP Flows**. More formally, a VoIP flow is defined as a collection of UDP packets which possess the same quadruple of (source IP, source port, destination IP, destination port). Transmitted voice data is often encoded by a Variable Bit Rate (VBR) codec to minimize average bit-rate thereby increasing overall service quality. To ensure low latency constraints, inter-arrival time between UDP packets is typically set between 10 and 50 milliseconds. The size of each UDP packet depends on the output bit-rate of the VBR codec.

**Robust VoIP Hash Overview:**

In the proposed robust hashing scheme, to compute the robust hash value of a VoIP flow, we follow three major procedures:

- **Representation:** Packet payload sizes of a VoIP flow varies according to the underlying audio content information. Therefore, in order to identify the audio content of a VoIP call, we first represent the variations on the bit rate of a VoIP flow across time by a sparse discrete signal called **bitrate variation signal**. The bitrate variation signal is basically shows the difference between payload sizes of consecutive packets at corresponding times. Hence, it captures both packet timings and bitrate variations of a VoIP flow in one dimension.

- **Projection:** To compute the hash value, we then project the bitrate variation signal onto pseudorandom smooth bases. Projecting the input signal onto smooth bases functions is widely used in multimedia hashing \[26\] \[41\], as it provides robustness to slight changes in the input.

- **Quantization:** Finally the signs the projection values are output as the resulting hash value. The basic intuition is that the slight modifications on the VoIP flow are usually not powerful enough to flip the signs of the projection values. On the other hand, the signs of projection values of two uncorrelated VoIP flows are expected to be uncorrelated.
In the following paragraphs, we present each procedure in formal details.

**Bitrate Variation Signal:**

To formally present bitrate variation signal, consider a VoIP flow with \( P \) packets, where the packets are in chronological order\(^1\). For these \( P \) packets, let \( T_i \) denote the global time-stamp of \( i^{th} \) packet as the number of milliseconds passed since the global epoch, where \( i = 0, 1, 2, ..., P-1 \). Using this, we represent the relative timestamp of \( i^{th} \) packet by \( \hat{T}_i \) indicating the elapsed time since the beginning of that particular flow, such that:

\[
\hat{T}_i = T_i - T_0 \tag{4.10}
\]

On the other hand, let \( B_i \) denote the size of the payload of the \( i^{th} \) packet in bytes. Then, we represent the difference of the sizes of consecutive packets with \( B_i^\Delta \), such that:

\[
B_i^\Delta = B_i - B_{i-1} \tag{4.11}
\]

Following that, to represent a VoIP flow’s bitrate variations over time we define the **bitrate variation signal** \((V)\) as follows:

\(^1\)Since the packets are captured in real time, they are essentially in chronological order by default.
\[ V(n) = \sum_{i=1}^{P-1} B_i^\Delta \left[ \delta(n - \bar{T}_i) \right] \]  

(4.12)

where \( \delta(n) \) is the Dirac delta function. Note that, \( V(n) \) is a discrete signal which has N-1 nonzero values at relative timestamps of all packets except the first one. Note that bitrate variation signal contains both packet-timing and packet payload size information, which represents the underlying content of VoIP flows. As an example, the bitrate variation signal extracted from 200 consecutive packets of a VoIP flow is illustrated in Figure 4.9.

**Smooth Bases and Hash Computation:**

Once the bitrate variation signal is constructed, the robust hash value is calculated by projecting \( V \) onto \( L \) smooth pseudorandom bases, which are denoted by \( R_l(n) \), such that:

\[ H_l = \sum_n V(n) R_l(n) \]  

(4.13)

where \( l = 1, 2, 3, ..., L \). Each pseudorandom base is generated independently by smoothing an array of i.i.d Gaussian random variables. To smooth pseudorandom arrays, we used a Gaussian lowpass filter with \( \sigma = 50 \). As an example, few of the bases we used are plotted Figure 4.10.

Finally, we use the signs of these projection values to compute the robust hash value of the input VoIP flow. More formally, each bit of \( L \)-bit robust hash \( (h) \) is computed by 1-Bit quantization of a corresponding projection value such that:

\[
    h_l = \begin{cases} 
        1, & \text{if } H_l \geq 0 \\
        0, & \text{if } H_l < 0 
    \end{cases}
\]  

(4.14)

where \( l = 1, 2, 3, ..., L \).

**Online Hash Calculation in Real-Time:**

In order the proposed scheme to scale up to large networks, where hundreds of thousands of VoIP are monitored, a fast algorithm is required to compute robust hash values in real-time. In this section we present an efficient algorithm, which computes the robust hash of a VoIP flow cumulatively as the packets are captured requiring minimal memory.
The proposed algorithm assumes that the smooth pseudorandom bases are previously computed and stored in the memory. Note that, in practice these pseudorandom basis arrays should be longer than bitrate variation signals of all possible VoIP flows. Therefore, length of the input signals should be bounded by a certain value. For instance, in our experiments we mostly use 200 consecutive packets of VoIP flows, which approximately corresponds to 12 seconds. Hence, in the experiments we safely employed 15-second long pseudorandom bases.

Once the pseudorandom bases are computed, the proposed algorithm updates the projection values each time a packet is captured. The intuition is based on the fact that the bitrate variation signal 'V(n)' has nonzero entries only when n = i. Therefore, we can rewrite Equation (4.13) as:

\[ H_l = \sum_{i=1}^{P-1} V(\hat{T}_i) R_l(\hat{T}_i) \] (4.15)

combining this with Equation (4.12), we get:

\[ H_l = \sum_{i=1}^{P-1} B^\Delta_i R_l(\hat{T}_i) \] (4.16)

Using the above equation one can compute the projection values cumulatively as the packets arrived. A simple algorithm, named Compute_Hash, to compute the robust hash in real-time using this equation is given below.

**Algorithm 4.2**  
\( h = \text{Compute}_H\text{ash} \)  

\[ H \leftarrow [0, 0, 0, ..., 0] \{/\text{Initialize projection values } H_1, H_2, ..., H_L\} \]

for all captured packet \( i \) such that \( i = 0, 1, ..., P - 1 \) do  
if \( i = 0 \) then  
flowStart \leftarrow T_i \{/\text{first packet timestamp is when the flow starts}\}  
else  
\( \hat{T}_i \leftarrow T_i - \text{flowStart} \{/\text{relative timestamp}\} \)  
\( B^\Delta_i = B_i - B_{i-1} \{/\text{difference in the bitrate}\} \)  
\( H \leftarrow H + B^\Delta_i \left[ R_1(\hat{T}_i), R_2(\hat{T}_i), ..., R_L(\hat{T}_i) \right] \{/\text{update projections}\} \)  
end if  
end for  
\( h = \text{sign}(H) \)  
Output \( h \)

The Compute_Hash algorithm is very efficient and runs in \( O(N) \) time where \( N \) is the
number of packets. On the other hand, it requires a constant memory space as it has to keep only the pseudorandom bases and the flowStart.

### 4.7.3 Experiments

To evaluate the efficacy of the proposed scheme, we set up an experiment where we evaluated the detection performance under various network impairments. For this purpose, we first played 2 hours of recorded speech over a single Skype call and captured its UDP packets. We employed Skype because it is one of the most widely used VoIP application. Then, to obtain a VoIP flow segment, we picked 200 consecutive packets starting from a randomly selected packet. Using this strategy we obtained 10,000 VoIP flow segments. After that, we applied packet delays, jitter and packet drops to the original segments to simulate the network impairments and consequently obtained 10,000 modified segments. More specifically, we delayed every packet by $D_l$ milliseconds. Then we added jitter $J_t$ on each packet’s timestamp where $J_t$ was drawn from a zero-mean Laplacian distribution with standard deviation $\sigma_{J_t}$. Finally, we randomly dropped packets with probability $P_d$ and obtained the modified VoIP segments. After that, we computed the Hamming distance between the robust hashes of the original segments and the modified segments. If the Hamming distance is below a threshold, then we consider it as a successful detection. On the other hand, we also compared the robust hash of each VoIP segment with another randomly selected segment. In this case, if the Hamming distance is above a threshold, then we consider it as a false alarm.

