DISSERTATION

COOPERATIVE DEFENSE MECHANISM FOR DETECTION, IDENTIFICATION AND FILTERING OF DDOS ATTACKS

Submitted by

Negar Mosharraf Ghahfarokhi

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2016

Doctoral Committee:

Advisor: Anura P. Jayasumana
Co-Advisor: Indrakshi Ray

Ali Pezeshki
Yashwant Malaiya

ABSTRACT

COOPERATIVE DEFENSE MECHANISM FOR DETECTION, IDENTIFICATION AND FILTERING OF DDOS ATTACKS

Distributed Denial of Service (DDoS) attacks, which are intended to make the service unavailable for legitimate users, originate in a highly distributed manner providing the illusion of legitimate traffic. The number of attacks and the volume of traffic associated with attacks continue to increase dramatically. At these traffic intensities, the network infrastructure upstream from the intended victim also becomes severely affected necessitating that attack traffic be filtered as close as possible to the attack sources. However, it is difficult to anticipate and identify such nodes as the attacks originate at widely distributed nodes and spread through various routes. To successfully respond by dropping traffic, the mitigating approach must identify routers on traffic paths with significant attack traffic and respond with minimum effect on legitimate traffic. We develop a suite of solutions to address this problem. Cooperative Defense Mechanism (CDM), a distributed responsive defense mechanism for DDoS attacks is presented. CDM allows the identification of network routers closer to the attack sources and provides these routers a profile that facilitates discrimination between legitimate and malicious packets during the attack and thus enables them to drop traffic perceived as malicious in a distributed manner.

The cooperative defense model consists of three main components: (1) an attack traffic identification mechanism, (2) a filtering mechanism, and (3) a cooperative mechanism to identify the most effective points for filtering.

First, we investigate the features of attack and normal traffic to develop an identification model for the attack traffic. The main challenge is to detect attack traffic without misclassifying legitimate traffic thus avoiding the disruption of normal services. The parameters such as source/destination IP address, port number, size of packet are employed to establish the identification model and develop a scoring mechanism that provides the basis to create a history-based profile. The effectiveness of this approach is in blocking attack traffic and allowing the legitimate traffic at upstream routers. Experimental results, based on a recent traffic trace from Colorado State University, indicate that the filtering model is able to protect the victim node on average from ~95% of attack traffic while preserving ~75% of the normal legitimate traffic.

Second component, the filtering mechanism, is aimed at dropping attack traffic closer to the sources while minimizing the impact on legitimate traffic. The filters are propagated to selected upstream routers during the attack, keeping the communication and memory overhead associated low. A Bloom filter based mechanism is proposed to efficiently implement and disseminate the proposed history-based profile. Moreover, we introduce a novel data structure, which we refer to as the Compacted Bloom Filter (CmBF) that further improves performance, uses less storage, reduces the communication and computation costs, and provides the same functionality as a standard Bloom filter. However, unlike the standard Bloom filter, CmBF limits false positives significantly at the expense of false negatives in membership queries. Our work is motivated by a class of applications that must transmit Bloom filters over a network and endpoint machines having limitations on memory availability to meet specific false positive probability. We derive expressions for the false positive and false negative rates. Simulation results are used to validate the derived expressions and explore the tradeoffs when using the CmBF.

The third component identifies the most effective points where the Bloom filter based mechanism can be placed to mitigate the attacks. We do this by monitoring the network traffic during the attack period. The approach tries to minimize the modifications required to the routers and the current protocols to combat DDoS attacks. Such modifications will have a low complexity and will be scalable. Our solution benefits by using a recently developed technology, typically implemented as Small Formfactor Probes (SFP) using FPGAs, which helps gather, distribute, and analyze information from a distributed Ethernet network. SFProbes can plug into any SFP compatible elements such as routers in a way that does not interfere with the traffic flow. Our approach uses a subset of probes to identify the nodes that carry attack traffic. Extensive simulation in OPNET© with CAIDA attack dataset shows that our solution is able to place all the filtering routers in the vicinity of the attacker nodes (within the first three routers) and stops 95% of attack traffic while allowing 77% of the legitimate traffic to reach the victim node when the percentage of participating SFProbes in the network is 80%. Results also demonstrate the effectiveness of the mechanism in preserving valuable network resources and link utilizations for other end-users during the attack time, thereby preserving the service availability and minimizing the attack impact.

An analytical model for packet-pair dispersion signature in multi-hop networks is presented in the last part of our investigation. Path signature is an essential tool for numerous applications that need to distinguish between different network paths, diagnose problems, test protocols for realistic network conditions, and determine if two paths share common links. Our approach is a passive technique that relies on existing network traffic and hence does not consume network resources for measurements. The relationship between the input and the output gaps of packet-pairs and the corresponding distribution of end-to-end packet-pair dispersions are derived. This

derivation is then used to determine the signature characterizing the path where the path signatures can provide other properties of a path, such as an available bandwidth estimate, utilization, and bottleneck capacity of the path to monitor and diagnose network problems. The analytical model was verified using OPNET® simulations and was used to evaluate the impact of factors such as the number of hops, initial dispersion, link capacities, and cross traffic that affect the shape of the signature.

The proposed solutions are tested on either real world or realistic data to establish performance and accuracy. Real network traffic and DDoS attack dataset from sources such as Colorado State University (FRGP Continuous Flow Data), DARPA, CAIDA 2007 as well as Auckland University dataset have been used to evaluate the presented techniques.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my special appreciation and thanks to my advisor Prof. Anura Jayasumana, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow. This work would not have been a possibility without his most kind support. My sincere thanks go to my co-advisor, Prof. Indrakshi Ray, who gave me insightful advice and enhance my academic skills. I would like to thank you for their valuable time and helpful guideline that have enriched the quality of my work and dissertation. I am also extremely grateful to my committee Prof. Ali Pezeshki and Prof. Yashwant Malaiya for guiding and helping throughout the years.

This research was supported in part by the JDSU Advanced Technology Program, and their feedback helped produce significant outcomes. I would also like to acknowledge the Cyber Systems and Technology Group (formerly the DARPA Intrusion Detection Evaluation Group) of MIT Lincoln Laboratory, Center for Applied Internet Data Analysis (CAIDA), Waikato Applied Network Dynamics Research Group and the Information Marketplace for Policy and Analysis of Cyber-risk & Trust- IMPACT (formerly PREDICT) for their datasets which were used to evaluate the outcomes. I also acknowledge the OPNET/Riverbed University Program for the use of IT Guru Academic Edition software.

My deepest appreciation goes to my lovely husband, Hassan Dadkhah, who kindly stood by me and supported me all along and would like to dedicate this thesis. I can’t thank you enough for encouraging me throughout this experience and more importantly, for your patience.

Last but not least, words cannot express how grateful I am to my mother, father, my mother-in law and father-in-law, for all of the sacrifices that they’ve made on my behalf. I am blessed

with the most supportive sister and brother. I like to thank them for their unconditional love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Internet services and resources form an integral part of our daily life. It is, therefore, important to protect such services and resources from Denial-of-Service (DoS) attacks. DoS attacks, which are intended to make the service unavailable for legitimate users, have been known to the network research community since the early 1980s. From past incidents of DDoS attacks against commercial websites like Yahoo, E-bay and E*Trade, it is evident that all computer systems connected to the Internet are vulnerable to DDoS attacks [1], [2]. As emergency and essential services rely on their infrastructures, the consequences of DDoS attack could result in serious financial losses for E-commerce. It is for this reason(s) that defense against DDoS attacks has become one of the most important research issues in the Internet security community [3]. During a three-week period in mid-2001, researchers from the University of California, San Diego, detected approximately 128,000 DoS attacks against more than 5,000 targets [4]. On February 9, 2000, Yahoo, eBay, Amazon.com, E*trade, ZDnet, buy.com, the FBI, and several other websites fell victim to DDoS attacks, resulting in millions of dollars in damages and inconveniences [5], [6]. Despite significant research into countermeasures, DDoS attacks still remain a major threat today [4]. Recent examples include a record 400 Gbit/s DDoS attack against CloudFlare, a rate about 100 Gbit/s more than the largest previously seen DDoS attack [5]. The impact of DDoS attacks to legitimate users may vary from minor inconvenience to disastrous consequences. The frequencies and the impact of DDoS attacks have motivated researchers in the internet security community to provide techniques for preventing, detecting, and surviving such attacks [3]. The majority of proposed approaches have

failed to provide service availability in the presence of DDoS attacks. Towards this end, we investigate and develop a cooperative defense scheme based on an identification model for DDoS attacks located at routers closer to the attack sources. It is important to somehow distinguish attack traffic from legitimate traffic and allow legitimate traffic access to the victim node when an attack occurs.

In general, there are several factors that make defending against DDoS attacks hard. First, the traffic flow volume through the victim node can reach very high values, e.g., 400Gbps [7]. Second, the attack occurs in a highly distributed manner – this gives the illusion of legitimate traffic and makes detection difficult. In addition, a flooding attack can appear like a flash crowd, which occurs when a large number of legitimate users connect to a server simultaneously [8]. Differentiating the attack traffic from flash crowd traffic is one of the hardest steps in detecting and protecting against DDoS attacks. The presence of zombies that spoof the IP address of the source of the attack make it even more difficult to identify and trace the attacks back to their actual sources. Researchers have worked on providing a mechanism that uses different features of traffic to distinguish attack traffic from legitimate ones. However, these mechanisms are able to detect only attacks having those specific features [9]. Moreover, once the attacker knows the features that are of interest to the detection mechanism, he can develop strategies that bypass the detection mechanisms. For example, consider the scheme based on the abrupt change of traffic volume [10], [11], [12], [13]. In this approach, the attacker can bypass the detection mechanism by sending out an attack flow to change the statistics of the traffic. For the scheme based on flow dissymmetry [14], [5], [15], [16], [17], the attacker may use randomly spoofed source IP addresses and send out the same amount of SYN packets and FIN/RST packets that can go unnoticed when compared to legitimate traffic flows. An effective approach presented in [18] has

established a history of a legitimate IP address that has appeared before; however, the adversary can bypass this mechanism by starting to send IP address frequently before beginning the attack. Therefore, we need a more robust and efficient defense mechanism for effectively stopping attack traffic while allowing legitimate traffic to the extent possible. The first step of our proposed approach addresses this problem and establishes an identification model to detect attack without misclassifying legitimate traffic. An unusually high traffic volume may not by itself be a good indicator of a DDoS attack, as it can occur due to flash crowds as well. Thus, other features that help distinguish attacks from normal traffic have to be incorporated. We look into multiple features of DDoS attacks and normal traffic to extract characteristics that give information to discriminate a DDoS attack. These features and their relationships are used to establish high confidence IP address history and develop a normal traffic profile that can be used to categorize normal and attack traffic more accurately. The parameter set for establishing the identification model are source IP address and port number, destination IP address, size of packet, packet type (ICMP, UDP, TCP), frequency of IP address and for TCP traffic, those IP addresses with a successful TCP handshake. These features are used to develop a scoring mechanism that provides the basis for normal traffic signatures. The effectiveness of this approach in blocking attack traffic and allowing legitimate traffic at upstream routers has been demonstrated using DARPA dataset [19], CSU dataset [20] as well as CAIDA attack dataset [21]. The experiments indicate that the filtering model can protect the victim node from ~95% of attack traffic while allowing ~75% of legitimate traffic.

The second challenge after determining an identification model is how to effectively use history-based profile to prevent the attack in a cooperative defense model. During the attack period, prebuilt filters must be propagated to the routers at responsive points, and filter attack

traffic while preserving legitimate traffic. Moreover, the routers must check all packets destined for the victim node and this is a costly task for routers. Therefore, a well distributed, effective and efficient filtering structure would be a significant contribution. A Bloom filter is a space-efficient probabilistic data structure that is used to test for set memberships in many domains, including networking applications. Although such a check can be done efficiently, false positives are possible. A Bloom filter is used for representing the contents of the history-based profile, which is used to distinguish between legitimate and attack traffic. Such a filter helps reduce communication and computation costs and also storage requirements of the upstream routers that check for malicious traffic. This mechanism requires passing Bloom filters as a part of the message. Consequently, it is important to minimize the size of the filters, such that storage and processing costs are minimized. We introduce a novel data structure, which we refer to as the Compacted Bloom Filter (CmBF) that improves performance, uses less storage, and provides the same functionality as a traditional Bloom filter for applications, such as IP traceback, web caching, and peer-to-peer networks. For the same fraction of false positives, the CmBF generally offers memory saving two or more bits over the standard Bloom filter. This space saving is determined by false negatives introduced in the CmBF structure, which does not exist in the standard Bloom filter. The false negative rate has a tradeoff between false positive rate and memory reduction size, which are all discussed in detail throughout Chapter 4. The other advantage of the CmBF is that it is very simple and practical, much like the standard Bloom filter construction and therefore, this construction is useful in practice.

The final challenge to provide a cooperative defense mechanism is to determine where to apply the filtering mechanism. A victim node is a good point to discriminate a DDoS attack; however, the victim node is not a good point to filter the attack and may even crash. This is

exactly the goal of the attack. In addition, it is a late reaction to the DDoS attack, as huge attack traffic may strain Internet resources elsewhere. Since attackers cooperate to perform successful attacks, defenders must also form and cooperate with each other to defeat the DDoS attack. Therefore, a distributed approach that is able to effectively respond to DDoS attacks with minimum damage to legitimate traffic with minimal overhead is investigated. Placing filters at the upstream nodes may be expensive because all packets passing through such a node must be processed irrespective of whether a significant amount of attack traffic passes through it or not. Placing filters close to the victim nodes, in contrast, causes resource wastage, as the attack traffic passes through the network before it is stopped. Our proposed solution addresses this problem by using a recently developed technology, typically implemented as Small Formfactor Probes (SFP) using Field Programmable Gate Arrays (FPGAs). Our approach uses SFProbes, at a subset of ports in the network, to identify the upstream links, and thus, nodes which carry attack traffic and close to source of the attack. The technique has been validated with two real-world data sets. Compared with recent cooperative approaches, our scheme reduces processing overhead while maintaining the Quality of Service (QoS) for legitimate traffic during the attack period and making the scheme more deployable. Some works appear in distributed defense mechanisms for DDoS attacks [22] [23] [24] [25] [26]. Such schemes [22] [23] [24] detect attacks downstream close to the victim and this attack signature is propagated upstream to the routers doing filtering located close to the source of the attack. The drawback of these approaches is the difficulty of securely forwarding the attack signature to the upstream routers. Some schemes [23] [24] need a global key distribution infrastructure for authenticating and verifying the attack signature [27], which is used for differentiating attack traffic from the legitimate one. Note that creating an accurate attack signature is difficult due to the presence of zombies, who can spoof the IP

address of the attack source. Firecol mechanism [25] also presents a distributed collaborative system to detect flooding DDoS attacks. This mechanism relies on an Intrusion Protection System (IPS) located in the internet service providers. IPSs are effective in stopping attack traffic, but they have very large communications overhead and also significantly decrease the performance of the routers. We present a responsive defense mechanism that has low complexity and is scalable, named Cooperative Defense Mechanism (CDM), to combat DDoS attacks. Moreover, the approach tries to minimize the modifications required to the routers and the current protocols. Results for CAIDA attack set [21], for example, indicate that the responsive mechanism protects the victim nodes from 95% of attack traffic close to the sources of attack, while allowing 77% of legitimate traffic.

The last part of our investigation presents a new analytical model for the packet-pair based signature that accurately describes the behavior of packet-pairs in multihop network paths with multiple tight links. Packet-pair dispersions can be used to generate path signatures and the signatures assist in distinguishing paths from one another, monitoring networks, diagnosing problems, developing deeper understanding of network behavior, testing protocols for realistic network conditions, and determining if two paths share common links. An accurate model of the packet-pair technique is important not only for creating accurate path signatures for different scenarios, but also for enhancing the accuracy and efficiency of measurement techniques for parameter estimation, where our work mainly addresses this challenge.

The rest of the thesis is arranged as follows. Chapter 2 presents the precise problem statement addressed in this work. Characteristics of network traffic against DDoS attacks are addressed in Chapter 3. Compacted Bloom filter is presented in Chapter 4. Then, we proceed to the responsive defense mechanism. Chapter 5 discusses the distributed mechanism to protect victim node

against attack. A packet pair dispersion signature in multihop network is researched in Chapter 6. Conclusions are presented in Chapter 7. Finally, Appendix A lists key source codes.