We present the resulting ROC curves in Figures 4.11 and 4.12. We initially fix the parameters to $D_l = 2000$ ms, $\sigma_{J_t} = 5$, $P_d = 0.01$, hash length $L = 256$, input length = 200 packets. To demonstrate the algorithms sensitivity to each parameter, we varied only one parameter on each of the Figures 4.11(a), 4.11(b), 4.12(a) and 4.12(b). We didn’t vary delay since the resulting robust hashes are invariant under delay because the hash function uses relative packet timestamps with respect to the start of each flow. We observe in these figures that, the proposed scheme performs satisfactorily under reasonable jitter and packet drop rates. Also, it is observed in Figure 4.12(a) that increasing hash length improves the performance up to some point and then saturates. Hence $L = 256$ seems to be appropriate since the longer the hash values are the harder to manage them (storage, computation etc.) Finally, Figure 4.12(a)
suggests that longer input signals are easier to detect. This is expected since longer signals have more content information which distinguishes it from others.

4.8 Conclusion

In this chapter, we presented an efficient correlated flow detection scheme, which can be used in online stepping-stone detection applications. The proposed scheme continuously maintains sketches of network flows’ packet-timing information from a stream of captured packets at the border of a network. These sketches are then used to efficiently identify correlated flows, since the correlated flows have similar sketches. The proposed scheme computes flow sketches very efficiently by a streaming algorithm, which performs a few arithmetic operations for each packet. In addition, the sketches of a pair of correlated flows remain similar, even if the flows encounter various timing perturbations, thereby allowing the proposed scheme detect the correlated flows even under delays, jitter, chaff, etc to some extent. Finally, using the fact that correlated flows have similar sketches, the proposed scheme identifies correlated ingress/egress flow pairs among \( n \) ingress and \( m \) egress flows in \( O(n + \sqrt{nm}) \) time, as compared to known techniques, which requires \( O(nm) \).

We also presented a variant of the proposed sketching scheme for VoIP flows, which could...
be used for VoIP call tracking applications. The proposed scheme exploits a VoIP flow’s packet timings and bitrate variations to identify the audio contents of that flow with a short binary string. Our experiments show that the proposed scheme can successfully identify VoIP calls under various network impairments, such as delay, jitter and packet drops. We also showed that the proposed hash values can be efficiently computed in real-time, thereby allowing us to potentially employ it in large scale VoIP tracking applications.

There is always room for improvement. We observed in the experiments that, when we use longer time-slots in order to be more resistant to packet delays, the sketches of uncorrelated flows start to exhibit some similarities, thereby negatively affecting the detection performance. Shorter time-slots, on the other hand, have limited resistance to packet delays. To combine the advantages of both sides and potentially improve the resistance to packet delays, we plan to use randomly varying time-slot lengths. Meanwhile, we used a fixed threshold when deciding if two flows are correlated or not. However, using a different threshold for each pair of flows, regarding several features, such as flow durations, number of packets etc, will potentially improve the detection performance. We leave the exploration of these improvements as future work.

Figure 4.12: ROC curves for different sketch and signal lengths. The parameters are $Dl = 2000$ ms, $\sigma_{Jt} = 5$, $Pd = 0.01$, hash length $L = 256$, input length = 200 packets.
5.1 Introduction

Botnets, which are networks of compromised hosts (bots) under the control of a botmaster, have become a major threat for today’s networks. Botmasters use botnets to perform various malicious activities such as spamming, phishing, stealing sensitive information, conducting distributed denial of service (DDoS) attacks, scanning to find more hosts to compromise, etc. Bots, which perform such malicious activities, often go over the radar and get detected by Intrusion/Anomaly Detection Systems. In fact, network administrators regularly discover bots, which expose themselves by sending spam emails or performing a scan, etc. If such a bot is discovered in a network, typically it is immediately quarantined or removed. However, some interesting and important questions remain, such as: “Does the network contain more bots which haven’t been exposed yet?” “Can the discovered bot be leveraged to find dormant bots in the network before they commit any malicious activity?”

In this chapter we propose an efficient technique to address these questions. After discovering a P2P bot in a network, the proposed technique provides a list of hosts in the same network which potentially belong to the same P2P botnet as the discovered bot. Network administrators can use this list as a starting point for their investigations and potentially identify more bots in their network once they discover one.

The proposed technique is based on an analysis of the recent history of the network’s traffic. The idea is that, since along with the discovered bot, the dormant bots probably have also been
receiving command and control (C&C) messages and updates from the botmaster via some form of a C&C channel. Therefore, if we could characterize the C&C channel from the discovered bot’s recent traffic, then we would be able to link the discovered bot to the hosts that exhibit similar C&C traffic characteristics and label them as potential bots. However, characterizing the C&C channel is not a trivial task. First of all, a C&C channel cannot be characterized by its payload as most of the recent botnets such as Nugache, Storm, Waledac and Conficker employ advanced encryption mechanisms [48,54,73,82,88]. Similarly, several features based on packet sizes and timings, such as packets per flow, bytes per flow, flows per hour, etc. may not be useful in characterizing a C&C channel, since botmasters can easily randomize them thereby obtaining different feature values for each bot [86].

For botnets with centralized C&C architecture, where all bots receive commands from a few central control servers, the source of the C&C messages can be used to characterize the corresponding C&C channel. Therefore, one can reveal potential bots in the network to some extent [62], by identifying hosts which have been in communication with the same control servers as the discovered bot. Nevertheless, this kind of source analysis fall short for botnets which utilize peer-to-peer (P2P) architecture without any central server.

In a P2P botnet, each bot acts both as a server and a client. The botmaster can use any node to inject C&C messages. To receive and distribute these C&C messages, each P2P bot communicates with a small subset of the botnet (i.e. peer list) [48,54,88]. Typically each bot of a P2P botnet maintains its own peer list independently. Hence, no obvious common source of C&C messages can be observed among the P2P bots in a network, thereby preventing us from linking the discovered bot with the dormant bots.

However, despite the large size of a P2P botnet, in this work we show for P2P botnets with an unstructured topology, where bots randomly pick peers to communicate with, that there is a surprisingly high probability of any given pair of P2P bots in a network communicating with at least one common external bot during some time window. This can be explained by principles similar to the Birthday Paradox and we refer to external hosts which communicate with multiple internal hosts during a time window as mutual-contacts. Hence, one can potentially use these mutual-contacts to link the discovered bot, which we refer to as the seed-bot, to at least a few of the dormant bots.
In the proposed technique, we exploit this conversational pattern occur due to mutual-contacts. To identify several dormant bots, we present an iterative algorithm, where we start with extracting mutual-contacts between each pair of hosts in the network from the flow records captured at the network border during a window of time prior to the discovery of the seed-bot. In order to eliminate obvious mutual-contacts, such as google.com or microsoft.com, we only consider the mutual-contacts which have communicated with only less than a few internal hosts, since external P2P bots are expected to communicate with only few internal hosts, which are also P2P bots themselves. The basic intuition of the proposed method is that, if a host shares mutual-contacts with a P2P bot, then it is likely to be a member of the same botnet too. Once we obtain the mutual-contacts, we iteratively identify a few other potential P2P bots from the seed-bot and then from the newly-found potential P2P bots we identify other potential P2P bots and so on. During this process, we assign a confidence level to each identified potential P2P bot based on with whom and how many mutual-contacts it shares. More specifically, the more the number of mutual-contacts a candidate host has with the seed-bot and/or with other candidate hosts having high confidence levels, the higher our confidence that this candidate host is indeed part of the botnet. Once we compute the confidence levels, we declare the hosts, which have confidence levels higher than a threshold, as potential members of the same P2P botnet as the seed-bot. The contributions of the proposed scheme can be summarized as:

- The proposed method is not an anomaly detection scheme and hence doesn’t require P2P bots to exhibit any overtly malicious activity.

- Similarly, it is not a behavior clustering algorithm and therefore doesn’t require any common behavior exhibited by all other P2P bots.