CHAPTER 2

PROBLEM STATEMENT

Many works have tried to address DDoS in the past, but DDoS attacks remain a major security problem; detection and protection is hard, especially when it comes to highly distributed implementations. Most of the works deploy a single point (source-end, core-end and victim-end) to apply the responsive mechanism against DDoS traffic. As opposed to this approach, a distributed defense mechanism deploys different points for different tasks through the network to protect the victim node/network. In general, discriminating a DDoS attack at the victim point is straightforward compared to identifying it in a distributed manner; however, it is not a good point to filter the attack packets, and even if you do, it does not reduce the network traffic due to flooding attacks. On the other side, an efficient defense mechanism needs to drop attack packets at routers close to the source and not just at the victim node. Therefore, deployment at different points in the network is an important consideration for creating an accurate filter to separate good traffic from attack traffic, and finding an efficient method to filter. At present, there is no strong cooperation mechanism between routers and the victim node to identify and protect against the attack in a distributed scheme. Hence, providing a cooperative defense mechanism can be a significant improvement in this area.

Challenges that inspired this work and overall research goals and objectives are reviewed in section 2.1. In section 2.2, the approach employed under this work is presented.

2.1 Research Goals and Objectives

On the Internet, the lack of authentication allows the attacks to create fake identities and send malicious traffic with impunity. It is growing rapidly and is becoming harder to defend against;

therefore, protecting resources from frequent and large DDoS attacks motivates this research community. However, the existing security mechanisms have achieved only limited success in providing service availability during the attacks, and we need an efficient defense method for effectively stopping attack traffic while preserving legitimate traffic as much as possible. The goal of our research is to develop distributed and responsive defense mechanisms to address this problem in a network. As attack identification is an important procedure to direct any further action, the first challenge is how to discriminate attack traffic without misclassifying traffic. It is difficult to discover the identity of the attackers, because attacks make use of source addresses of IP packets of innocent nodes by faking source addresses, for example, by randomly generating them. Identifying clues which can be used as precursors for detecting such attacks are proactively analyzed. For this, we seek the differentiating features between attacks and regular traffic, and identify traffic variables which give information about the occurrence of DDoS attacks. A history-based profile is a promising approach to create a high confidence IP address history that serves to distinguish good traffic from the bad during the attack time. Such a profile then may be used to create a filtering mechanism to ensure that the victim node can continue to receive most of the normal traffic and remain operational. To prevent the network in the vicinity of the victim getting overloaded, it is necessary to develop a filtering mechanism that could be applied at different upstream router points. Thus an accurate and practical mechanism that requires low computational and storage overhead at routers is necessary. Furthermore, it is necessary to be able to deploy such a filter on demand with minimal overhead. Our research therefore will investigate novel data structures which are efficient and effective in a distributed network environment.

After creating an appropriate filtering mechanism, the challenge is where to filter the attack traffic in an efficient way. As we discussed, victim point is not a good point to filter the attack packet, and even if you do, it does not reduce the network traffic due to flooding attacks. An efficient defense mechanism needs to drop the attack packet at routers close to the source and not just at the victim. The ultimate goal of this research is to develop a distributed model based on a cooperative scheme that can filter attack traffic before it reaches and congests the victim network as close as possible to source of attack.

Recently developed technology that uses Small Formfactor Probes (SFP) using Field Programmable Gate Arrays (FPGAs) provide the potential for non-intrusive monitoring of traffic at router interfaces. This research makes effective use of SPF probes to identify the responsive points corresponding to routers which carry significant volumes of the attack traffic, and then activate packet filtering at such routers. Our approach aims at minimizing the modifications required to the routers with no changes to existing networking protocols, while maximizing the arrival rate for legitimate traffic and minimize the attack flow during the attack time.

Also of interest are techniques that allow the characterization of links in networks, so that network conditions can be non-intrusively monitored. Thus we investigate packet-pair based signatures that can accurately describe the behavior of packet-pairs in multihop network paths with multiple tight links. Accurate models of the packet-pair technique are important not only for creating accurate path signatures for different scenarios, but also for enhancing the accuracy and efficiency of measurement techniques for parameter estimation.

As discussed above, the potential damages caused by internet threats have become more serious, the need for defending against these threats has increased significantly. Among all network attacks, the DDoS attack is one of the most common, most harmful, hard to be traced

and difficult to prevent, and therefore is very serious. As the DDoS attack makes use of many different sources to send a lot of useless or harmful packets to the target in a distributed way, a proper solution to taking on DDoS threats needs a cooperation mechanism from multiple points of the network. However, this is difficult. Our research objective is to develop a cooperative defense mechanism (CMD) to address this problem. The research objectives of this work are organized around the following four aspects:

1- The identification of attacks is an important aspect of the defense of a DDoS attack, and affects the overall performance and effectiveness of the defense mechanism. When the attack is aborted or blocked by the defense mechanism, the victim should be able to continue operating normally. A key problem to tackle when trying to continue services under DDoS attacks is to distinguish attack traffic from legitimate traffic. Therefore, the first challenge in our research is to establish an identification model to discriminate between legitimate and attack traffic. It is important to maximize malicious traffic drops while minimizing legitimate traffic drops, and it is an important step and procedure to direct any further action. Selecting an effective set of parameters and defining a reliable history-based profile that can distinguish between normal packets and malicious packets is the main contribution of the first step.

2- The next objective, after generating an identification model, is how the provided profile can be applied to stop attack traffic. The filtering mechanism is an appropriate technique to reduce the effect of attack traffic; however, it has some challenges. Looking up the entire profile in the routers during the attack time is a costly task, as upstream routers must process all packets toward the victim-node. It will create an extreme overhead on the routers and is the other challenge in this mechanism; therefore, an efficient method to

conduct filtering is a significant contribution and will cause significant improvements in this area. To overcome this problem, we investigate a novel idea.

3- The next objective, after generating a filtering model, is where this mechanism should be deployed. According to the infrastructure of the internet, the responsive mechanism can be applied at different points. Victim-end point is a good point to detect and discriminate malicious traffic from legitimate traffic; however, the major problem with victim-end defense technique is that the victim points are not a good point for filtering attack traffic because the bandwidth might already be saturated. Moreover, it is too late to apply the filter where the source-end defense point is the appropriate point to filtering malicious traffic before the network congestion happens at the victim-end point [28]. Hence, a technique with a cooperative mechanism between those points can be an efficient defense technique against DDoS attacks. Our objective is to develop a novel structure as the cooperative defense model, in which the victim-end point is considered to discriminate DDoS traffic and generate a history-based profile, as described earlier, and the upstream routers are considered response points to block malicious traffic. Our proposed approach tries to minimize router resource consumption, identify heavy attack traffic flows, and, finally, filter the attack as close as possible to the attack sources, thus reducing the impact on the network.

4- The next objective derive a model for the packet-pair technique for multihop paths with multiple tight links, thus more accurately capturing the stochastic nature of cross traffic and its interaction with packet-pairs. In contrast, the existing stochastic delay model, presented in [29], describes the analytical relationship between the input and output dispersions under the assumption of a single tight link. Such a model can potentially lead

to techniques that detect links with abnormal traffic increases correspond to DDoS type attacks.

2.2 Approach

In this section, we establish the solution approaches to address the research objectives and goals identified so far. The first step is how to identify attack traffic without misclassifying traffic. Most of the existing solutions to detect attacks rely on monitoring the volume of traffic received by the victim. These methods may mistake a natural sudden increase in traffic as an attack. On the other hand, the attackers now tend to use spoofed IP addresses to perform DDoS attacks, which make it difficult to distinguish between normal flows and attack flows. In our research, we investigate the specific attack features as well as normal traffic characteristics to identify the attack traffic. To establish an identification model, a normal traffic history-based profile on some traffic properties, such as source IP, size of the packet, number of packets per IP, port number, and protocol type is created. This profile can assess to classify normal and attack traffic more accurately. To improve the identification quality, these features are used to develop a scoring mechanism that provides the basis for normal traffic signatures, and we investigate them in this section. According to our scheme, a collection of legitimate IPs creates a history that can differentiate good traffic from bad traffic during the attack time. Peng et al. [18] have shown that more reliable IP address are those that are used more frequently; therefore, to establish the history-based profile, we consider the frequency of the IP address as one of the features to create our identification model as well.

In general, there are two mechanisms to respond to DDoS attacks: rate-limiting and filtering. In rate-limiting techniques, the attack traffic is not totally filtered, but the rate is limited and all traffic towards the victim is subjected to a limited rate, whether the traffic is legitimate or

malicious. In the filtering technique, the attack traffic is totally filtered according to an identification model. Filtering is a promising technique and inflicts extremely low damage to legitimate traffic while quickly detecting outgoing attacks. When an attack is blocked by a filtering mechanism, the victim node can continue to operate normally, and filtering does not harm legitimate traffic. This feature makes filtering an interesting solution to DDoS attacks, although it has some challenges. As we explained, it is challenging to select an effective set of parameters and define a reliable profile that can distinguish between normal packets and malicious packets. Therefore, we investigated strategies for creating an accurate filter to characterize normal and attack traffic in the first step and addressed those challenges in Chapter 3. We further investigated characteristics of network traffic against DDoS attacks by analysis of network traffic collected from Colorado State University, Auckland University, and attack traffic from CAIDA.

It is important to minimize the size of the filters such that storage and processing costs are minimized. A Bloom filter may be used as the basis to solve this problem. A Bloom filter is a space-efficient probabilistic data structure that checks whether an element is a member of a set. In recent years, the popularity of Bloom filters has grown, and they are now being used in different areas, including peer-to-peer systems, web caches, database systems, spell checkers - [25] [26], and networking [27]. As a result, in the past decades, a lot of efforts have been put into improving Bloom filters. We present a new variant of the Bloom filter with the same functionality as the standard Bloom filter. We call it the Compacted Bloom filter (CmBF). Our work is motivated by applications that must transmit Bloom filters over the network and/or endpoint machines that have restrictions on available memory to meet specific false positive probability. We intend to reduce the message traffic that is blocked and introduce an efficient IP

address lookup mechanism during the attack. This approach degrades substantial overhead cost in the upstream routers when the filtering must check for every packet that comes to a victim node during attack time. Although a standard Bloom filter construct is space-efficient for simple membership queries, they are rather inefficient when the applications need to meet a specific false positive rate and the endpoints have limited storage capacity. Our work is motivated by applications that must transmit Bloom filters over the network and endpoint machines have restriction on memory availability to meet specific false positive probability. Our research presents the Compacted Bloom Filter (CmBF) structure to use as a novel idea to address this challenge, which improves performance in large networks and reduces memory compared to the original Bloom filter structure. The key idea of the proposed CmBF is that the CmBF indicates the location of 1's that was set in the standard Bloom filter. This procedure reduces memory requirements by introducing false negatives in membership query.

In order to combat such attacks, a distributed defense mechanism is needed that will thwart the attack traffic in real-time. We proposed one such mechanism that identify filters that block attack traffic and allow legitimate traffic as close to the source node as possible, so that network resources are not wasted in propagating the attack. Our solution benefits by using a recently developed technology, typically implemented as Small Formfactor Probes (SFP) using FPGAs (Field Programmable Gate Array). An example of such hardware is JDSU SFProbes and Packet Portal [30]. Packet Portal is a new approach to gathering, distributing and analyzing information from a distributed Ethernet network. These packet portals have been deployed on operating networks for network monitoring functions. Our approach uses these probes, at a subset of ports in the network, to identify the upstream links, and thus nodes which carry attack traffic.

Effectiveness of the scheme is evaluated using extensive simulation in OPNET© with a real-world network topology and is presented in Chapter 5 in detail.

In the last part of the research approach, the analytical model provides deeper insights to path signatures in the presence of multiple tight links, thus enabling efficient and accurate network monitoring, problem diagnosis, and estimation of link and path parameters, such as end-to-end network bandwidths and link capacities. We investigate the effects due to initial dispersion, cross traffic and the number of the hops on end-to-end signatures using the analytical model. We also show how link properties such as available bandwidth and arrival rate of cross traffic can be estimated based on link signatures. This is the case with passive techniques that rely on existing network traffic, thus not consuming network resources for measurements [31].

CHAPTER 3

ON THE CHARACTERISTICS OF NETWORK TRAFFIC AGAINST DDOS ATTACKS

3.1 Introduction

Distributed Denials of Service (DDoS) attacks have become one of the most serious threats on the Internet. With large-scale DDOS attacks, it is necessary to stop attack traffic closer to the sources without disrupting legitimate traffic to the maximum extent possible. A responsive defense model that filters potential attack traffic and prevents it from reaching the victim network is developed. We investigate the features of network traffic that can be used for discriminating attacks from normal traffic. We use these extracted features to develop an accurate and robust signature-based filtering model that forms the basis of a detection and defense mechanism. A Bloom filter based mechanism is proposed to efficiently implement and disseminate the proposed signature-based model; it helps reduce the communication and computation costs and the storage requirements of the upstream routers that check for malicious traffic. The approach is verified and evaluated using the DARPA 1998 dataset [19] as well as extensive analysis over Colorado State University [20] and University of Auckland dataset [6]. The experimental results show the effectiveness of our new scheme in blocking attack traffic and allowing most of the legitimate traffic at upstream router points. We analyze the impact of different packet features on the accuracy and efficacy of the filtering mechanism.

Distributed Denial-of-Service (DoS) attacks on Internet based systems and infrastructure have become a common occurrence. Multiple machines, which typically are compromised, flood the victim at fast rates. Detecting these machines and isolating them in a timely manner is non-trivial. From the incidents of DDoS attacks against commercial websites like Yahoo, E-bay and

E*Trade, it is evident that all the computer systems connected to the Internet are vulnerable to DDoS attacks [1] [2], [32], [33], [34] A coordinated DDoS attack within a single day on them and several other websites resulted in millions of dollars in damages and service unavailability [5] [6]. Latest instance include the biggest DDoS attacks against CloudFlare with 400 Gbit/s and break the record of largest DDoS attack [7].

The effect of DDoS attacks can vary from minor disturbance to devastating outcomes. The iteration and the impact of DDoS attacks have motivated researchers to provide techniques for preventing, detecting, and responding [3] to them while most approaches have failed to provide service availability during the attacks. Our approach address this issue and demonstrate an effective method to distinguish attack traffic from legitimate traffic and allow legitimate traffic access to the victim node in the event of an attack.

Researchers have worked on providing a mechanism that uses features that distinguish attack traffic from legitimate ones. However, these mechanisms have weakness within detection where they can only detect the attacks contain those specific features [9]. Moreover, the attacker is able to develop mechanism to bypass once knows those features. For instance, the scheme based on flow dissymmetry [14], [5], [15], [16], [17] can bypass with attacker by use the random spoofing source IP address and send out the same amount of SYN packet, and FIN/RST packets that can go unnoticed when compared with legitimate traffic flows. Moreover, a major drawback of the proposed approaches is that they cannot discriminate flash crowd traffic from DDoS attack traffic. An efficient approach [2] called a History-based IP Filtering (HIF) was proposed to discriminate good traffic from bad traffic. This approach is based on monitoring the number of the new source IP addresses instead of the volume of the traffic. Jung [35] discovered that most of the IP addresses in a flash crowd traffic appeared on the website before, while very few IP

addresses have made a prior appearance in the case of DDoS attacks. HIF keeps a history of the legitimate IP addresses that have appeared before and applies filters in the edge router based on this history. However, an adversary can bypass this mechanism by starting to send packets with its IP address before conducting the attack. Therefore, we need a more robust and efficient defense mechanism for efficiently stopping attack traffic while allowing legitimate traffic to the extent possible.