- It utilizes the pairwise mutual-contact relationships between pairs of bot peers, which arise due to P2P C&C communications. We validate the existence of such relationships both mathematically and experimentally.

- The proposed method is generic and doesn’t depend on specific properties of specific botnets. Therefore, it doesn’t require reverse engineering bot binaries or C&C protocols [16].
To evade the proposed scheme, a botmaster has to make sure that the peer lists of the bots within a same network are as distinct as possible, thereby minimizing the probability of creating mutual-contacts. This may be achieved by imposing a structure on the topology of a P2P botnet. Several structured P2P architectures have been proposed [17, 58, 77, 87]. Such architectures, however, are basically Distributed Hash Table (DHT) implementations, where the main purpose is to efficiently locate the corresponding data item in the P2P network for a given key (index). Therefore these architectures don’t provide any explicit mechanism to reduce the probability of mutual-contacts among peers within a same network. Furthermore, in order to be robust against node failures and node removal attacks, even structured P2P architectures have to include some level of randomness in their peer selection mechanisms [77], which may result in several new mutual-contacts among peers.

To eliminate mutual contacts, peers in a same network have to coordinate with each other so that they won’t communicate with the peers in each other’s peer list. In some sense, peers in a same network have to form their own tiny botnet among themselves and appear as a single node to the remaining of the P2P botnet. These intra-network communications among the peers in a same network, however, would potentially yield new means of detecting P2P bots in a network. Nevertheless, even if a botmaster manages to deploy a mutual-contact-free P2P architecture, then two or more networks can easily share their flow records, possibly in a privacy preserving manner [29], to detect P2P bots in their networks by utilizing the mutual-contacts among P2P bots in different networks. This strategy effectively merges the network traffic of collaborating networks and virtually form a larger network, which inevitably contains mutual-contacts since the botmaster cannot know which networks would collaborate in the first place. In summary, although it may be possible for a botmaster to design a structured P2P topology which is free of mutual-contacts and robust to node removal, designing such a topology is not trivial and beyond the scope of this work. In this work, we focus on unstructured P2P botnets which are widely observed in today’s networks [32, 54, 73, 82]. In this chapter, the term P2P botnet refers to unstructured P2P botnet unless stated otherwise.

The rest of the chapter is organized as follows: We present the related work in Section 5.2. In Section 5.3, we explain the basic idea and details of the proposed method. Following that, we present our experimental results with the Nugache botnet in Section 5.4. In Section 5.5,
we mathematically argue why the proposed scheme works. Then in Section 5.6, we discuss limitations of the proposed scheme, possible evasion techniques and their implications on P2P botnets. Finally, we conclude the chapter and propose the planned future work in Section 5.7.

5.2 Related Work

Early botnets employed centralized command and control (C&C) servers to distribute commands and updates to individual bots, usually through IRC or HTTP protocols [31]. Although a centralized structure is simple and easy to manage, it suffers from a single point of failure and is susceptible to traditional defenses such as domain revocation, DNS redirection, blacklisting etc. Therefore, botmasters have begun to adopt a P2P architecture for C&C channels. In [57], authors argue that it is harder to detect P2P bots especially with a limited view of their traffic from a single Autonomous System. In P2P botnets each bot acts both as a server and a client allowing botmasters to publish commands and updates from any point in the botnet [48] [54]. In [28], authors investigate the effects of different botnet structures.

There have been numerous techniques proposed to detect botnets. In [67] and [64], the authors employ machine learning techniques where they train classifiers with different features to detect botnet C&C flows. In [89], Strayer et. al. proposed a technique to detect botnet activity by exploiting temporal correlations between C&C activities of the bots belonging to the same botnet. Binkey and Singh proposed a technique to detect IRC botnets in [14] using botnet-related anomalies in TCP and IRC statistics. Another IRC botnet detection scheme was proposed by Goebel and Holz in [47], where the authors exploited the structure and evolution of IRC nicknames used by IRC bots. In [62], Karasaridis et. al. combined traffic aggregates with IDS alarms to identify centralized botnets within a Tier-1 ISP. In [50], Gu et. al. proposed BotHunter, which searches for a specific pattern of events in IDS logs to detect successful infections caused by centralized botnets.

All the above schemes were designed to detect either specific botnets that they were trained for, or centralized botnets. In general, detecting P2P bots in a network is harder since there is no trivial correlation that allows us to link together the P2P bots in a network, especially when bot peers communicate with each other using encryption and through random ports [33,48,54]. As
a completely different problem from ours, crawler based methods were proposed to enumerate P2P bots globally in [60] and [54]. Since crawlers cannot enumerate P2P bots behind NAT and/or firewall in [59] Kang et. al. proposed a sybil attack based passive monitoring scheme to enumerate P2P bots even behind NAT or firewall. However, P2P bot enumeration methods are not intended to identify local P2P bots in a network. Also, they require implantation of bot peers which requires reverse engineering of a bot binary and its C&C protocol. Coming back to our problem, there have been few techniques proposed which are able to detect local P2P bots assuming that P2P bots exhibit similar malicious activities and similar connection patterns. In [51], Gu et. al. proposed BotSniffer to detect bots based on spatial-temporal correlation between bot responses to commands. Following that, in [49], Gu et. al. proposed BotMiner which clusters the hosts in a network by their malicious activity and communication patterns. Their results showed that members of a botnet usually fall within the same cluster. Similarly, in [107], Yen and Reiter proposed a scheme called TAMD, where traffic containing similar external IPs, similar payloads and similar internal platform types are aggregated to detect botnets in a network. Although clustering based botnet detection schemes are successful in detecting many current P2P bots, botmasters can evade them by assigning different tasks to the bots in the same network or by randomizing their communication patterns as acknowledged in [49]. In [86], authors systematically investigate such evasion techniques. Also, clustering based schemes fall short in detecting idle P2P bots which haven’t exhibited any overt behavior yet.

5.3 Finding Friends of An Enemy

5.3.1 Basic Idea

Consider the botnet illustrated in Figure 5.1(a). The basic idea of the proposed method is that, Host A can be linked to Host B since they both communicate with Host X. Similarly Host B and Host C are linked together through Host Y and Host Z. As a result, if Host A is known to be a member of a P2P botnet, one can conclude that Host B is likely to be a member too. Similarly, if Host B is likely to be a member, then Host C is also likely to be a member.

It is clear that, aside from P2P botnet traffic, legitimate traffic probably includes several
Figure 5.1: Illustration of P2P Botnet communications for a network (a) and its corresponding mutual-contact graph (b). The network contains 2 benign hosts and 3 bots (Hosts A, B, and C). The bots are members of a P2P botnet with 9 bots in total. Mutual-contact relationship among hosts, which is indicated by red dashed arrows in (a), are represented by the mutual-contact graph in (b). The edge capacities are determined by the number of mutual contacts between nodes.

mutual-contacts among hosts as well. For instance, there are some very popular servers that almost every host in the network communicates with such as google.com, microsoft.com, etc. As a result, every host in the network is linked with most of the other hosts through these popular mutual-contacts. To prevent this, only private mutual-contacts, which communicate with less than $k$ internal hosts during an observation window, are considered in the proposed method. Here, $k$ is the privacy-threshold. It is highly unlikely that external peers that are part of the botnet will be communicating with many internal hosts that do not belong to the botnet. Therefore, private mutual-contacts can be strong indicators of peer relationships among hosts.

In the rest of this chapter, we use the term mutual-contacts to mean private mutual contacts.

5.3.2 The Proposed Scheme

In the proposed scheme, we first extract mutual-contacts from the flow records captured at the network border for a time window prior to discovering the seed-bot. We then represent the mutual-contact relationships among hosts by a directed graph called the mutual-contacts graph, such that:

- Nodes represent the hosts in the network.
• There is a bidirectional edge between two nodes if the corresponding hosts have at least one mutual-contact during the given time window.

• Each edge has a capacity determined by the number of mutual-contacts between corresponding nodes.

• As an example, the mutual-contact graph for the network illustrated in Figure 5.1(a) is shown in Figure 5.1(b).