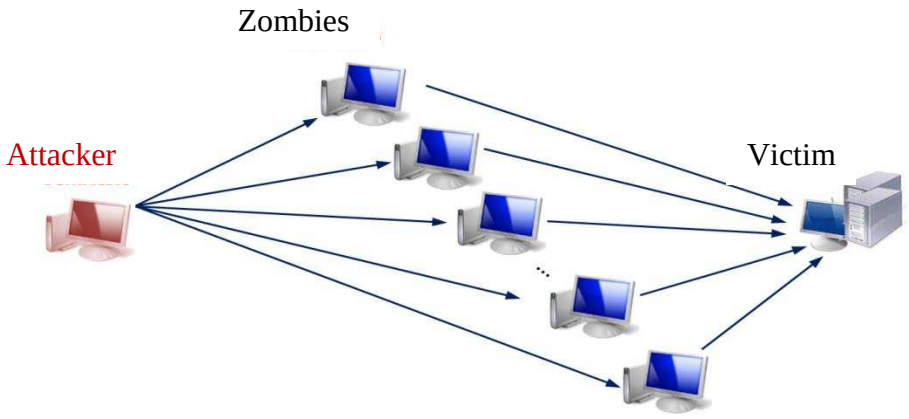
The goal of this chapter is to introduce and evaluate a responsive defense mechanism against DDoS attacks. The first challenge is how to detect attack traffic without misclassifying legitimate traffic. We look into multiple features of DDoS attacks and normal traffic to extract characteristics that give information about the occurrence of the DDoS attack. These features and their relationships are used to establish high confidence IP address history and to develop a normal traffic profile that can be used to categorize normal and attack traffic more accurately. The high confidence IP history will be used to create filters that differentiate the good traffic from the bad ones during the attack. When the attack is blocked by an efficient filtering mechanism, the victim node can continue to receive most of the normal traffic and remain operational and filtering does not impact legitimate traffic. This feature makes filtering technique a desirable solution; however, filtering has additional overheads. The effectiveness of our approach is validated using the Colorado State University [20], University of Auckland [6] as well as DARPA 1998 Intrusion Detection Dataset [19] as describe in Section 3.4.1 and 3.4.2. During an attack, the filters must be propagated to the upstream routers, preferably as close to the source of attack as possible. Moreover, these routers must check each packet to determine whether it is legitimate or not. Towards this end, we propose an efficient approach based on Bloom filter data structure, which facilitates a distributed response from routers closer to attack

nodes and serves to prohibit attack packets from reaching the victim node while allowing the legitimate traffic to pass through. We will address this challenge in Chapter 5.

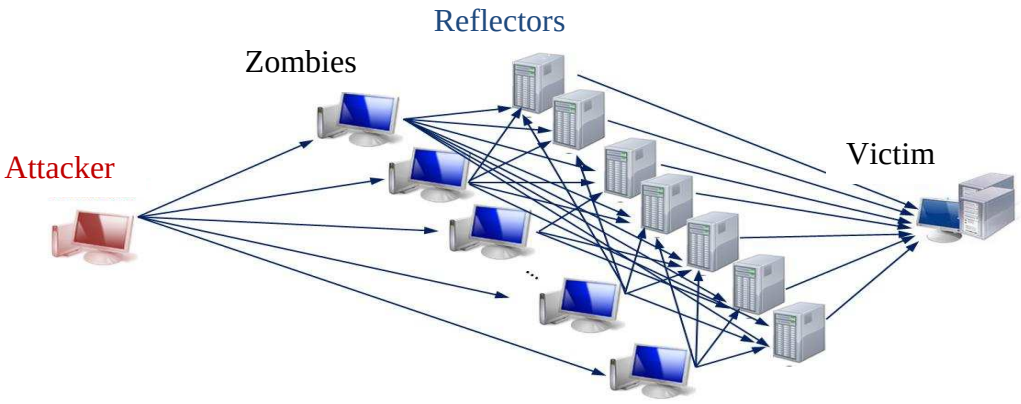
The rest of this chapter is organized as follows. Section 3.2 reviews to the DDoS attack characteristics. Section 3.3 describes the proposed method to discriminate DDoS attacks from normal traffic in details. Section 3.4 validates our model using DARPA dataset, University of Auckland as well as Colorado State University dataset. Section 3.5 concludes the chapter.

3.2 DDoS Attacks

In general DDoS attacks send large volumes of packets and consume critical resources in a network that makes the service unavailable for legitimate use. The attacker usually uses spoofed IP addresses in order to make a trace back more difficult, thereby thwarting the discovery of the real source of the attack. Flooding attack is one of the well-known types of DDoS attacks (e.g., spoofed/non spoofed UDP flood, ICMP flood, TCP SYN flood, DNS flood, VoIP flood, etc. [36], [8]) that focus on exhausting bandwidth of the victim's network. Typically, DDoS attacks involve two steps. In the first step, some vulnerable systems are compromised and attack tools are installed on these systems. These compromised machines are called "zombies". In the second step, the attacker requests the zombies to send a flood of packets to the victim(s) [37]. DDoS flooding attacks can be divided into two types: direct attacks and reflector attacks. In a direct attack, the attacker sends a flood of packets directed towards the victim through the zombies [28] as shown in Figure. 3.1 (a). In the reflector attacks, there is one more layer between the attacker and the victim. The attacker sends attack message to the reflector machine through the zombies and uses the IP address of the victim as the source IP address. The reflector machine replies in response to the victim, causing packet flooding as shown in Figure. 3.1 (b).



(a)



(b)

Figure. 3.1. Flooding attacks. (a) direct attack (b) reflector attack

3.3 Identification Mechanism

In this section, we explain our approach for classifying normal and attack traffic. Our responsive defense mechanism differentiates between legitimate and attack packets during the attack and responds to it by causing traffic perceived as malicious to be dropped. To successfully respond to attack the approach must accurately detect the attacks and respond by minimally blocking legitimate traffic. It should also have low communication, computation, and storage overheads.

In order to continue to provide services under DDoS attacks, we need to distinguish attack traffic from legitimate traffic. An unusually high traffic volume may not be a good indicator of a DDoS attack, as it can occur due to flash crowds as well. We need to consider other features that help distinguish DDoS attacks from normal traffic. Since many of the previous approaches depend on monitoring the volume of the traffic, detecting distributed flooding attacks is hard. During bandwidth attacks, most source IP addresses are new to the victim, whereas most IP addresses in a flash crowd have appeared at the victim before. Specifically, Jung et al. [35] mentions that around 82.9% of all IP addresses involved in flash crowd events have sent prior requests. Peng et al. [38] advocate the use of network connection history to distinguish good packets from bad ones. Many enterprises, e.g., universities, banks, etc., also have a group of users that access their services regularly. Although the user base fluctuates with new additions, deletions, etc., in general such a base changes at a much slower time scale compared to attacks and disruptions. These observations often form the basis of mechanisms to filter out the attack traffic at the victim. This provides a practical solution based on IP address history of previous addresses that arrive at the victim node to distinguish bad and good packet. However, the attacker can bypass the history-based IP filtering if attacker starts to send requests and communicate with a victim node prior to conducting the attack.

We address this problem by using multiple features that help distinguish normal and attack traffic and by developing accurate normal traffic signatures and a scoring mechanism based on these features. The normal traffic signatures and the scoring mechanism help create a high confidence IP history that serves to differentiate the good traffic from the bad traffic during the attack time. Using this high confidence IP address history, we can defend against DDoS attacks as explained below.

3.3.1 Feature Selection

We investigate DDoS attacks in details in order to find the features that help distinguish between attacks and normal traffic. Lee et al. [39] mention some parameters such as source/destination IP address, port number, and packet type (ICMP, TCP, UDP) that have been used to detect DDoS attacks. We will use them together with packet size to form the features in our identification model. The other feature is the frequency of an IP address. Recall that, Jung et al [35] determined that most source IP addresses are new to the victim during bandwidth attacks, whereas with flash crowd traffic known source IP addresses are most common. Thus, the frequency of an IP address is a feature that may help distinguish an attack from normal traffic. Since our goal is in creating a history based only on legitimate and valid IP addresses, we only consider those IP addresses with a successful TCP handshake. Note that, a spoofed IP address will not have a complete three-way handshake [38]. The attackers are therefore forced to use legitimate IPs and establish a three-way handshake. This limits the number of IP addresses that an attacker can use and the attack can be identified by monitoring for abrupt change in the traffic volume during the attack time. Our approach uses a much more comprehensive set of features compared to existing identification approaches, and use them in an integrated manner to create filters as described later in Section 3.3.2.

3.3.2 Metrics

Our parameter set, denoted by *P*, for establishing the identification model for historical IP addresses are as follows:

- Source IP address
- port number

– Size of packet

In our model $K=3$ where we consider source IP address, size of packet and port number as the parameters and the other features such as Packet type (ICMP, UDP, TCP) and establish three-way handshake for TCP traffic can be added. For each of the set of parameters in the set $P= \{P_i \mid i=0,..K\}$ our model maintains the frequency of occurrence within a certain time window and it's Cumulative Distribution Function (CDF).

Frequency (f_{ij}): For each P_i, we evaluate the frequency of occurrence of different values $f_{i1}, f_{i2}..., f_{iM}$ within a time window. In our case, $i=1$ corresponds to source IP address, $i=2$ corresponds to port number and $i=3$ corresponds to size of packet.

f_{ij} = *Number of packets for which parameter i has the value j within a time window* (3.1)

Note that the total number of packets is $N= \sum_{\forall j} f_{ij}$. As an example, consider case of the source IP address ($i=1$), there are 1000 different IP addresses within the time window, then M=1000. For instance, suppose there are a total of 5000 packets and a particular IP address j occur 30 times, then $f_{1j}=30$ and $N=5000$.

Cumulative Distribution Function (*CDF*): The CDF $C_X(x)$ measures the probability that the variable X takes on a value less than or equal to x.

Now consider the parameter i. Let F_i be defined as the random variable representing $\{ f_{ij} \mid \forall j \}$, CDF of F_i is defined as follows:

$$C_{F_i}(x) = P(F_i \leq x) \quad (3.2)$$

For instance C_{F1} *(10)* demonstrates the probability that source IP address frequency is less than or equal to 10. We use these metrics and features for generating the IP address history, which is addressed next.

3.3.3 History Creation

Our goal is to define a good signature to make the IP address database accurate and robust, and to make it hard to be passed by an attacker. Our approach overcomes some of the deficiencies of existing approaches, such as use of only the IP address field as a feature useful for distinguishing attack and normal traffic. A key observation that can be used for defense against DDoS attack is that the DDOS attacks tend to use randomly spoofed IP addresses [8] and the other packet features, such as, port number and size of packet, are selected randomly as well. Moreover, the interaction of these features also exhibits some anomaly when compared to that of the normal traffic. Therefore, we make use of the individual features, and also the interactions and correlations, in defining the signature. The signature is based on the CDF of each parameter's frequency during the training period. This signature assists us in selecting reliable IP addresses during the training period and later filters traffic based on those IP addresses. This signature for each feature determines which values occur more frequently during normal traffic conditions. The next step is assigning scores to IP addresses in the training period and generating the source IP address history. The score value for each IP address depends on the frequencies and the signatures of the selected features. Frequency threshold a_l, which indicates the level of reliability in our model and the corresponding scores are presented in Table 3.1. For each selected feature value, if it is more than a_l in the related signature, it is assigned a score b_j indicating the confidence level. Here four levels are defined, $b_4 > b_3 > b_2 > b_1$, to assign different reliabilities to different IP addresses based on how frequently different feature values occur. In

our model, α_i selected from 70%, 50%, and 30% indicates how the selected feature follows the signature of normal traffic condition. This method allocates highest weight to the top 30% of the IP addresses (*i=1*), port numbers (*i=2*) and sizes of packets (*i=3*) that occur most frequency. Note the value of frequency threshold α_i can be adjusted dynamically for each victim node based on the frequency and the signatures of selected feature for that point. Our selection of particular values for α_i is based on experience with different datasets, but it may be fine-tuned as necessary. According to our model, when the selected feature occurs more frequently in normal traffic, as indicated by the signature of normal traffic, we give it the higher weight for it by assigning score b_4. In contrast, when α_i has value less than 30% assign the lowest confidence level factor b_1 to it. Such a feature value is considered as something that is not a common occurrence in normal traffic and thus dropping such packets will have less of an effect on the normal traffic. Frequent occurrence of such a value during an attack therefore is also considered as a potential threat.

Table. 3.1. The score manager

Case	Frequency	Conclusion	Score
1	$\geq\alpha_1$ (=70%)	High Confidence	b_4 =4
2	$\geq\alpha_2$ (=50%)	Medium	b_3 =3
3	$\geq\alpha_3$ (=30%)	Low	b_2 =2
4	$\prec\alpha_4$ (=30%)	Potential threat	b_1=1

For example, consider source IP address 129.1.1.1 with frequency 10000. When C_{F1} *(10000)*$\geq$*70%,* it means according to IP address signature in training window 70% of IP addresses occur at frequency less than 10000. i.e., this is based on top 30% of occurrence rates of IP addresses and we give a high score for this feature.

To summarize, the above procedure has allocate weight from the set $\{b_1,..,\ b_4\}$ to each IP address, each packet size, and each port number. Next we assign a net score S_β for each IP

addresses β so that those IP addresses with high net scores can be included in the profile for normal traffic.

$$S_\beta = (f_{1\beta} * (allocated\ \ b\ \ for\ IP address\ \ \beta) + Max_{j,k}((f_{2j} * (allocated\ \ b\ for\ \ port\ number\ \ j) + f_{3k}(allocated\ \ b\ for\ size\ of\ \ packet\ \ k)))/N$$

$$Where\ \ j\ and\ k\ correspond\ \ to\ port\ number\ and\ size\ of\ diffrent\ \ packets\ \ with\ IP\ address\ \ \beta \quad (3.3)$$

S_β is defined as the score value for each IP address β and is determined according to the frequency f_{ij} and confidence degree b_j in Eq. 3.3. Thus S_β consists of three components. For the above example, the first component of $S_{129.1.1.1}$ is computed for IP address 129.1.1.1 based on $f_{1129.1.1.1}$ and b_4 that is 4. The second and third components are selected by taking the maximum value of the sum of corresponding values for port number and size of packet over different packets with this IP address The IP addresses that have an overall score S_β higher than a threshold v are selected as legitimate IP addresses for our history. We will discuss about the impact of threshold value v in Section 3.4.1.3. As noted, for TCP connections we consider one additional condition --only IP addresses with a successful TCP handshake are classified as valid. Therefore, for TCP connections, source IP addresses which have established a three-way handshake and have a score higher than ν are selected. This helps us create a signature-based IP address history and this history can be used through routers to perform filtering for the victim node.

3.3.4 Bloom Filter Mechanism

Since the filtering mechanism is to be applied closer to the victim point, and the network bandwidth may already be saturated during an attack, transferring the entire history and looking

it up in the upstream routers during the attack is expensive since upstream routers must process all packets targeted towards the victim node. This will impose an additional overhead on the routers. To overcome this problem, we propose a Bloom filter [40] based mechanism to represent efficiently the filtering mechanism for withholding the malicious packets. This way, the victim node need not transfer the full list of IP addresses in the history to the upstream routers, but instead needs to transfer only the Bloom filter that represents the contents of the IP address history as shown in Figure. 3.2. It causes a significant reduction of message traffic and introduces an efficient filtering mechanism that can be deployed closer to the attack sources during the attack. This approach reduces the overhead cost significantly in the ingress routers since every packet that comes to a victim node during attack must be checked.

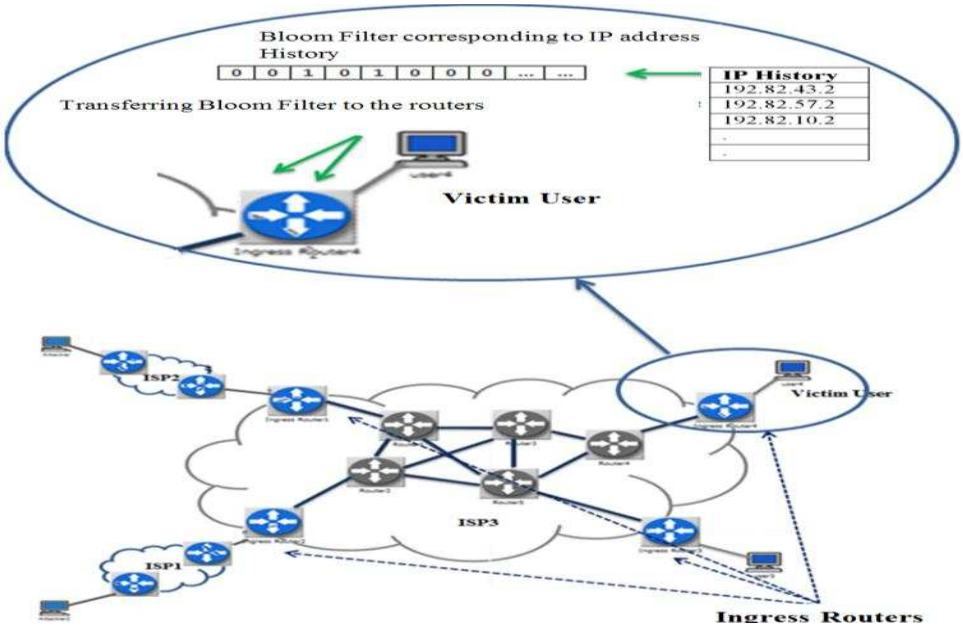


Figure. 3.2. Bloom filtering mechanism for history-based profile

Bloom filter is a space efficient probabilistic data structure for presenting whether an element is a member of a set or not. We briefly review Bloom filter structure as we will discuss about Bloom filter in detail in Chapter 4. A Bloom filter for a set $S = \{s_1,s_2,...,s_n\}$ of n elements is described by an array of m bits, all initially set to 0. A Bloom filter uses k independent hash

functions $h_1,...,h_k$ with range $\{1,...,m\}$. We make the natural assumption that these hash functions map each item in the set of interest to a random number uniform over the range $\{1,...,m\}$ for mathematical convenience [41]. For each element $s \in S$, the bits $h_i(s)$ are set to 1 for $1 \leq i \leq k$. A location can be set to 1 multiple times, but only the first change has an effect. To check if an item x is in S, we check whether all $h_i(x)$ are set to 1. If at least one bit is not 1, then clearly x is not a member of S. If all $h_i(x)$ are set to 1, we assume that x is in S, although there is a probability of a false positive, where it suggests that an element x is in S even though it is not. For many applications, this is acceptable as long as the probability of a false positive is sufficiently small. The probability of a false positive for an element not in the set, or, the false positive rate, can be calculated in a straightforward fashion, given our assumption that hash functions are perfectly random [40]. After all the elements of S are hashed into the Bloom filter, the probability p that a specific bit is still 0 is:

$$p = (1 - \frac{1}{m})^{kn} \approx e^{-kn/m} \tag{3.4}$$

We let $p = e^{-kn/m}$. The probability of a false positive f is then:

$$f = (1 - (1 - \frac{1}{m})^{kn}) \approx (1 - e^{-kn/m})^k = (1 - p)^k \tag{3.5}$$

We use the asymptotic approximations p and f to represent respectively the probability a bit in the Bloom filter is 0 and the probability of a false positive from now on for convenience.

It is clear that there are three fundamental performance metrics for Bloom filters: the probability of error (corresponding to the false positive rate f), size of the Bloom filter array (corresponding to the array size m) and the number of hash function k. Note that Bloom filters

are highly efficient even if $m=cn$ for a small constant c, such as $c=8$. Although Bloom filters introduce false positive rates, their use is justified because of the reduction in the network traffic and overhead. In our analysis, we limit the false positive rate to 0.1. In order to achieve this false positive rate, we should set c to 5 in our mechanism. That means the Bloom filter array is 5 times the number of IP addresses kept in the history. Moreover, according to Eq.3.5, 4 hash functions are required to provide a Bloom filter from the IP address history.

3.4 Model Validation

The objective of this experiment is to evaluate the accuracy and robustness of our filtering model to protect against DDoS attacks. To illustrate the effect of signature to distinguish attack and normal traffic we examined three different datasets with extensive analysis and evaluation. We will start by the DARPA 1998 Intrusion Detection dataset [19] and then we will discuss the result based on traces collected from University of Auckland in New Zealand [6]. The last part of our evaluation shows the result based on network traffic dataset collected from Colorado State University.

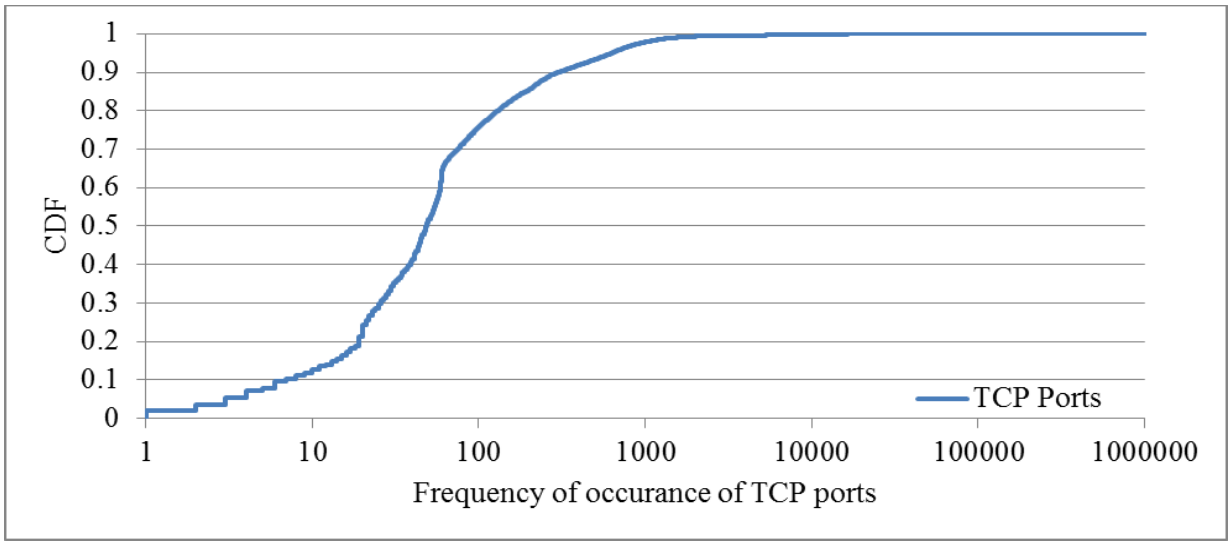
3.4.1 DARPA 1998 Intrusion Detection Dataset

This set is selected as it is the comprehensive traffic trace that is available containing the full header information of packets. It contains 7 weeks of training datasets to generate signatures and establish an IP address history and 2 weeks of testing data set to evaluate our technique. Moreover, the training part of the dataset consists of normal traffic as well as labeled attacks. We selected this dataset for our evaluation as our goal is to evaluate how good the signature and scoring mechanism is in discriminating malicious and normal traffic without misclassifying them.

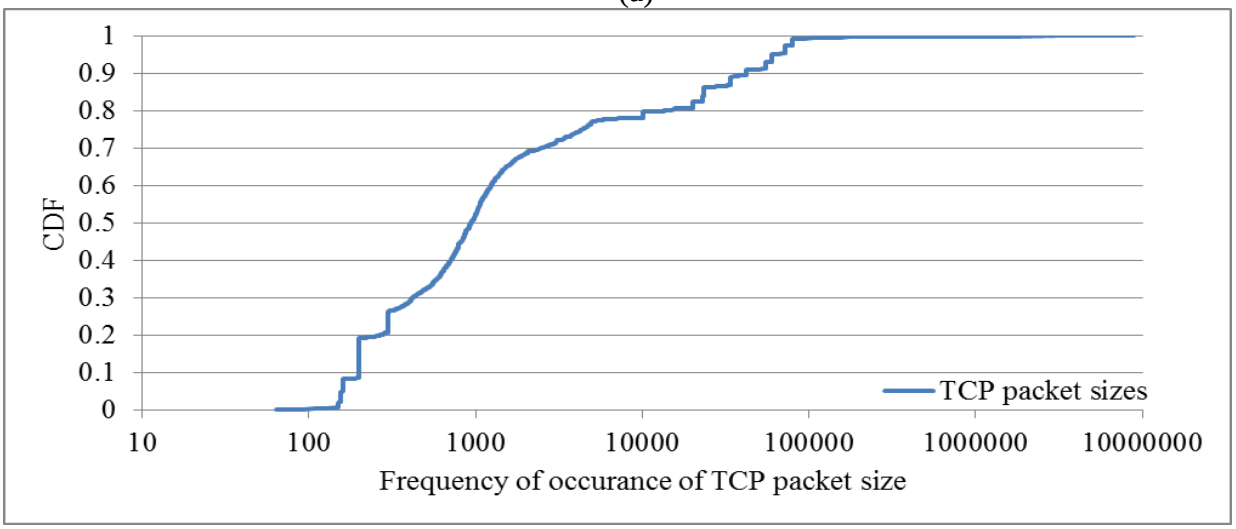
3.4.1.1 Experiment Setup

The first step is to create the IP address history from the training dataset, according to the generated signatures and the overall score of selected features. The second step constructs the Bloom filter based on the results of the first step. In the first step, the signature of port number, the size of the packet and IP address's frequency rate are created for each traffic type (ICMP, TCP, UDP) separately. Figure 3.3 shows the signature of port number, packet size and IP address's frequency rate for TCP traffic in the DARPA 1998 dataset. As we note the signature contain CDF of each parameter's frequency during the training period. According to Figure 3.3 for TCP traffic, port number frequency around 100 can achieve the highest score as $C_{F2}(100) \geq 70\%$. Same condition exists for packet size and IP address with frequency around 1500 and 5 respectively.

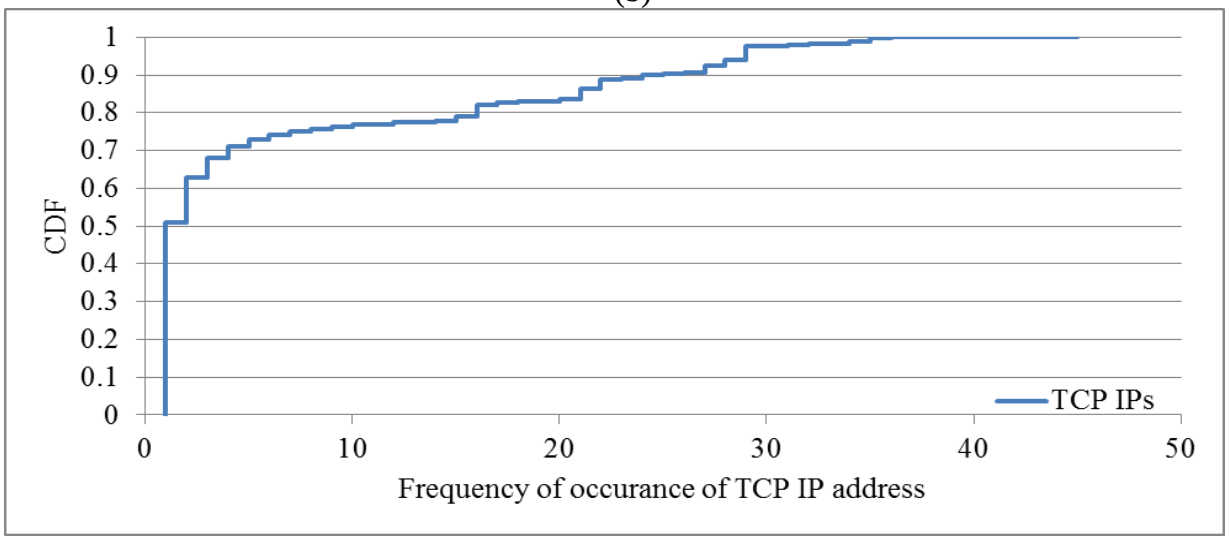
In the next step, the IP address history is created based on the overall score and value of v. In our experiment, v is determined to be 0.36. This value means the signature of at least two selected features should have α_j more than 50% and one should have it more than 30% in training time; however, this value can be set to a higher value for more protection. Determining the value of v is a trade-off among history size, history accuracy, and protection rate. With increasing the value of v, size of history and protection rate increases and accuracy of history to pass legitimate traffic decreases. In Section 3.4.1.3 we will show the impact of the score threshold v on accuracy and robustness of the history mechanism as well.



(a)



(b)



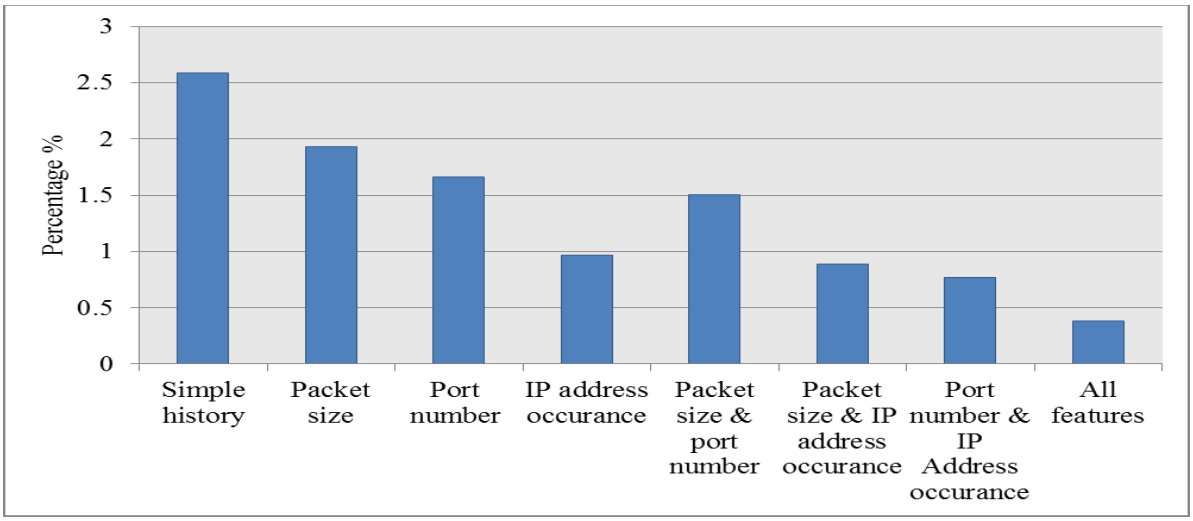
(c)

Figure. 3.3. Signatures of TCP traffic: (a) port number, (b) packet size, (c) IP address

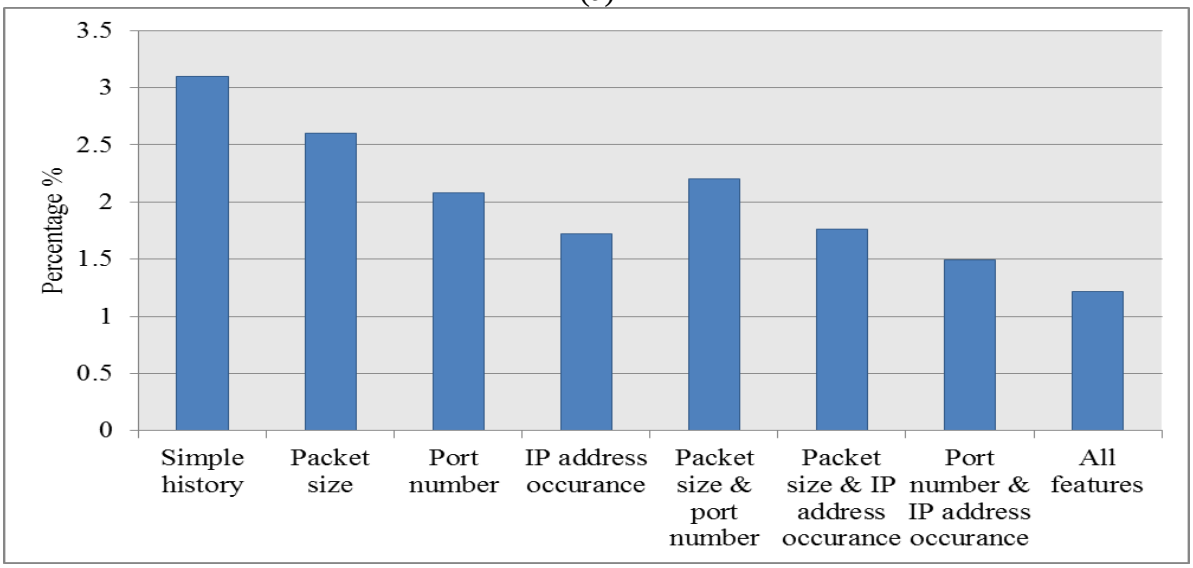
3.4.1.2 Results

Signature-based IP address history is evaluated on the basis of these parameters: (i) *Attack traffic detection rate* is the percentage of attack data set that cannot pass the Bloom filter and reach to the victim points during the attack time; (ii) *Normal traffic detection rate* is the percentage of normal tracing traffic that can pass the Bloom filter during the attack period; (iii) *False positive rate* is the percentage of normal traffic that is prohibited by the Bloom filter in training period, (iv) *False negative rate* is the percentage of attack traffic that passes the signature and is considered as legitimate traffic during training period, (v) *Size of Bloom filter* is based on the length of the Bloom filter array that is transferred to the upstream routers.