It is expected that, hosts which are likely to be P2P bots are at a short distance from the seed-bot on a mutual-contacts graph since such hosts are expected to have mutual-contacts with the seed-bot itself and/or with other hosts which have mutual-contacts with the seed-bot. The mutual contacts graph illustrated in Figure 5.2(a) displays such behavior (black edges). Relying on this, the proposed scheme essentially performs a graph traversal starting from the seed-bot and computes a confidence level of being a P2P bot for each visited node. This traversal process can be illustrated as pumping red dye into the mutual-contacts graph from the node representing the seed-bot as depicted in Figure 5.2(b). During the process, the dye coming to a node is distributed across its outgoing edges proportional to their capacities. Therefore, the dye accumulated in a node reflects our confidence of that host being a part of the same botnet as the seed-bot. That is because, the more mutual-contacts that a host shares with a P2P bot (i.e. connected with a higher capacity edge), the more likely that host is also a P2P bot. As a result, after pumping dye for a while, we declare the nodes which accumulates more dye than a threshold as peers of the same P2P botnet as the seed-bot. In Figure 5.2(b), it is also observed that along with the P2P bots, few benign hosts share mutual-contacts with P2P bots (connected via green edges in Figure 2(a)), and therefore receive some amount of dye. Such hosts potentially result in false positives. However, the capacity of the edges connecting these benign hosts to P2P bots are usually low thereby keeping the dye accumulated on these benign hosts below the threshold in most cases. In the following subsections, we present each step of this algorithm in greater detail.
Figure 5.2: (a) Illustration of a mutual-contacts graph. P2P bots tend to share mutual-contacts with each other (black edges). Also some benign hosts share mutual-contacts among themselves (blue edges), which may be due to a legitimate P2P application. (b) Illustration of the dye-flow in the Dye-Pumping algorithm.

5.3.3 The “Mutual-Contacts” Graph

We denote the mutual-contacts graph constructed from the flow records of a network by $G = (N, E)$, where the nodes correspond to the hosts and the edges indicate private mutual-contacts. Each edge on the graph has a capacity which is determined by the exact number of mutual-contacts between corresponding hosts. More formally, if $E_{ij}$ represents the capacity of the edge between nodes $N_i$ and $N_j$, then we can write:

$$E_{ij} = E_{ji} = |S(N_i) \cap S(N_j)|$$

where $S(N_i)$ represents the set of mutual-contacts which $N_i$ was in communication with during the observation period and $|\cdot|$ represents the cardinality of a set. Notice that, if nodes $N_i$ and $N_j$ don’t share any mutual-contacts then there is no edge between them on the graph or equivalently $E_{ij} = 0$. 
5.3.4 The “Dye-Pumping” Algorithm

Once the mutual-contacts graph is constructed, the dye-pumping algorithm is executed to compute the confidence levels of hosts being a part of the P2P botnet. The dye-pumping algorithm iteratively pumps dye to the mutual-contacts graph from the seed node and picks the nodes which accumulates more dye than a threshold. During the process, dye coming to a node is distributed to other nodes proportional to a heuristic called the dye-attraction coefficient, which helps the algorithm to funnel more dye towards the nodes which are more likely to be P2P bots.

**The Dye-Attraction Coefficient** is denoted by $\gamma_{ji}$, and indicates what portion of the dye arriving at Node $j$ will be distributed to Node $i$ in the next iteration. Intuitively, it represents our confidence on Node $i$ being a P2P bot given that Node $j$ is a P2P bot. Such confidence gets higher as Node $i$ and Node $j$ share more private mutual-contacts with each other. On the other hand, our confidence reduces if Node $i$ shares mutual-contacts with many other nodes in the graph. The reason is that, we expect to have few bots in our network and therefore if a host shares mutual-contacts with many other hosts, then these mutual-contacts are probably due to a different application other than botnet C&C. Consequently, we compute the dye-attraction coefficient from Node $j$ to Node $i$ as follows:

$$
\gamma_{ji} = \frac{E_{ji}}{(D_i)^\beta}
$$

where $D_i$ is the degree of Node $N_i$ (i.e. number of neighbors or edges that $N_i$ has) and $\beta$ is the **Node Degree Sensitivity Coefficient**. Basically, nodes with high degrees receive less and less dye as $\beta$ increases.

**The Dye-pumping Algorithm** has three inputs, namely the edge capacities ($E_{ji}$) of the mutual-contacts graph ($E$ represents the matrix containing all $E_{ji}$ values), the index ($s$) of the seed node $N_s$, and the number of iterations ($maxIter$). Given these inputs, the dye-pumping algorithm first computes the dye-attraction coefficients from edge capacities and forms the transition matrix $T$ such that:
\[ T(i, j) = \gamma_{ji} = \frac{E_{ji}}{(D_i)^j} \]

where \( i = 1, \ldots, v \) and \( j = 1, \ldots, v \). Also \( T(i, j) = 0 \) if \( i = j \). Notice that the transition matrix of a mutual-contacts graph with \( v \) nodes is a \( v \times v \) square matrix.

Following that, the algorithm normalizes \( T \), so that each of its columns sums to 1 (i.e. stochastic matrix). If \( \mathbf{T} \) indicates the normalized transition matrix, the normalization procedure can be written as \( \mathbf{T}(i, j) = \frac{T(i, j)}{\sum_{i=1}^{v} T(i, j)} \). After normalization, the algorithm iteratively pumps dye to the mutual-contacts graph from the seed node. For this purpose, let the column vector \( \mathbf{L} \) is the dye level vector, where \( L(i) \) indicates the the dye level accumulated at node \( i \). The pumping begins with filling the seed node with dye and leaving the others empty such that:

\[
L(i) = \begin{cases} 
1, & \text{if } s = i \\
0, & \text{elsewhere}
\end{cases}
\]

Once the seed node is filled with dye, the algorithm pumps the dye from the seed node across the mutual-contacts graph. Since the outgoing edges distribute the dye accumulated within a node proportional to their capacities, the dye levels at next iteration can be computed as:

\[
L(i) = \sum_{j=1}^{v} T(j, i)L(j)
\]

which can be also written in matrix form as \( L = \mathbf{TL} \). At each iteration, after updating \( L \), the algorithm pumps more dye to the graph from the seed node by updating \( L(s) = L(s) + 1 \). Following that the vector \( L \) is normalized after each iteration as \( L = \frac{L}{\sum_{i=1}^{v} L(i)} \). Finally after \( maxIter \) iterations, the dye-pumping algorithm outputs the dye-level vector \( L \). The steps of the dye-pumping algorithm are summarized below:

Once the algorithm outputs the vector \( L \), the dye level of each node \( (L(i)) \) indicates the confidence level for the corresponding host being a member of the same P2P botnet as the seed node. To have a more conclusive result, we set a threshold \( thr \) such that the nodes having a dye level greater than \( thr \) are declared as potential members of the same P2P botnet as the seed node.
Algorithm 5.1 \(Dye\_Pumping(E, s, maxIter)\)

1: \(T \leftarrow computeTransitionMatrix(E)\)
2: \(T \leftarrow normalize(T)\)
3: \(L \leftarrow [0, 0, ..., 0]^T\) \{initialize \(L\) as a zero vector\}
4: for \(iter = 1\) to \(maxIter\) do
5: \(L(s) \leftarrow L(s) + 1\) \{Pump dye from the seed node\}
6: \(L \leftarrow \frac{L}{\sum_{L[i]}L[i]}\) \{Normalize fluid level vector\}
7: \(L \leftarrow TL\) \{Distribute dye in network for one iteration\}
8: end for
9: output \(L\)

5.4 Experiments

5.4.1 Detecting Nugache Peers

In order to systematically assess the performance of the proposed scheme against a real-world botnet, one needs to know the IP addresses of the members of a P2P botnet in a given network. Otherwise, nothing can be said about the true positive or false alarm rate without knowing the ground truth. One way to obtain the ground truth is to blend real botnet data into the network traffic and make few hosts look as if they have been infected by the botnet. This strategy essentially aggregates real botnet traffic and real user traffic on some of the hosts and therefore provides a realistic scenario. From the proposed scheme’s perspective, to make a host look like a P2P bot, one can first capture the flow records of the network, which contains the host, during a time window. Then one can collect the flow records form a real P2P bot during a similar time window. Following that, one can change the bot’s IP address in these botnet flow records to a selected host’s IP address and append them to the flow records of the entire network so that, along with its original traffic, the selected host will appear as if it has also communicated with the external IP addresses that the real bot has talked to.