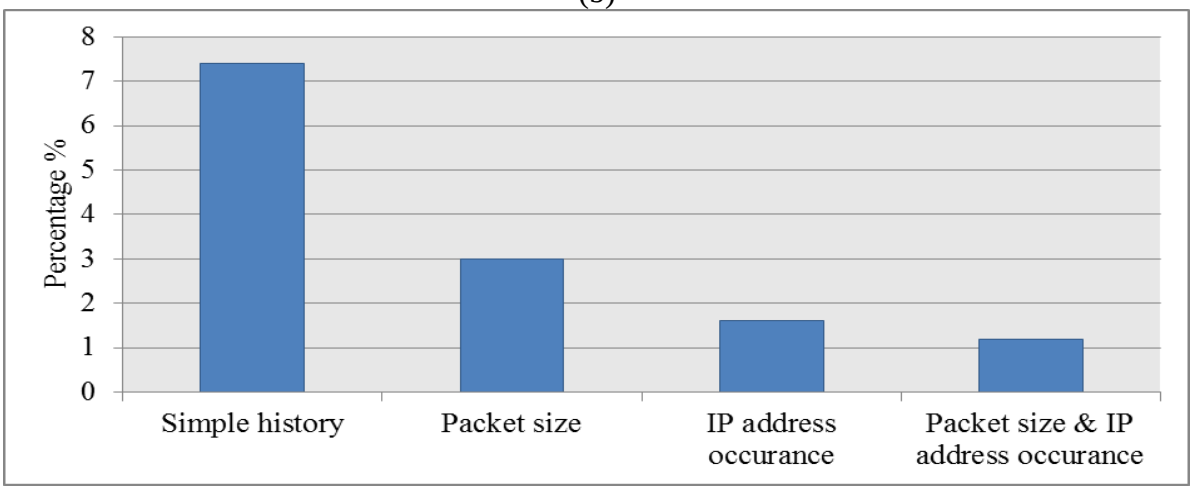
Figure. 3.4 shows the effectiveness of using signature to distinguish attack and normal traffic for different types of traffic. It is interesting to see that the false negative rate is around 1% for TCP, UDP and ICMP when we consider all selected features in our model. We evaluate the effectiveness of each selected feature to create accurate and robustness filtering as well. As shown in Figure. 3.4, the false negative rate decreases as we use more features. In addition, the experimental results determine that IP address frequency is the most effective feature for all traffic types. For TCP traffic, port number and packet size are the next important ones; however, for UDP traffic, packet size is more effective compared to the port number. Note that, in the figures simple history refers to creating history just based on frequency of IP addresses and establishing a three-way handshake for TCP traffic that was presented in [38]. It shows how the signature and score mechanism could be more effective for determining an accurate filter. In addition, it is robust against DDoS attacks as less than 1% of the attack traffic can bypass the filtering mechanism during the normal traffic conditions. Based on our results, we believe that our mechanism can be deployed in real networks.



(a)



(b)



(c)

Figure. 3.4. False negative rate: (a) TCP, (b) UDP, (c) ICMP

We have verified that the signature-based IP address history has good accuracy in terms of capturing legitimate and high confidence IP addresses. The next stage is to evaluate the efficiency of the Bloom filter that is calculated based on IP address history. As shown in Figure. 3.5, the size of Bloom filter required is around 30KB for IP address history based on all selected features; however, the size of the Bloom filter required increases to 73 KB if the IP address history is created just using packet size. Note that, a Bloom filter with 0.1 false positive rates will have a length of Bloom filter that is 5 times the number of inserted elements *n*, which are IP address of history in our case and 5 hash functions are used. According to Bloom filter equations it is possible to set a lower false positive rate; however, it required increasing the size of Bloom filters. Note that, we select the false positive rate as 0.1 and the length of Bloom filter as *5n*. Moreover, 5 hash functions require creating a Bloom filter with this condition.

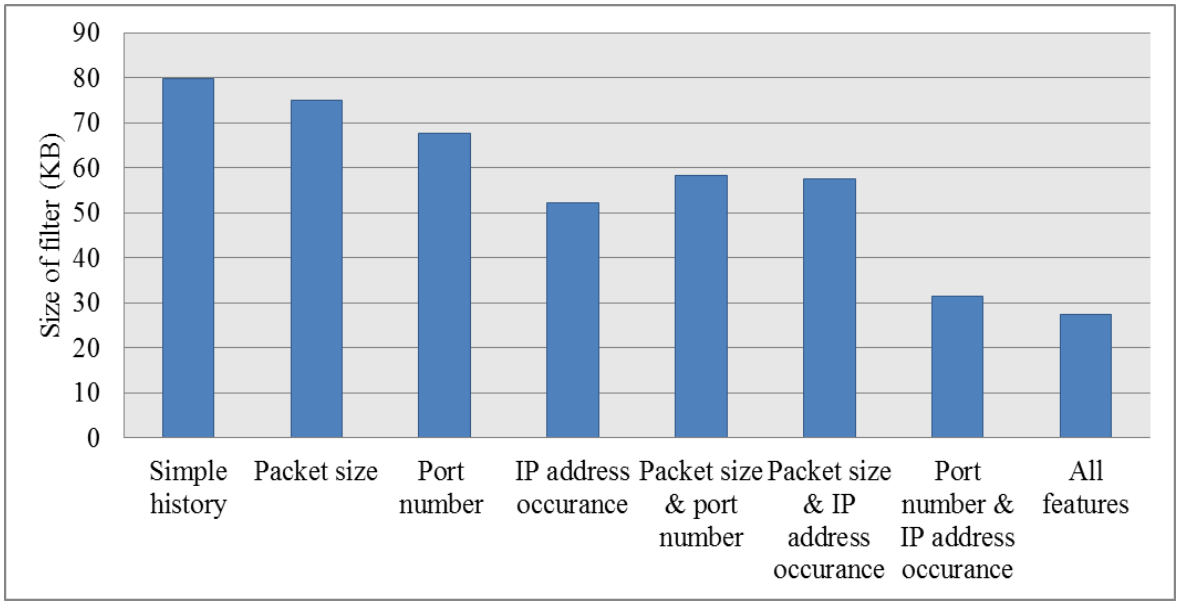
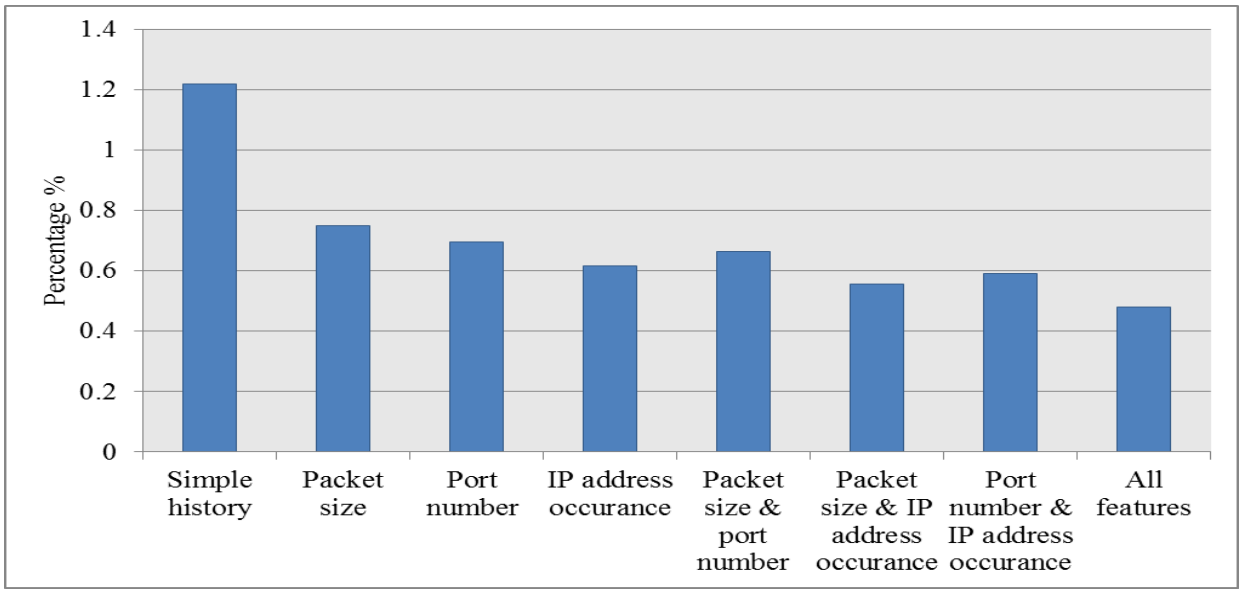


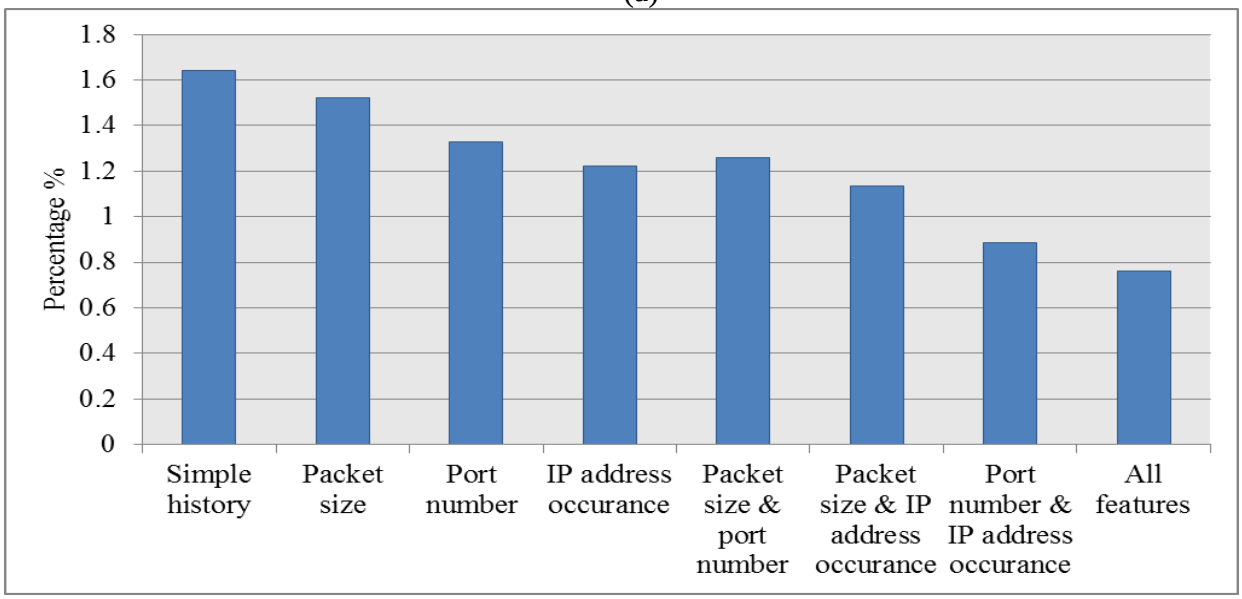
Figure. 3.5. Size of Bloom filter

When the DDoS attack is detected, the pre-built Bloom filter is transferred to the upstream routers to filter out the attack traffic. The Bloom filter is evaluated with 2 weeks testing DARPA 1998 dataset. As shown in Figure. 3.6, attack traffic detection rate is around 99.2% for TCP

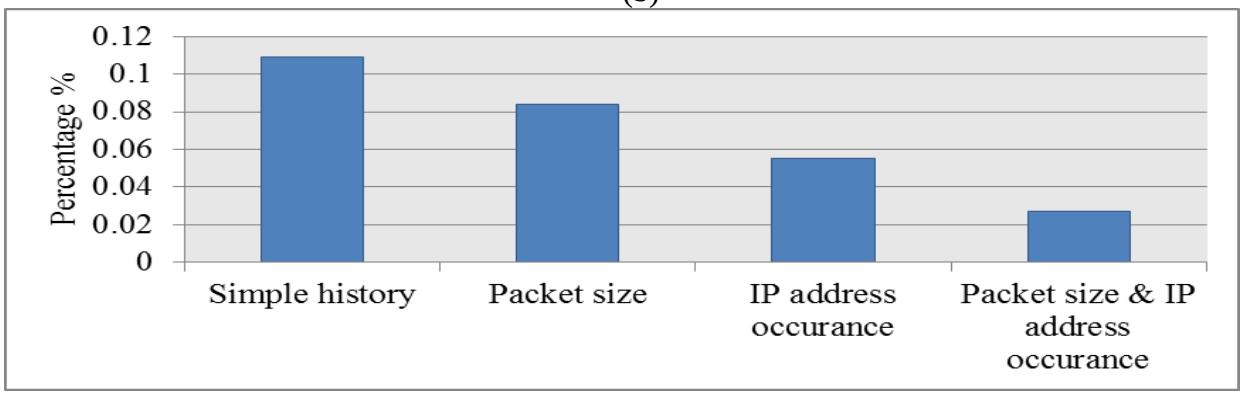
traffic and 99.6% and 99.8% for UDP and ICMP traffic if we consider all selected features to create the history. Note that, the attack traffic detection rate decreases if we consider fewer features and it goes down to 88% for simple history condition.



(a)



(b)

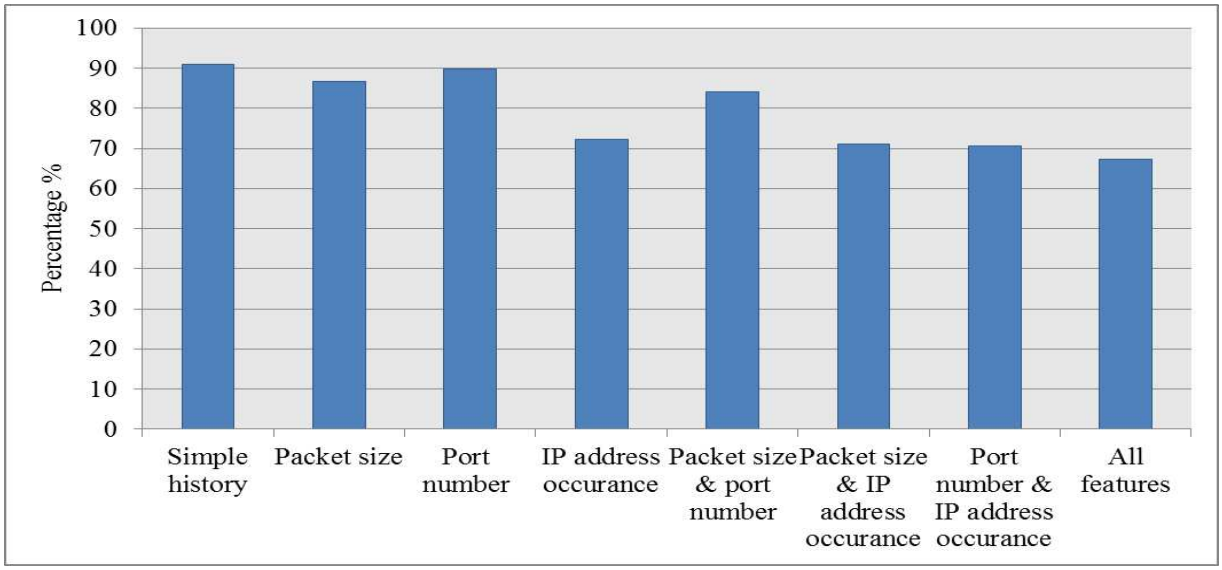


(c)

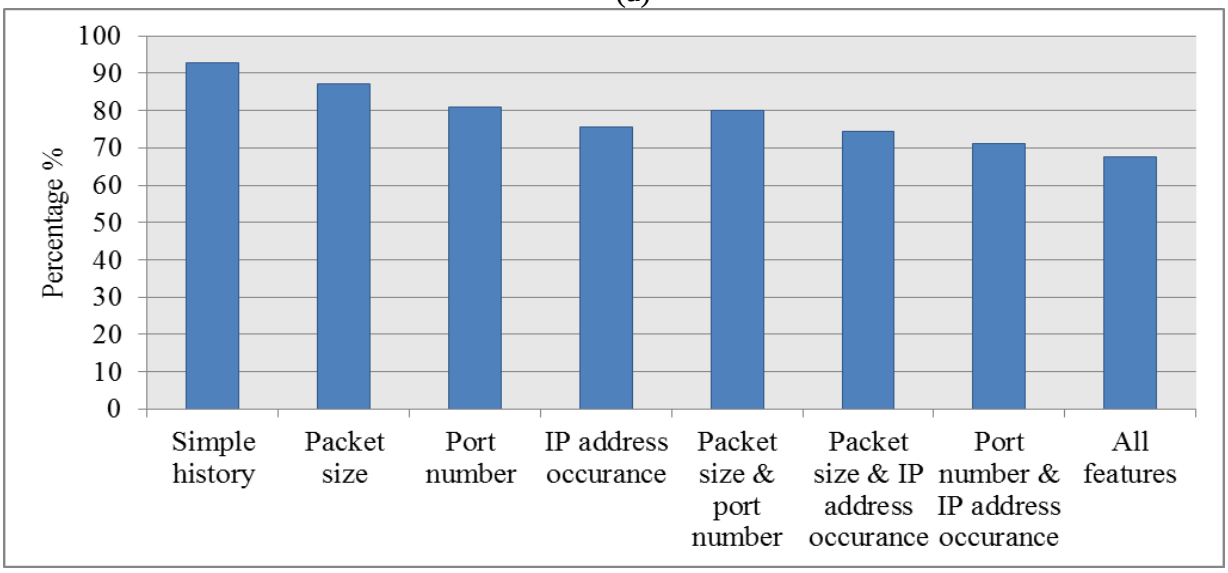
Figure. 3.6. Percentage of attack traffic that passes filter (a) TCP, (b) UDP, (c) ICMP

The other important parameter to consider is how many normal packets can pass through the Bloom filter. As shown in Figure. 3.7, normal traffic detection rate is around 70% for TCP and UDP traffic and it is more than 82% for ICMP when the history is generated based on all the selected features. This demonstrates that the signature based IP address performance is highly reliable for aborting malicious packets, but withholds some legitimate traffic. Normal traffic detection rate increases to around 75% to 85% if fewer features are used to generate IP address history, but the Attack traffic detection rate decreases at the same time. Thus, there is a tradeoff between accuracy of the Attack traffic detection rate and normal traffic detection rate.

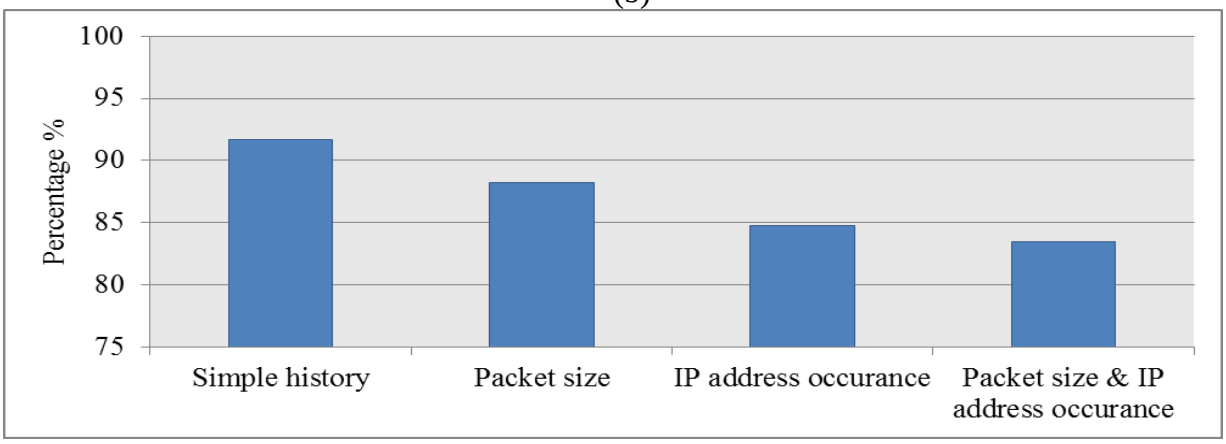
The evaluation part shows very good results in successful filtering of the bad traffic and permitting the good traffic during the attack time. The experiment result verifies that our signature mechanism can be deployed in real networks as it has addressed some critical drawbacks of the previous approaches.



(a)



(b)



(c)

Figure. 3.7. Normal traffic detection rate (a) TCP, (b) UDP, (c) ICMP

3.4.1.3 Impact of Threshold ν

Figure. 3.8 represents the effect of threshold ν on creating accurate and robust filtering, when the plot shows false positive and negative rates of IP address history based on all selected features. When ν increases, fewer malicious IP addresses are selected as legitimate ones during training period. Consequently, it reduces the false negative rate. On the other hand, increasing ν causes a rise in the false positive rate. It means with higher ν, more legitimate IP addresses are blocked by the Bloom filter. As a result, selecting ν has trade-off among history's accuracy and protection rate. Note that, increasing the value of ν provides higher protection rate and lower accuracy and impacts the size of the history as well. According to the results ν=0.36 was selected in our simulation as a reasonable trade-off among false positive and negative rate.

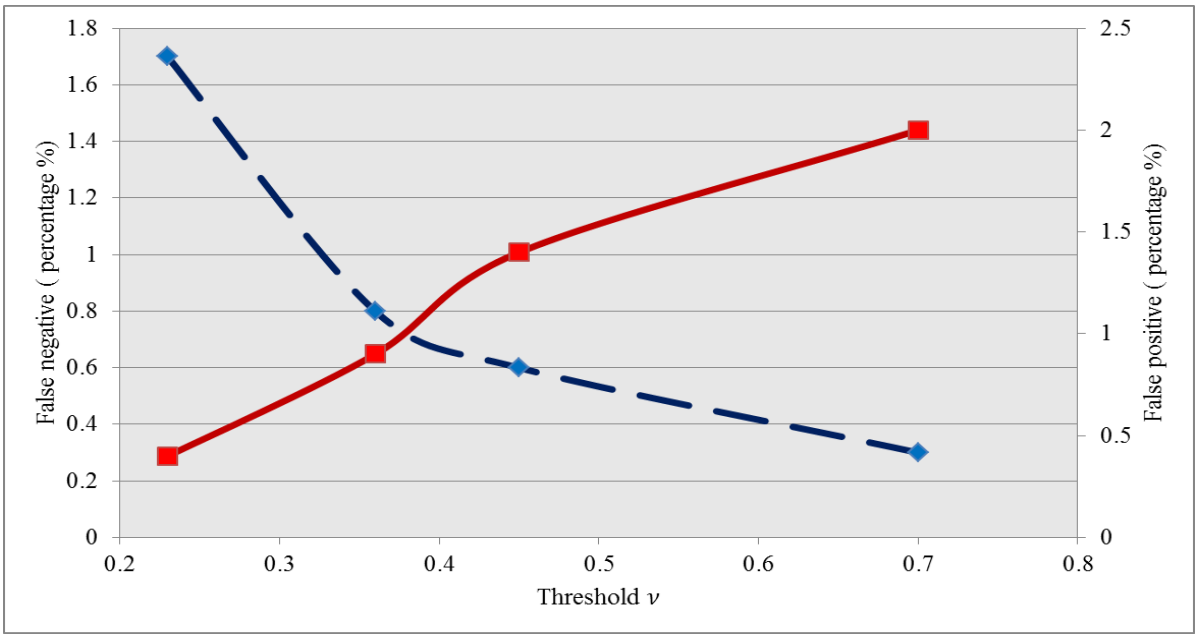


Figure. 3.8. False negative and false positive ratio with respect to different ν value

3.4.2 University of Auckland Dataset

In this section we demonstrate the implementation details of responsive defense mechanism based on traces collected from University of Auckland in New Zealand [6]. The packet trace is

one collected from the University of Auckland in New Zealand. The packet trace contains 6.5 week IP header trace taken with 155 Mbps Internet links [6]. The total uncompressed data is 180 GB and all IP addresses have been mapped into 10.*.*.* using one to one hash mapping for privacy. The Auckland traces record three type of IP traffic (TCP, UDP, ICMP).

We use "The CAIDA attack dataset 2007" to create an attack scenario in this experiment as the Auckland trace did not contain any attack data. Attack dataset contains approximately one hour of anonymized traffic traces from a DDoS attack on August 4, 2007 that contains 359,656,205 attack packets from 9,066 unique IP addresses [21].

3.4.2.1 Maintaining the IP Address History

As we mentioned earlier, the first step is creating the IP address history according to normal traffic at the end node. There are two difference mechanisms to maintain the IP address history: sliding window, history without updating. In the first one the IP addresses keep in the history with specific window length. Every day expired IP addresses are removed from the history and new IP addresses are added to it. In this part of experiment we set the length of window to be two weeks. That means for each day we update the history according to the all legitimate IP addresses that come to Auckland University during the two weeks before that day. In the second approach the history creates with specific length and keeps it without updating for whole of the next two weeks. We will show the difference in accuracy of history profiles created by these two mechanisms. It provides insight into how frequently the filters need to be updated.

3.4.2.2 Results

Signature-based IP address history is evaluated on the basis of these five parameters: *History Accuracy*, *Attack traffic detection rate*, *Normal traffic detection rate*, *False positive rate*, *Size of*

Bloom filter. Our experiment was conducted as follows: 1) create IP address History based on entire IP address; and 2) create IP address History based on first three octets of IP address. The two approaches provide different accuracy and Bloom filter size tradeoffs.

3.4.2.2.1 IP Address History Based on Entire IP Address

As discussed, we use a sliding window to keep IP address in the history. We build the IP addresses history using data trace from March 12, 2001 to March 25, 2001 and compare trace taken from March 26 to April 9, 2001 with the previous two weeks trace. As shown in Figure. 3.9 the history accuracy is about 82-92% for each day and this verifies that most IP addresses that appear in the network under normal conditions have previously appeared in the network and follow the signature of the end point. We also show in Figure. 3.10 effectiveness of using sliding window mechanism and updating history each day. It is interesting to see that the history accuracy drops when we consider the two week history without updating from 12 to 25 March 2001. The accuracy decrease as shown in Figure. 3.10. For instance, the history accuracy drops from 92% to less than 78% in April 7th.

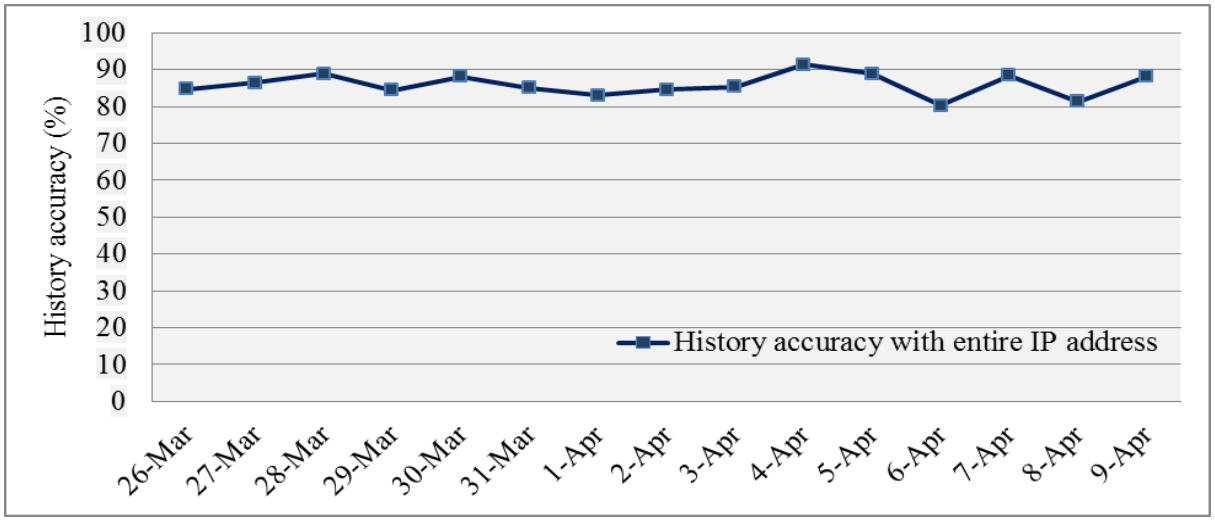


Figure. 3.9. History accuracy with entire IP address

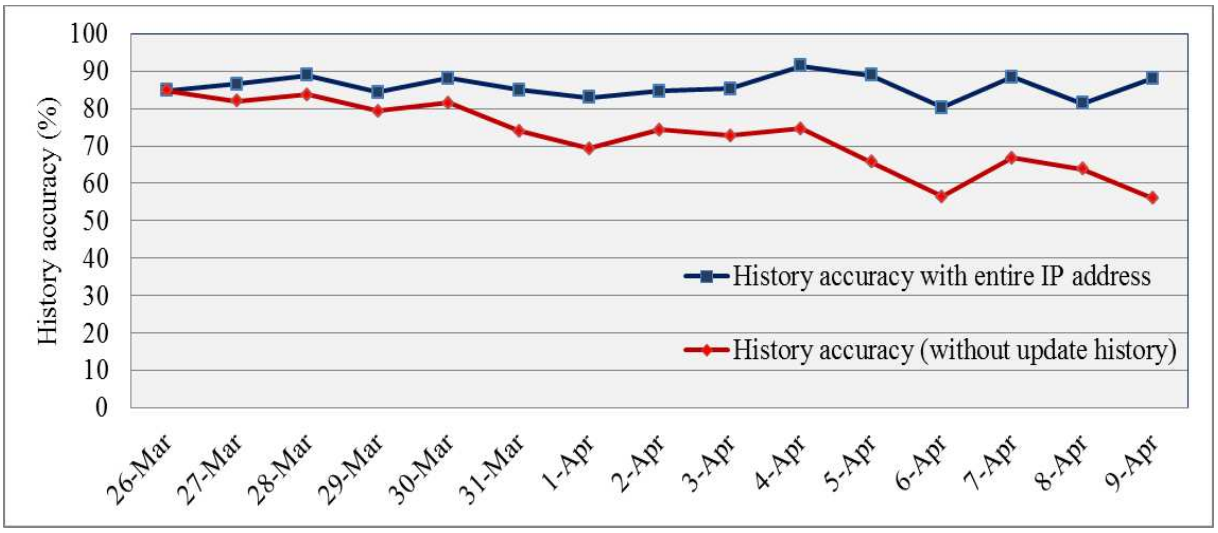


Figure. 3.10. History accuracy with sliding window technique and without updating technique

As we have verified, IP address history has good accuracy in terms capturing legitimate and high confidence IP addresses. The next stage is to evaluate efficiency of the Bloom filter structure that is created based on signature-based IP address history for each day. As shown in Table 3.2, the size of history that is created for each day contained around 420,000 IP addresses. A Bloom filter with 0.1 false positive rate according to Eq. 3.5 will have a length of Bloom filter 5 times the number of inserted elements, which are the number of IP addresses kept in the history. According to Bloom filter equations it is possible to set lower false positive rate; however, it is a tradeoff between size of Bloom filter and accepted false positive rate. We select the false positive rate as 0.1 and $5n$ for length of Bloom filter as reasonable error rate and Bloom filter size. Moreover, according to Eq. 3.5, 4 hash functions are required to provide a Bloom filter from the IP address history.

Assume that a DDoS attack is detected at Auckland university point and we need to prevent DDoS attack overloading traffic at this stage now. The pre-built Bloom filter for that day is transferred and applies in appropriate locations to filter out the attack traffic. To evaluate our Bloom filter efficiency we add real DDoS attack dataset from CAIDA 2007 [21] to the normal

background traffic trace. This type of DDoS attack consumes computing resource on the server and all of the bandwidth of the network connecting the server to the Internet. As shown in Figure. 3.11 attack traffic detection rate is around 95% for entire two week test. The other important parameter according to efficiency of our model is how many of the normal packets can pass the Bloom filter and do not block. As shown in Figure. 3.12 Normal traffic detection rate is more than 80% each day highlighting the Bloom filter performance and it is highly reliable responsive mechanism to abort the malicious packets and low damage to legitimate traffic. The other evaluation parameter is Size of Bloom filter. As shown in Table 3.2 size of Bloom filter is around 34KB that is an efficient size of message for transferring to the routers.