In order to establish the ground truth for our experiments, we utilize the data collected from the Nugache botnet, which has been thoroughly studied in [88] [32]. Briefly speaking, Nugache is a P2P botnet that uses random high-numbered ports for its communication over TCP. The data we use in our experiments was compiled by the Nugache crawler presented in [33] and its communication between Nugache peers.
Figure 5.3: Properties of the mutual-contacts graph of the background traffic for different privacy threshold (k) values.

**Nugache Botnet Data:** The details of the Nugache botnet and Nugache crawler can be found in [88] and [32]. In summary, the C&C protocol of Nugache enables querying a peer for its list of known peers and a list of recently communicated peers. Using these functionalities, the crawler starts from a series of seed peers and traverses the botnet by querying peers for their list of known peers. While crawling, the crawler also maintains the list of recently communicated peers for each accessible Nugache peer. Consequently, when the crawler finishes crawling, it produces list of recently communicated peers for several Nugache peers.

In our experiments, we use the data collected by the crawler a few years ago when Nugache was active. To collect data, the crawler was executed repeatedly for 9 days, where each execution lasted roughly 30 to 45 minutes. We use 24-hour observation window for our experiments. Hence, we employ several randomly selected 24-hour segments of the crawler data from 9-day results in our experiments to cover the botnet dynamics during all 9 days. We observe in any of these 24-hour segments that on average 904 Nugache peers responded to the crawler. We also observe that on average 34% of all possible pairs of Nugache peers in the data set communicated with at least one mutual-contact.

**Background Traffic:** In order to obtain a background traffic that we can blend Nugache traffic into, we captured the flow records observed at the border of our university’s network during a typical weekday (i.e. the observation window is 24 hours). Collected flow records indicate that there were 2128 active IP addresses in our network during the observation window.
We then extract mutual-contacts from the recorded data. To ensure a valid communication (i.e. not a scan flow), we only consider external IPs which exchange at least 256 bytes in both directions with at least one internal IP. Finally we build the corresponding mutual-contacts graph to serve as a basis for our experiments. We immediately observe in the mutual-contacts graph that the DNS servers in our network share significantly large number of mutual-contacts between each other. As a matter of fact, the DNS servers constitutes the highest-magnitude entries of the first eigenvector of the matrix \((E)\) whose entries are the corresponding edge capacities \((E_{ij})\). This is not surprising since DNS servers in a network communicates with many other DNS servers around the world. Obviously this relationship among DNS servers dominates the mutual-contacts graph and taints the results of Dye-Pumping algorithm. Hence, we remove all the edges of 11 DNS servers in our network from the mutual-contacts graph.

The mutual-contacts graph extracted from the background traffic suggests that majority of the hosts share no or very few mutual-contacts with other nodes. This can be observed in Figure 5.3(a), where we plot the distribution of node degrees (i.e. no of edge of a node). Figure 5.3(a) also shows that the nodes usually have higher degrees for higher privacy threshold \((k)\) values as expected. To investigate the nodes which share mutual-contacts with each other, we look at the clustering coefficient, which is defined as the ratio of the number of the actual edges of a node to the number of all possible edges among it’s neighbors. We plot clustering coefficient distribution of the nodes in Figure 5.3(b). We observe that the mutual contact-graph is a lot more clustered than a comparable random graph (i.e same number of nodes and edges). For instance the clustering coefficient distribution of a random graph comparable to the mutual-contacts graph with \(k = 5\) has a mean of 0.006 and standard deviation of 0.009. This suggests that there are communities of hosts in our network where community members usually communicates with same external IPs exclusive to the corresponding community. One can speculate that these communities may represent peers of different P2P networks (legitimate or bot) or a group of users visiting a same list of rare websites etc.

**Experiments with Nugache:** In order to assess the performance of the proposed scheme in detecting Nugache bots, we randomly pick \(m\) Nugache peers from a randomly selected 24-hour segment of the crawler data. Then, we compute the mutual-contacts graph corresponding
Figure 5.4: Experiment results with Nugache. The parameters are $k = 5$ and $\beta = 2$

...to these $m$ Nugache peers based the recently-communicated peers field of the crawler data.

Then we randomly pick $m$ internal hosts from the mutual-contacts graph corresponding to the background traffic. Finally, we superpose the mutual-contacts graph of the Nugache peers onto in the mutual-contacts graph of the background traffic where $m$ Nugache peers coincide with $m$ selected internal hosts. This procedure essentially blends the Nugache traffic into the background traffic so that each of these $m$ selected internal hosts looked like as if they communicated with the peers that the corresponding $m$ Nugache peer recently communicated with. Consequently, each of these $m$ selected hosts becomes a real Nugache peer and constitute the ground truth as far as the proposed scheme is concerned. Once we obtain the superposed mutual-contacts graph, we randomly select one of the $m$ hosts as the seed bot and run Dye-Pumping algorithm to detect other $m - 1$ hosts whose flow records were modified according to the Nugache crawler data. We set the number of iterations to $maxIter = 5$ for Dye-Pumping algorithm since it is almost impossible to find P2P botnet peers more than 3 hops away from the seed node due to the Erdős-Rényi model as will be explained in Section 5.5. In the end, we return the list of hosts which accumulate more dye than the threshold as P2P bots. To obtain statistically reliable results, we repeat the experiment 100 times, each time with different selection of $m$ hosts and $m$ Nugache peers. We also pick a different 24-hour segment of crawler data at every 20th repetition.
Results (Precision & Recall): To gauge the algorithm’s performance, we compute the average precision and recall. In our context, precision can be defined as the ratio of the number of Nugache peers in the returned list of hosts to the length of the returned list. On the other hand, recall can be defined as the ratio of the number of Nugache peers in the returned list to the number of all Nugache peers in the network except the seed bot \((m - 1)\). Figure 5.4 presents the average precision and recall values for different number of Nugache peers \((m)\) and different threshold values \((\text{thr})\). We set the privacy threshold \(k = 5\) and node degree sensitivity coefficient \(\beta = 2\). It is observed that several dormant Nugache peers can be identified by the proposed technique when the threshold is set to an appropriate value. For instance, in Figure 5.5 we observe that, if there are 17 Nugache peers in the network, the proposed scheme on average returns 35 hosts, 11 of which are Nugache peers. As a result, upon obtaining the list of potential P2P bots, a network administrator can perform a more detailed investigation (perhaps physically) on the hosts in the list and potentially uncover several dormant P2P bots. Meanwhile, the returned list also contains some hosts which are not Nugache peers since such hosts happen to be connected to one or more Nugache bots on the mutual-contacts graph due to mutual-contacts created by other applications. Interestingly, it is observed in Figure 5.4 that both precision and recall values increase as the number of bots \((m)\) increases. This is due to a property of Erdős-Rényi random graphs that—as will be explained in the next section—the probability of having a short path between two nodes increases with the number of nodes. It is also observed that, increasing the threshold increases precision but decreases recall, as naturally expected for any detection system.

Effects of Privacy Threshold \((k)\): When we repeat the experiments for different \(k\) values, we don’t observe a major change in the precision performance. On the other hand, we observe, as shown Figure 5.6(a), that recall performance improves as we decrease \(k\) so long as the number of P2P bots in the network is low. The recall performance improves because more background traffic is filtered out for lower \(k\), thereby removing significant portion of the benign edges. However, if there are many P2P bots in the network and if \(k\) is small (i.e. \(k = 3\)), more than \(k\) of them are likely to communicate with several common external peers and therefore some of the botnet communications are likely to be filtered out as well. The effect of this phenomenon can
Figure 5.5: Returned host counts for \( thr = 5 \times 10^{-4} \). Darker parts of bars represent average number of bots returned by the algorithm in one execution, whereas lighter parts represents average number of hosts which are not bots.

be observed in Figure 5.6(a), where recall performance diminishes for large number of Nugache peers. Hence, based on Figure 5.6(a) we can say that \( k = 5 \) is an appropriate setting for our network.