Table. 3.2. Number of unique IP address in the history and size of Bloom filter

	Entire IP addresses in history	**Size of Bloom filter (KB)**	**First three octets of IP addresses in history**	**Size of Bloom filter (KB)**
26-Mar	443,599	34.65	5,728	0.44
27-Mar	442,822	34.59	5,748	0.44
28-Mar	411,234	32.12	5,753	0.44
29-Mar	429,822	33.57	5,867	0.45
30-Mar	433,220	33.84	5,957	0.46
31-Mar	425,230	33.22	6,030	0.47
1-Apr	426,611	33.32	6,030	0.47
2-Apr	423,819	33.11	6,029	0.47
3-Apr	416,279	33.52	6,026	0.47
4-Apr	425,376	33.22	6,026	0.47
5-Apr	427,791	33.42	6,036	0.47
6-Apr	423,920	33.11	6,032	0.47
7-Apr	424,616	33.20	6,031	0.47
8-Apr	426,994	33.22	6,026	0.47
9-Apr	425,040	33.20	6,025	0.47

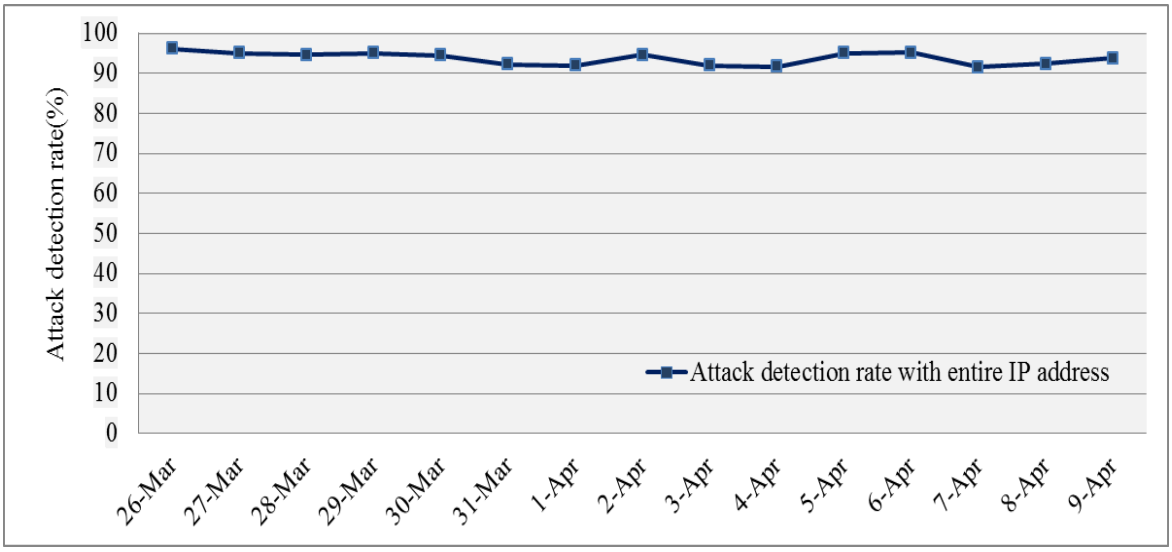


Figure. 3.11. Attack traffic detection rate

The Bloom filter testing shows substantial success in filtering of the bad traffic and passing the good traffic during the attack time. The experiment result verifies that our approach could be applicable in real network as it has addressed some critical drawbacks of the pervious approaches. In addition, in the next experiment we will show how a more compressed and efficient Bloom filter structure can provide further significant improvement.

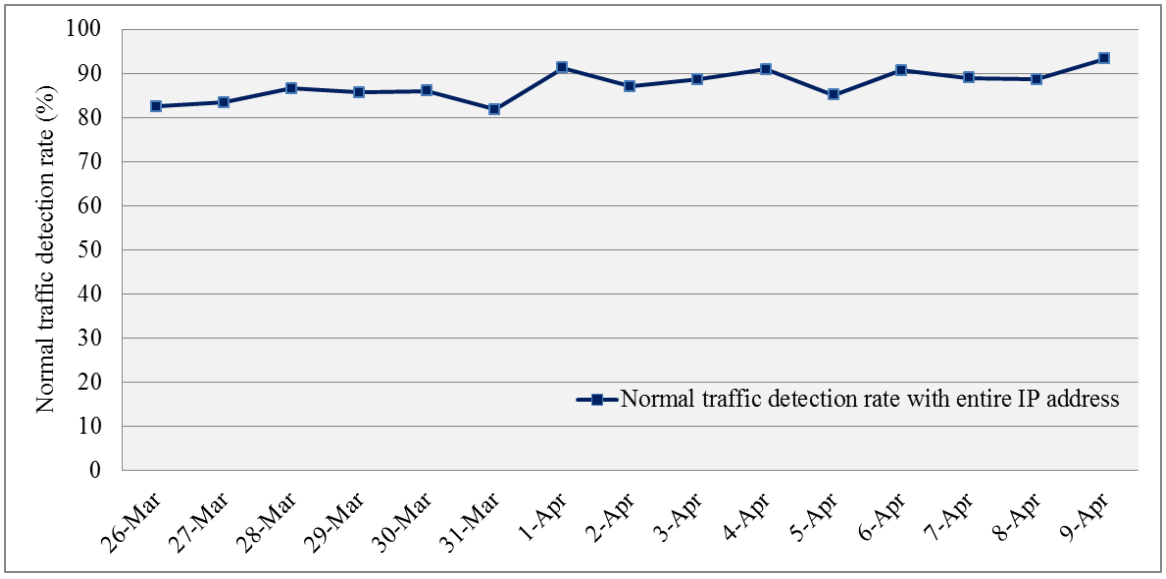


Figure. 3.12. Normal traffic detection rate

3.4.2.2.2 IP address History Based on the First Three Octets of IP Address

The main reason of this experiment is to provide smaller Bloom filters while maintaining the efficiency and accuracy of the IP address history. The idea to provide smaller but accurate Bloom filters by using first three octets of IP addresses. By applying this idea we achieve significant decrease of the number of unique IP addresses to keep in the history and consequently decrease the size of Bloom filter as shown in Table 3.2. The number of unique IP addresses that is kept in the history decreased to less than 0.01 of the previous IP address history. Furthermore the result has shown that at the same time the accuracy of the IP address history is increased by this change. That means the most of the normal traffic that comes to the specific node shared the first three octets of IP address. As shown in Figure. 3.13, the accuracy of IP address history improved to around 95% for most of the days during two weeks test traffic. Consequently Normal traffic detection rate increased and it is close to 90% as shown in Figure. 3.14.

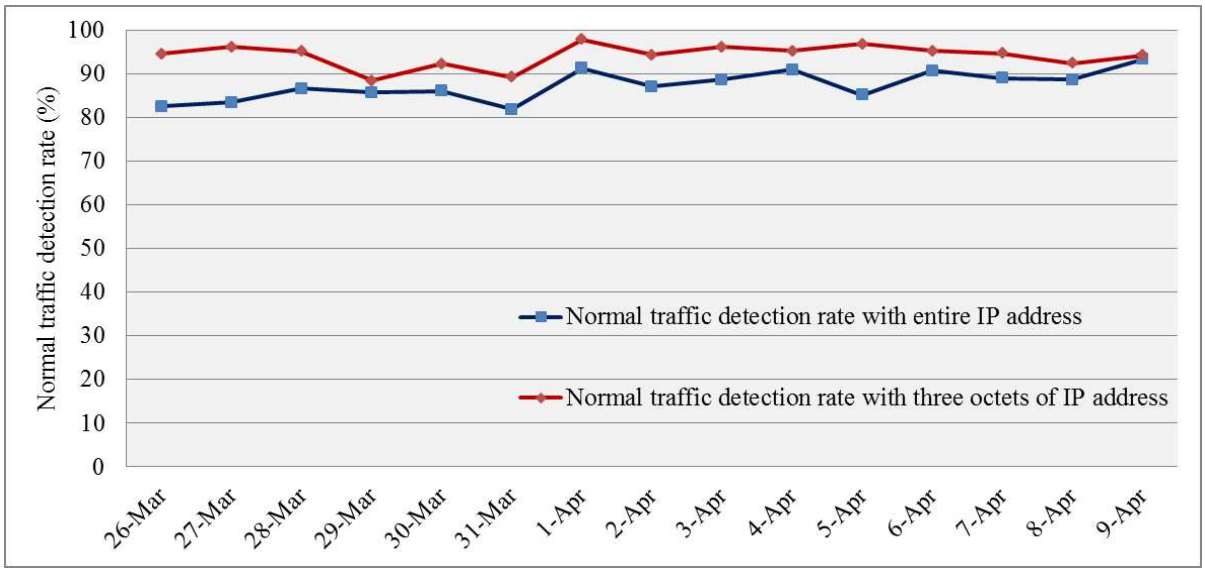


Figure 3.13. Normal traffic detection rate

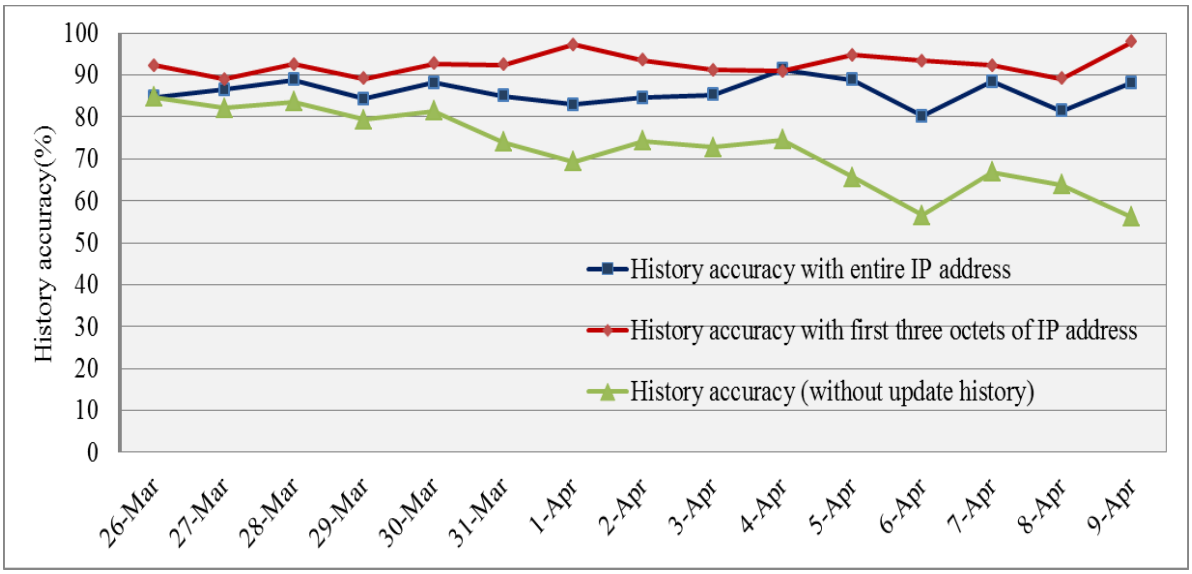


Figure. 3.14. History accuracy

The other important result of this experiment is shown in terms of attack traffic detection rate in Figure. 3.15. The result shows that the Bloom filter with first three octets can perform as well as the Bloom filter with complete IP addresses. In fact, in certain days it works better than the previous Bloom filter. According to Figure. 3.15 we observe almost 98% of attack traffic detection rate. Our experiment on the Auckland trace traffic shows that we can protect 95% of legitimate traffic with around 0.44KB of memory according to Figure. 3.13 and Table 3.2. It shows significant reduction in size of Bloom filter by using first three octets of IP address compare with using entire IP addresses. Moreover it can filter out about 98% of attack traffic for specific victim node.

The last result shows in Figure. 3.16 the false positive rate of Bloom filter that resulted in the first and second approaches. The acceptable false positive that we set was 0.1 and the results show the Bloom filter works correctly and in fact most of the time the false positive rate is less than maximum rate provided by to Eq. 3.5.

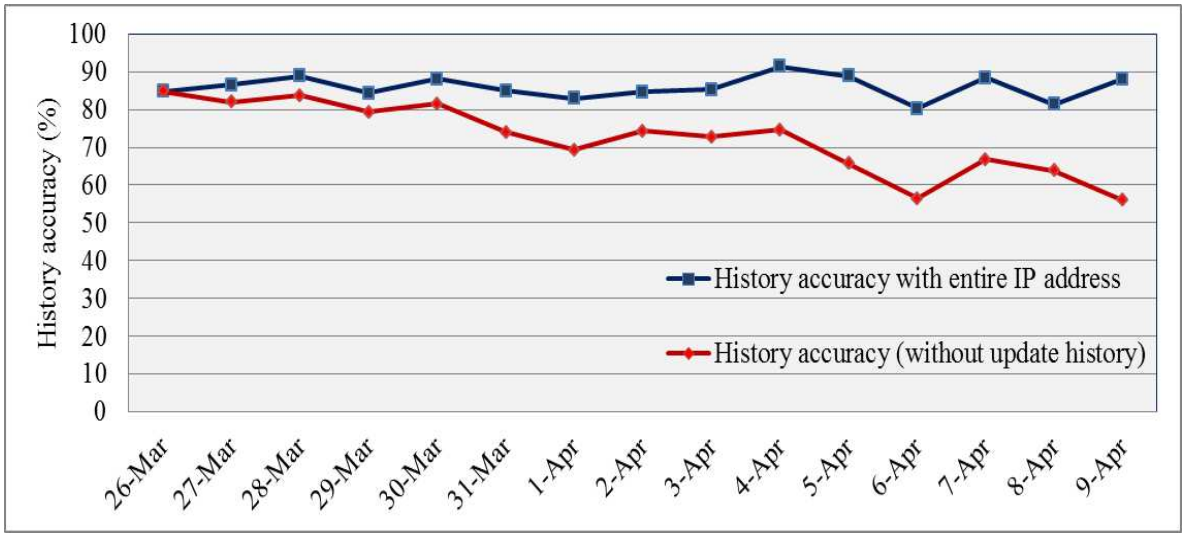


Figure. 3.15. Attack traffic detection rate

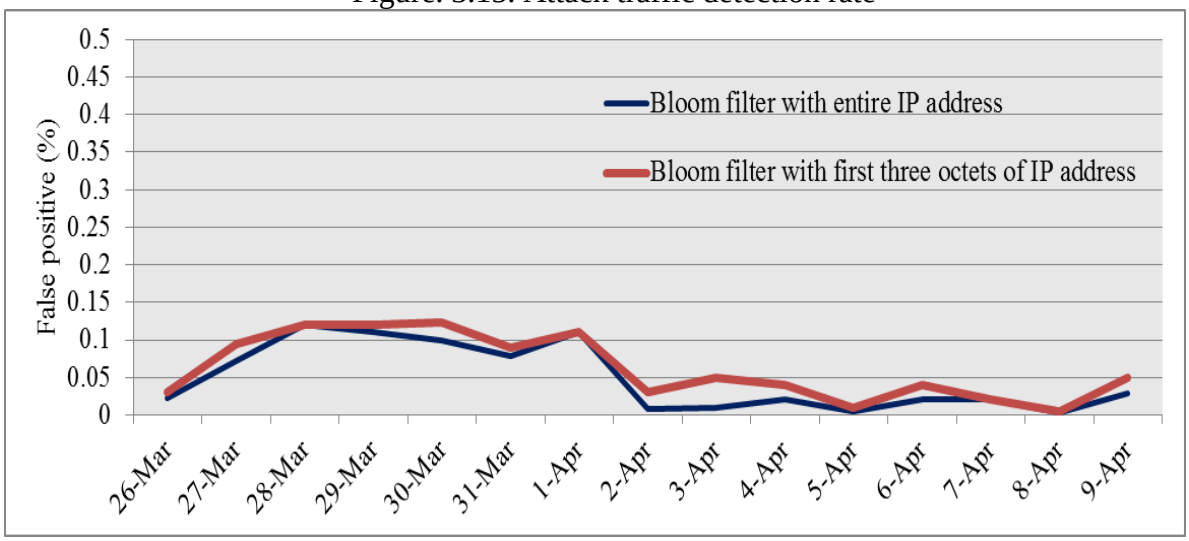


Figure. 3.16. False positive rate of Bloom filter

3.4.3 Colorado State University Dataset

In this section we demonstrate the effectiveness of our algorithm to characterize network traffic based on the trace collected at Colorado State University (CSU). The packet trace contains 4 week daily Argus files with flows on a 1Gb/s link from Feb 1st to Feb 28th 2015 where the total compressed data is about 20GB for each day [20].