**Effects Node Degree Sensitivity Coefficient (\( \beta \)):** As explained in Section 5.3.4, larger \( \beta \) values result in less dye-flow towards the nodes which have high degrees on a mutual-contacts graph. We would like to restrict the dye-flow to high-degree nodes, because edges between bots and high-degree nodes are probably not due to botnet communications but rather due to some other application which causes all the many other edges that high-degree nodes have. Larger \( \beta \) values cause the dye concentrated around the seed-bot and therefore improve the precision performance as observed in Figure 5.6(b). On the other hand, since the algorithm cannot reach far on the mutual-contacts graph for larger \( \beta \) values, the recall performance drops as \( \beta \) gets larger as observed in Figure 5.6(c). According to our experiments, \( \beta = 2 \) is an appropriate setting for our network.

### 5.5 Why “Dye-Pumping” Works: A Mathematical Analysis

The essence of the dye-pumping algorithm is that the members of a P2P botnet tend to have mutual-contacts between each other and therefore are closely connected on a corresponding
private mutual-contacts graph. In fact, the dye-pumping algorithm performs better if P2P bots in a network are connected to the seed node through shorter and higher-capacity paths, which yield higher volume of dye flow from the seed node to other bots. In this section, we investigate whether P2P bots in a network exhibit such properties.

5.5.1 Random Peer Selection Model

In a P2P network some peers might be more available than others and therefore they have a higher probability of being selected by other peers [48] [54] [59] [13]. Obviously, having such preferred peers in a P2P botnet increase the chance finding mutual-contacts between P2P bots in a network. However, the worst case, as long as unstructured P2P botnets are considered, from our work’s point of view is when there is no preferred peer in the botnet and all peers have equal probability of being contacted by any other peer, thereby minimizing the probability of private mutual-contacts between peers.

To investigate the probability of mutual-contacts in the worst case, we consider a generic botnet model, where each bot picks peers independently and randomly. The model has two configurable parameters such that; “$B$” is the number of all peer in the botnet and “$C$” is the number of peers that each peer communicate with during a specific observation window. Based on these parameters, each bot ($b_i$) in the model communicates with a uniform random subset ($S_i$) of all $B - 1$ available bots (excluding itself) in the model, where the cardinality of each subset is $C$. 
**Bot-Edge Probability:** In the random peer selection model, the probability of having an edge between two arbitrary bots $b_i$ and $b_j$ (i.e. bot-edge probability, $p_e$) is actually the probability of the intersection of the corresponding subsets being non-empty; such that $p_e = Pr(S_i \cap S_j \neq \emptyset)$. Since the number of elements in the intersection of two uniform random subsets can be computed using hypergeometric distribution, we can write the bot-edge probability as:

$$p_e = 1 - \frac{{C \choose 0}{B-1-C \choose C}}{{C \choose 0}{B-1 \choose C}}$$

(5.1)

Bot-edge probabilities are plotted in Figure 5.7(a). It is observed that, similar to the Birthday Paradox, as the number of contacted peers increases, the bot-edge probability increases very rapidly. Consequently, even for a very large botnet with 500k peers, the bot-edge probability is almost 1 when peers contact only 1000 other peers during the observation window.

**Bot-Edge Capacity:** Although, high bot-edge probabilities works in favor of the dye-pumping algorithm, the capacities of those edges are also important. It is obvious that, the higher the bot-edge capacities the better the dye-pumping algorithm performs. In the random peer selection model, the probability of a peer contacted by two given peers is $\left(\frac{C}{B}\right)^2$. Therefore, since there are $B$ peers in total, we can write the expected capacity of bot edges ($E[Cp]$) as:

$$E[Cp] = \left(\frac{C}{B}\right)^2 B = \frac{C^2}{B}$$

(5.2)

which is also the expected value of the corresponding hypergeometric distribution. Figure 5.7(b) plots the expected bot-edge probabilities. It is observed that, regardless of the botnet size, expected bot-edge capacity rapidly exceeds 1 and continues to increase as the number of contacted peers increases. Figure 5.7 suggests that the members of a P2P botnet will most probably be well connected with each other on a private mutual-contacts graph through high capacity edges, thereby allowing the dye-pumping algorithm to identify them.

### 5.5.2 Friends Stay Closely Connected (Erdős-Rényi Subgraphs)

The Dye-Pumping algorithm can only identify the P2P bots which are connected to the seed-bot via short paths on the mutual-contacts graph. Bots which are isolated from the seed-
Figure 5.7: Properties of random peer selection model for different botnet sizes ($B$) and different number of contacted peers ($C$) are plotted in Figure 5.7(a). Solid lines indicate the theoretical computation and the stars point the empirical estimation. Inner figures magnifies the region where $0 < C < 100$

bot cannot be accessed by the algorithm. In this subsection, given a bot-edge probability, we investigate how the P2P bots are expected to be oriented on a private mutual-contacts graph and what portion of the P2P nodes can be accessed by the dye-pumping algorithm.

To understand the structure of the subgraph formed by members of a P2P botnet on a mutual-contacts graph, suppose that there are $m$ bots in the network, and therefore the corresponding $m$ nodes on the graph. Let the set $X = \{X_1, X_2, ..., X_m\}$ denote these nodes and $p_e$ denote the probability of having an edge between any given $X_i$ and $X_j$, for $i \neq j$ where $1 \leq i \leq m$ and $1 \leq j \leq m$. Since $p_e$ is the same for any pair of $X_i$ and $X_j$, the subgraph formed by the nodes $X_1, X_2, ..., X_m$ on a private mutual-contacts graph is an Erdős-Rényi random graph [38] [39], where each possible edge in the graph appears with equal probability.

One interesting property shown by Erdős and Rényi is that, Erdős -Rényi graphs have a sharp threshold of edge-probability for graph connectivity [39]. More specifically, if the edge-probability is greater than the threshold then almost all of the graphs produced by the model will be connected. Erdős and Rényi have shown the sharp connectivity threshold is $\frac{\ln \beta}{\beta}$, where $\beta$ is the number of nodes in the graph. Therefore, if there are $m$ P2P bots in a network and if bot-edge probability is greater then $\frac{\ln m}{m}$, then the dye-pumping algorithm can identify all other P2P bots from a given seed bot with high probability. However, even if the bot-edge probability is below the threshold, the dye-pumping algorithm can still identify some of the P2P bots, which happen to be connected to the seed node on the private mutual-contacts graph. Therefore, we
are also interested in what portion of the nodes $X_1, X_2, ..., X_m$ are connected to the seed node by a short path. For this purpose, we can write an upper bound for the probability of accessing any of the $m$ bots ($X_i$) from the seed node at most $h$ hops as:

$$Pr < 1 - \prod_{j=1}^{h} \left( 1 - \left( p_e \right)^j \right)^{\text{Perm}(m-2,j-1)}$$

where $\text{Perm}(x, y) = \frac{x!}{(x-y)!}$ and $m$ is the number of P2P bots in the network. Figure 5.8 plots this probability for different $p_e$ and $m$ values. The maximum number of hops was set to $h = 3$.

It is observed that the probability approaches 1 as the bot-edge probability increases. The probability curves exhibit sharp increase around the connectivity threshold due to the sharp threshold phenomena mentioned earlier. Interestingly, it is also observed that, the probability grows faster as the number of nodes in the subgraph gets higher. Therefore, it gets easier for the proposed method to reveal P2P bots as the botmaster infects more hosts in the network.

We set the maximum path length to $h = 3$ since the Erdős-Rényi random graphs have short diameters. More specifically, assuming that there are not a very large number of P2P bots in a network and bot-edge probability is not very small, finding a P2P bot more than 3 hops away from the seed node is almost impossible.

In conclusion, according to the random peer selection model, members of a P2P botnet are expected to be closely connected to each other on a private mutual contacts graph despite large botnet sizes. This allows Dye-Pumping algorithm to identify the peers of a botnet.

\footnote{see Appendix B for details}
5.6 Limitations and Potential Improvements

The proposed method is able to identify P2P bots in a network as long as they are clustered through short and high capacity paths on a private mutual-contacts graph. Therefore, botmasters need to disturb this clustering structure in order to evade the proposed method. In this section, we review these possible evasion strategies, and their implications on the creation and maintenance of P2P botnets.

Eliminating Private Mutual-Contacts: One way to eliminate private mutual contacts is by increasing the popularity of private mutual-contacts that P2P bots in a network communicate with. If their popularity gets higher than the privacy threshold \((k)\), they will be omitted by the proposed scheme and will not result in edges in private mutual-contacts graphs. However, in order to achieve this, a botmaster has to control more than \(k\) hosts in that particular network, so that they can collectively boost a contact’s popularity beyond the privacy threshold. To defend against this strategy, the privacy threshold \((k)\) needs to be set as large as possible. Although, as discussed in Section 5.4, high \(k\) values impairs the recall performance of the proposed scheme, in most cases it is possible to find an appropriate \(k\) value. Also, for large networks potentially containing many P2P bots, the proposed technique can be applied on smaller subnets separately and independently to increase the likelihood that the number of P2P bots are below the privacy threshold.

Table 5.1: Summary of observed P2P botnet behavior. \(\Delta\) : Average number of unique IP addresses that a bot communicates with each day. \(\bigcirc\) : the number of mutual-contacts (the bot-edge capacities) between the two bots during 24 hours.

<table>
<thead>
<tr>
<th></th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\Delta)</td>
<td>(\bigcirc)</td>
<td>(\Delta)</td>
</tr>
<tr>
<td>Storm</td>
<td>5180</td>
<td>2861</td>
<td>4681</td>
</tr>
<tr>
<td>Waledac</td>
<td>1145</td>
<td>341</td>
<td>775</td>
</tr>
<tr>
<td>Nugache</td>
<td>45</td>
<td>0</td>
<td>53</td>
</tr>
</tbody>
</table>

Decreasing The Probability of Mutual-Contacts: Decreasing the probability of observing mutual-contacts between P2P bots is equivalent to decreasing the bot-edge probability \((p_e)\). As discussed in Section 5.5, a botmaster has to either(or both) increase the botnet size \((B)\) or
decrease the number of peers that each bot communicates with \( (C) \) in order to lower \( p_e \). It is clear that increasing \( B \) and decreasing \( C \) will inversely affect a P2P botnet’s robustness and efficiency. Although it may be possible for a botmaster to pull \( p_e \) down to a lower value, we observed in a controlled environment that peers of today’s botnets such as Storm and Waledac have very high bot-edge probabilities. To collect data for Storm and Waledac, we infected two Pentium IV, 512MB RAM Windows XP hosts, which were completely isolated from the rest of the network by a firewall. The firewall was also set to block all SMTP traffic to prevent any spam traffic. We observe that both Storm and Waledac communicate with fairly high number of unique peers during 24 hours, and therefore create many mutual-contacts as presented in Table 5.1. On the contrary, Nugache peers are less active and create far less mutual-contacts as observed in Table 5.1. Nevertheless, in Section 5.4, the proposed scheme is shown to successfully detect several Nugache peers, which are introduced to the network using the crawler data, despite their low communication activities. To collect data for Nugache, the bots were installed on a Pentium IV, 1GB RAM, running VMware Server with a Windows XP guest, as well as on bare metal machines on comparable hardware running Windows XP. The traces were captured within the protected network using a customized honeywall [91] and also using full-packet capture on an extrusion prevention system running OpenBSD with strict packet filter rules, as described in [33] The captured packets were converted to flow records using the SiLK tools [18] for establishing mutual contact sets and validating the algorithm.

**Using a Structured P2P Topology:** In principle, though not trivial as discussed in 5.1, a botmaster can adopt a structured P2P topology to decrease the probability of mutual contacts by making peers in a same network to communicate with different set of peers from each other. In such a case However, two or more networks can choose to share their flow records to exploit the mutual-contacts among P2P bots in different networks, which are unavoidable since the botmaster cannot know which networks would collaborate in the first place. Cooperating network can use privacy-preserving set operations such as [29] can be used to share data between networks without revealing any sensitive information.

**Poisoning Clusters:** The purpose of cluster poisoning for P2P networks is to destroy clustering structure of a graph by creating bogus edges [20]. Cluster poisoning appears to be very
hard to achieve in our context. In order to perform poisoning, a botmaster has to create an edge between a P2P bot and a benign node on a mutual-contacts graph. For this purpose, she needs to make both the bot and the benign host communicate with a mutual external IP. There are two ways to achieve this: the first one is that the botmaster can take control of the benign host and make it communicate with a desired mutual-contact. However, in this case the benign host will not be benign anymore and should be identified by the proposed algorithm. The second way is that the botmaster can listen to the traffic of the benign host and make the P2P bot contact with an external host which the benign host has communicated with. But it’s not easy for a botmaster to listen to network traffic of benign hosts unless she also possesses a router or a proxy in the same network.

5.7 Conclusion and Future Work

In this chapter, we presented a simple and efficient method to identify local members of a P2P botnet in a network, starting from a known member of the same botnet in the same network. The basic idea of the proposed method is that, the members of a same botnet are more likely to have mutual-contacts with each other than with benign hosts. We evaluate the proposed method using real P2P botnet (Nugache) data captured by a crawler. We also provide a mathematical analysis of the C&C structure of P2P botnets to characterize the performance of the proposed method. Both our analysis and experiments show that the proposed scheme is able to identify several dormant P2P bots in a network.

There are some limitations of the proposed scheme as discussed in Section 5.6. Perhaps the most important one is that, a botmaster can evade detection if she employs a structured P2P topology which ensures that her bots avoid mutual-contacts while communicating with each other. However, developing such a mechanism is not trivial for today’s botnets and currently available P2P topologies. Nevertheless, even if a botmaster achieves such a topology, two or more networks can mitigate this by sharing their network traffic, possibly in a privacy-preserving manner, to exploit the mutual-contacts which will possibly occur between peers in different networks. We leave the exploration of the benefits of data-sharing as future work. In addition, we plan to study on a new P2P botnet architectures, which potentially evade the
proposed scheme at least in some scenarios. This will allow us to further improve the proposed scheme to withstand potential evasion strategies, which might be employed by next generation botnets.
Conclusions and Future Work

In this thesis, we investigated the concept of network-based misuse detection in enterprise network settings. In summary, a network-based misuse detection scheme monitors network traffic for certain network traffic patterns which are exhibited due to a target malicious activity. For this purpose, first of all, such network patterns have to be determined in a network-based misuse detection scheme. To determine such patterns, one has to inspect network traffic at various levels, such as packet level, flow level, or host level, as the target malicious activity may exhibit patterns on one or more of those levels.

Once the network pattern for the target malicious activity has been determined, an pattern detection algorithm is executed to monitor network traffic in a network-based misuse detection scheme. Such algorithms have to be simple and computationally efficient, since they have to monitor high-volume of network traffic. Otherwise, a network-based misuse detection scheme using an inefficient algorithm could not be used for monitoring large networks.

In this thesis, we presented efficient misuse detection schemes for three particular problems, namely relay detection, correlated flow detection and P2P botnet detection.

We first presented an efficient relay node detection scheme. Due to their potential harmful effects, identifying relay nodes in the network can improve security policy enforcement. For this problem, we presented a statistical algorithm to detect delay-constrained real nodes in a network. The detection is based on the flow-level pattern that delay-constrained relay nodes have at least one pair of flows which have correlated packet timings. The detection algorithm
has linear time and space complexity and therefore can be used in online detection of relay nodes in large networks.

Experimental results show that the proposed relay detection scheme is quite robust against various possible adversarial or non-adversarial modifications on the underlying network traffic. In summary, the experiments reveal that the proposed scheme can withstand, up to some extent, packet delays which could be introduced due to packet processing in the relay node or for adversarial purposes. Also the algorithm is shown to be able to detect relay activity even if the flows contain chaff packets intended to defeat relay detection systems.

Second, we presented an efficient correlated flow detection scheme, which can be used in online stepping-stone detection applications. The proposed scheme continuously maintains sketches of network flows’ packet-timing information from a stream of captured packets at the border of a network. These sketches are then used to efficiently identify correlated flows, since the correlated flows have similar sketches. The proposed scheme computes flow sketches very efficiently by a streaming algorithm, which performs a few arithmetic operations for each packet. In addition, the sketches of a pair of correlated flows remain similar, even if the flows encounter various timing perturbations, thereby allowing the proposed scheme detect the correlated flows even under delays, jitter, chaff, etc to some extent. Finally, using the fact that correlated flows have similar sketches, the proposed scheme identifies correlated ingress/egress flow pairs among \( n \) ingress and \( m \) egress flows in \( O(n + \sqrt{nm}) \) time, as compared to known techniques, which requires \( O(nm) \).

In the context of correlated flow detection, we also presented a variant of the proposed sketching scheme for VoIP flows, which could be used for VoIP call tracking applications. The proposed scheme exploits a VoIP flow’s packet timings and bitrate variations to identify the audio contents of that flow with a short binary string. Our experiments show that the proposed scheme can successfully identify VoIP calls under various network impairments, such as delay, jitter and packet drops. We also showed that the proposed hash values can be efficiently computed in real-time, thereby allowing us to potentially employ it in large scale VoIP tracking applications.

Finally, we presented a simple and efficient method to identify local members of a P2P botnet in a network, starting from a known member of the same botnet in the same network.
The basic idea of the proposed method is that, the members of a same botnet are more likely to have mutual-contacts with each other than with benign hosts. We evaluate the proposed method using real P2P botnet (Nugache) data captured by a crawler. We also provide a mathematical analysis of the C&C structure of P2P botnets to characterize the performance of the proposed method. Both our analysis and experiments show that the proposed scheme is able to identify several dormant P2P bots in a network.

Network-based misuse detection has proven to be very efficacious in many scenarios. However, due to new attacks and malicious activities emerging everyday, a network-based misuse detection requires continuous improvement. Basically, for each new malicious activity, a new network pattern, which characterizes that new malicious activity, has to be determined. Following that, a new algorithm, which efficiently detects that pattern in network traffic, has to be designed. In this regard, we can roughly say that follow-up work of this thesis will be shaped according to new threats that will emerge in the future.

Aside from this, there is a great potential in extending the network-based misuse detection concept to the following two areas:

**Mobile/Cellular Data Networks:** Mobile devices are evolving very rapidly. Nowadays, most of the smart-phones allow users to run 3rd party applications. As we observed in computers, being more sophisticated and more complex inevitably make mobile devices more vulnerable to malware. As a result, mobile data networks have started to feel a significant threat from their very own users. A recent study has shown that a small number of compromised mobile devices can significantly degrade service to area-code sized regions [93]. Hence, it is extremely crucial to detect infected devices in a mobile network. We believe that network-based misuse detection approach can potentially yield very satisfactory results in this context.

**Security-as-a-Service:** The concept of Security-as-a-Service has been gaining significant attention recently. Offering network security measures as a service opens a new stream of revenue for Internet Service Providers. On the other hand, Security-as-a-Service clients potentially achieve high-level of network security without dealing with sophisticated network administration procedures. Monitoring client networks simultaneously gives Security-as-a-Service Providers a
unique advantage of accessing collective data. For instance, a provider can cross-correlate the
data collected from numerous sensors distributed across several client networks, thereby po-
tentially improving the performance of many security applications. Benefits of employing such
collective data in detecting correlated attacks has been discussed in [63]. Similarly, we envision
that our P2P botnet detection scheme performs much better if several networks collaborate and
share their network traces. In general, network-based misuse detection schemes can significantly
benefit from such collective data and collaboration and therefore has to be explored further. On
the other hand, service providers will obviously require computationally efficient and scalable
methods in order to reliably maintain the security of the networks of their numerous clients.
Therefore, the research on misuse detection under collaborative environment has to place great
emphasis on efficiency and scalability.
Bibliography


One Dimensional Random Walk

One dimensional random walk process basically characterizes the total displacement \( D \) from the origin after \( k \) steps, with step size \( s \), are taken. Each step is of size \( w \) and can be either to the right with probability \( p \) or to the left with probability \( 1 - p \). Considering the right hand side as positive and starting point as origin, the total displacement will be the summation of all steps such that:

\[
D = \sum_{i=1}^{k} w_i, \quad \text{where } P_r(w_i) = \begin{cases} 
p, & w_i = w \\
1-p, & w_i = -w
\end{cases}
\] (A.1)

Total displacement can also be written as:

\[
D = \left( \sum_{i=1}^{k_{\text{right}}} w \right) - \left( \sum_{i=1}^{k_{\text{left}}} w \right) = w(k_{\text{right}} - k_{\text{left}})
\] (A.2)

where \( k_{\text{right}} \) and \( k_{\text{left}} \) are number of steps taken towards right and left respectively. And since \( k_{\text{right}} + k_{\text{left}} = k \), we can write:

\[
D = w(k_{\text{right}} - (k - k_{\text{right}})) = w(2k_{\text{right}} - k)
\] (A.3)

Given probability mass function \( P(w_i) \) in Equation (A.1) and given \( k \) steps are taken, the distribution of \( k_{\text{right}} \) can be expressed as binomial distribution such that:
\[ P(k_{\text{right}} = z \mid k) = \binom{k}{z} p^z (1 - p)^{k-z} \quad (A.4) \]

For sufficiently large \( k \), Equation (A.4) can be approximated by a Gaussian distribution with \( \mu = \frac{k}{2} \) and \( \sigma^2 = kp(1 - p) \). Combining this with Equation (A.3), we obtain:

\[
\begin{align*}
P(D = w(z - k)\mid k) &\approx N\left(\frac{k}{2}, kp(1 - p)\right) \\
P(D = w2(z - k)\mid k) &\approx N\left(\frac{k}{2}, kp(1 - p)\right) \\
P(D = d\mid k) &\approx N(0, 4w^2kp(1 - p))
\end{align*}
\] (A.5)
Appendix B

Probability of Accessing a Given Node in Erdős-Rényi Graphs

We want to write the probability of accessing a given node at most \( h \) hops from the seed node in an Erdős-Rényi graph. Let \( Pr \) denote this probability and let \( m \) denote the number of nodes in the graph. First write the probability of not accessing a given node at most \( h \) hops (i.e. \( 1 - Pr \)). For this purpose, let \( P(NoA_i) \) denote the probability of not accessing a given node with \( i \) hops. Then we can write:

Meanwhile, in order to access a given node from the seed node at \( i \) steps, we have to select \( i - 1 \) nodes out of \( m - 2 \) remaining nodes and place them on the path between the seed node and the given node. Notice that, the order of the selected nodes are also important since reordering results in a different path. However, each reordering of the selected nodes doesn’t provide an independent path as several edges will be shared between paths obtained from reordering of the selected nodes. Therefore, \( P(NoA_i) \) is smaller than the product of the probabilities of not observing the paths obtained from all permutations. Since the probability of any given path of length \( i \) on a graph is \( (p_e)^i \), where \( p_e \) denotes the probability of a given edge appear in the graph, we can write:

\[
P(NoA_i) < (1 - (p_e)^i)^{Perm(m-2,i-1)}
\]

where \( Perm(x,y) = \frac{x!}{(x-y)!} \). Finally combining above two equations, we can write:
\[ Pr < 1 - \prod_{j=1}^{h} \left[ \left( 1 - (p_e)_{j} \right)^{\text{perm}(m-2j-1)} \right] \]