3.4.3.1 Results

As we mentioned earlier, the first step is creating the IP address history according to the normal traffic is collected at the end node. IP addresses are kept in the history with specific window length. In this part of the experiment we set the length of window to be one week. That means for each day we create signature-based IP address history according to all IP addresses come to CSU during the last seven day period. We built the IP address history using data trace from Feb 1st to Feb 21st and compared with the trace taken from Feb 8th to Feb 28th. We will discuss the effectiveness of history accuracy for different window size in Section 3.4.3.1.1 as well.

The total traffic volumes are given in Figure. 3.17 corresponding to the total numbers of packets have sent to the CSU including internal traffic shows for each day. For instance, we can observe CSU received around 210 million packets on Feb 10th 2015. We excluded internal traffic from the total traffic to analyze both conditions as shown in Figure. 3.17. For instance, the total amount of traffic is very close for both Feb 10th and 11th but by considering only external traffic we can see the high portion of traffic that came to CSU on Feb 10th was external traffic compare with Feb 11th.

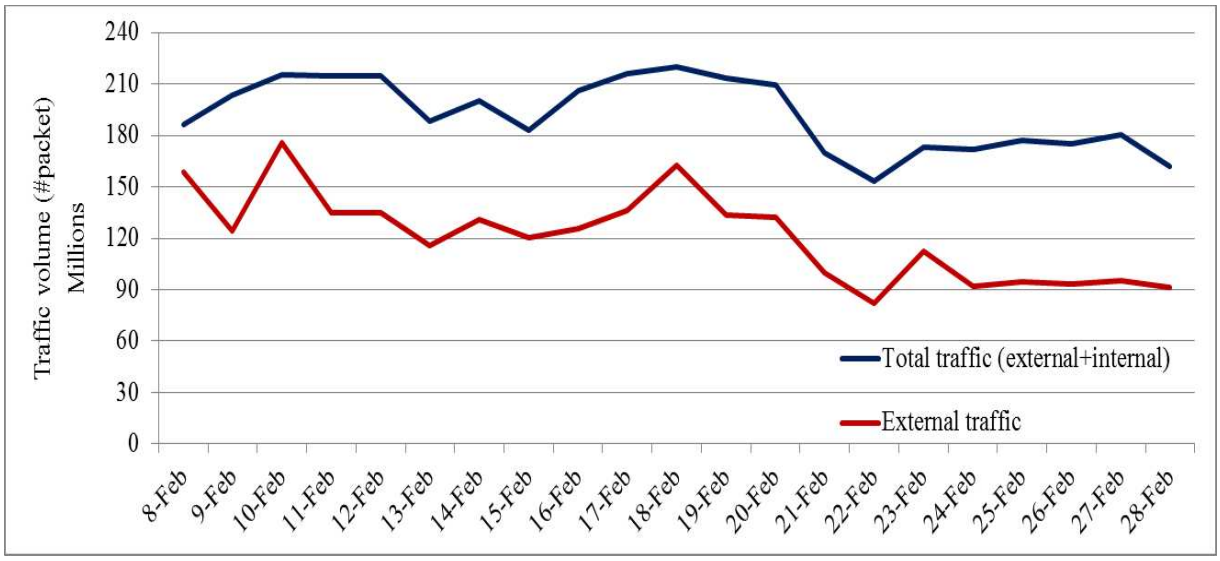


Figure. 3.17. Traffic volume

The first step is to create the signature-based IP address history from the daily traffic dataset according to generated signatures and overall score of selected features. The second step is to construct the Bloom filter based on the results from first step. In the first step, the signature of IP address's frequency, port number and size of the packet are created. As we discussed, for TCP traffic, IP addresses with successful TCP handshake are considered as the valid IP addresses. Moreover, for UDP traffic we ignore those incoming packets that just send a single packet during the day. This helps us to create a more reliable signature-based IP address history.

Figure. 3.18 shows the signature of IP address's frequency for the first week of CSU traffic. According to this figure, IP address with an occurrence rate above 500 can achieve the highest score as $C_{F1}(500) \geq 70\%$ and IP address with an occurrence rate below 20 get lowest score. From Figure. 3.18 we can also find that around 10% of IP addresses have a frequency above 2000 during a week and maximum frequency was for a few IP addresses with around 12000. The same analysis exists for port number and packet size signature as well.

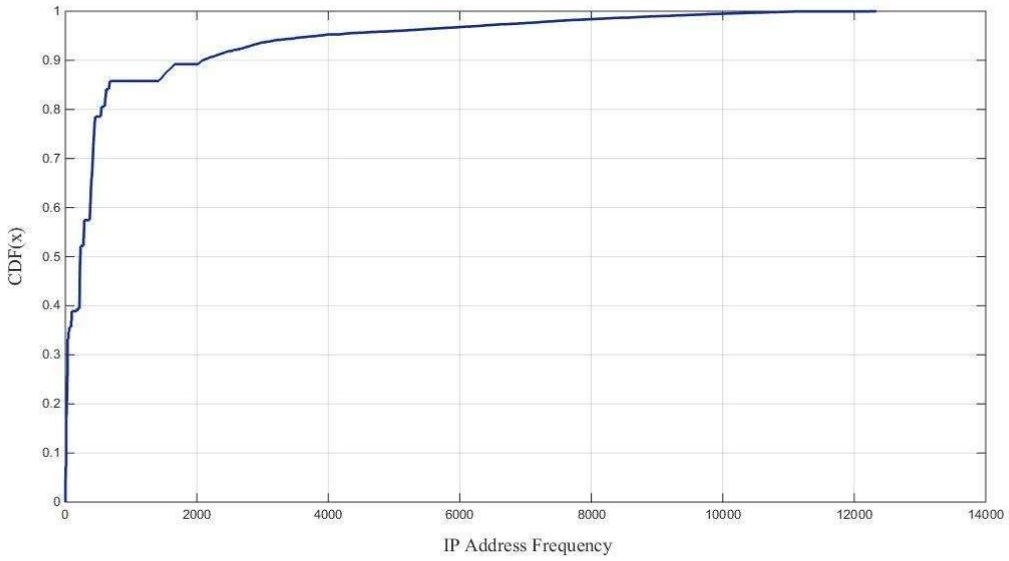


Figure. 3.18. Signatures of IP address frequency

In the next step, IP address history is created based on method described in Section 3.3.2. Our experiment was conducted based on the entire 4-octet IP address and also with the first three octets of IP addressed. The main contribution of this analysis is to provide smaller Bloom filter while maintaining the reasonable accuracy of IP address history. We compare the result of these two cases in each step.

As shown in Table 3.3 the number of unique IP addresses that were retained in the history is around 3,000,000 for each corresponding day. By accepting 10% false positive rate the size of Bloom filter would be between 1.4MB to 2.2MB for each day as it shows in Table 3.3 as well as in Figure. 3.19. According to the Bloom filter equations it is possible to set lower false positive rate; however, it is a tradeoff between the size of Bloom filter and accepted false positive rate. We accept the false positive rate as 0.1 and 5n for the length of Bloom filter as reasonable values for error rate and Bloom filter size, where n is the number of inserted element in Bloom filter, which are the number of IP addresses kept in the history.

Table. 3.3 Number of IP address in history

	Unique IP addresses in history	**Size of Bloom filter (MB)**	**Unique first three-octets of IP addresses in history**	**Size of Bloom filter (MB)**
8-Feb	3567433	2.22	1549329	0.96
9-Feb	2898726	1.81	1540879	0.96
10-Feb	2993910	1.87	1533835	0.95
11-Feb	3251485	2.03	1573583	0.98
12-Feb	3253085	2.03	1542783	0.96
13-Feb	3214944	2.0	1543112	0.96
14-Feb	3367433	2.1	1506123	0.94
15-Feb	3130830	1.95	1478050	0.92
16-Feb	2986955	1.86	1488892	0.93
17-Feb	3077742	1.92	1465505	0.91
18-Feb	3181456	1.98	1493287	0.93
19-Feb	3086945	1.92	1461922	0.91
20-Feb	3131923	1.95	1486039	0.92
21-Feb	3077169	1.92	1484376	0.92
22-Feb	2957518	1.84	1450000	0.9
23-Feb	3181719	1.98	1522926	0.95
24-Feb	3095821	1.93	1562730	0.97
25-Feb	3095821	1.93	1510018	0.94
26-Feb	2940737	1.83	1541806	0.96
27-Feb	2915739	1.82	1547367	0.96
28-Feb	2303440	1.43	1440457	0.9

As we mentioned, the idea is to provide a smaller Bloom filter while preserving the accuracy of the history. By creating the history with the first three octets of IP address we achieve significant reduction for the number of unique IP address to be kept in the history and consequently it decreases the size of Bloom filter as shown in Table 3.3 and Figure. 3.19. The number of unique addresses in history drop down to around 1,500,000 by using the first three octets of IP address and it shows around 50% reduction of total IP addresses need to keep in the history. As a result, the size of the Bloom filter reduces to around 0.9 MB for each corresponding

day as shown in Figure. 3.19. Furthermore, the result shows that at the same time the accuracy of IP address history increases due to this change. We will discuss about the history accuracy in the next step.

In particular, it is important to minimize the size of filters such that storage and processing cost are minimizing as we address in this part of analysis. In Chapter 4 we derive a new structure which we refer to as the Compacted Bloom Filter (CmBF) that use less storage with the same functionality as the standard Bloom filter, by introducing the false negative in membership query. In short, we can tradeoff between false positive rate, false negative rate and memory reduction by using CmBF data structure. According to that structure the size of Bloom filter can be reduced between 20% to 60% with almost similar false positive values compared with those for standard Bloom filter, but accepting 2% to 20% false negative. As shown in Figure. 3.19, the size of CmBF when entire 4-octet IP address can be in the range of 0.8 MB to 1.8 MB. For the IP address history based on the first three octets of IP address the size of CmBF is between 0.4 MB to 0.8 MB as well. As the result show the size of Bloom filter for each day can vary based on the scheme selected. For instance the maximum size of Bloom filter for Feb 8th is 1800 KB by using the standard Bloom filter and the 4-octet IP address whereas the minimum size is 400 KB by using CmBF and first three octets of IP address.

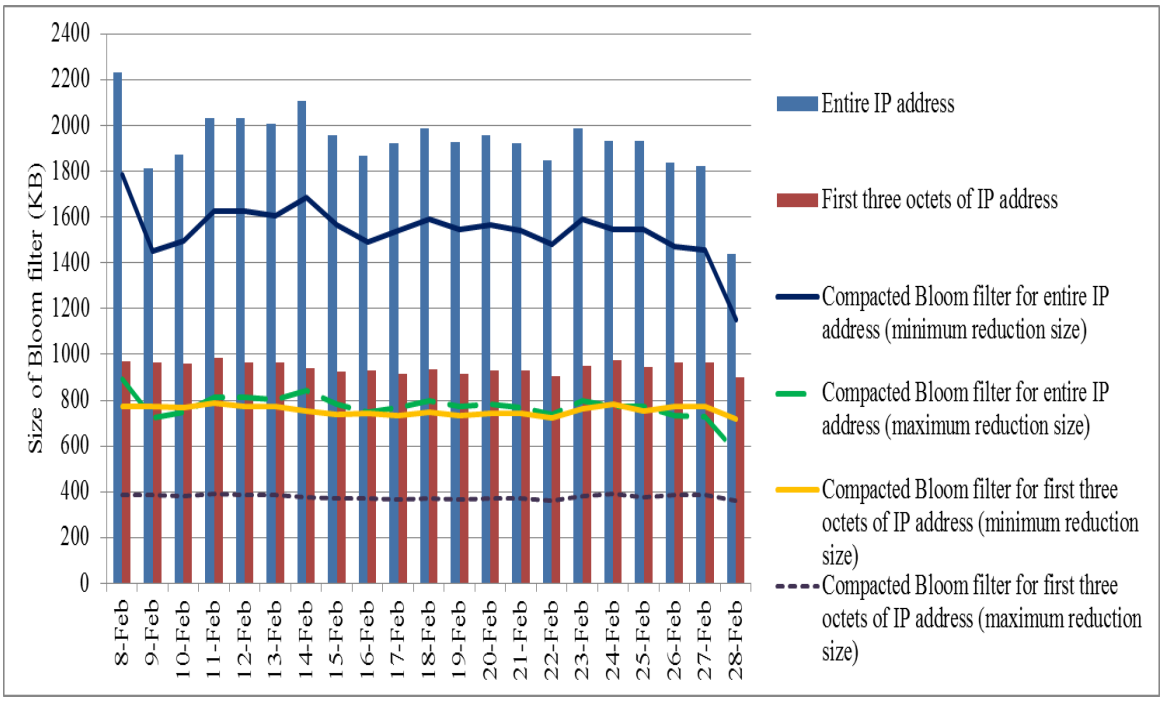


Figure. 3.19. Size of Bloom filter

The next stage is to evaluate the accuracy of the history, i.e., the percentage of traffic for each day that appeared in the history. As shown in Figure. 3.20 the history accuracy is about 60% to 80% for each day and this verifies that most of the IP addresses that appear in the CSU network under normal conditions have previously visited and follow the signature of each day's network traffic. We can see the highest history accuracy is for Feb 21st with 80% and the lowest one is for Feb 12th with 60%. Table 3.4 shows the number of unique IP address of packets visiting CSU and the number of IP address that match with history for each corresponding day. For instance, on Feb 8th the total number of unique IP addresses on packets coming to CSU is 2,709,558 and 1,790,443 of those IP addresses match with the history as of Feb 8th. We also show the effectiveness of using the first three octets of IP address in Figure. 3.20. In this case the accuracy of IP address history improves to around 75% for most of the days while the size of Bloom filter reduces to 50% as we discussed with respect to Figure. 3.19.

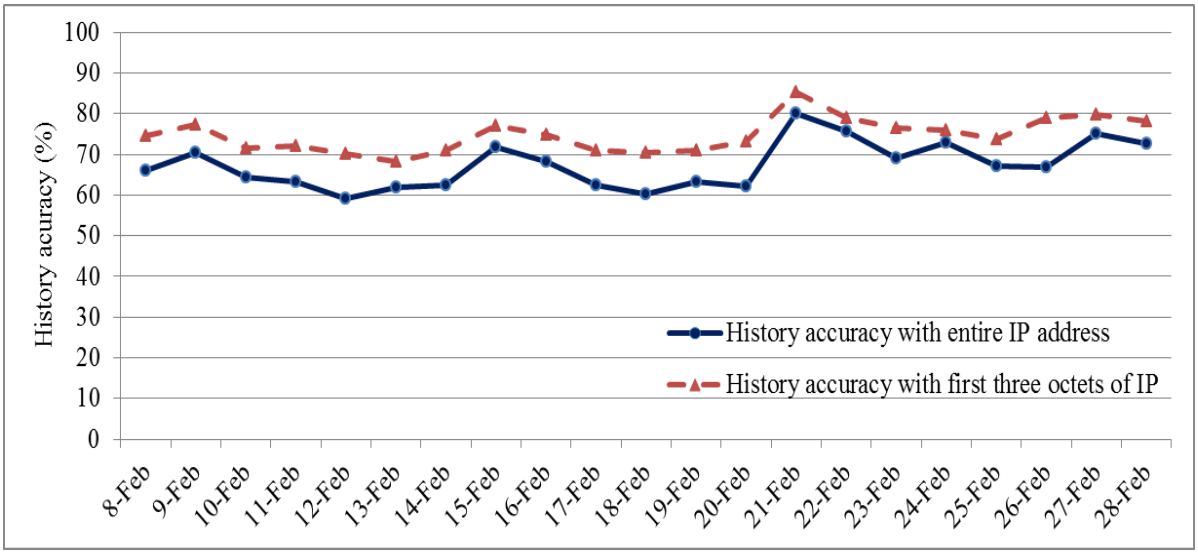


Figure. 3.20. History accuracy

Table. 3.4 Number of unique IP addresses of packets arriving to CSU and their match with history

	8-Feb	9-Feb	10-Feb	11-Feb	12-Feb	13-Feb	14-Feb	15-Feb	16-Feb	17-Feb
IP Match	1790443	1627396	1898679	1981443	1957757	1984888	1995127	1750964	1775144	1998038
Total IP in History	2709558	2309466	3009246	3129533	3316582	3201160	3191604	2440758	2598016	3206285

18-Feb	19-Feb	20-Feb	21-Feb	22-Feb	23-Feb	24-Feb	25-Feb	26-Feb	27-Feb	28-Feb
1989359	1808770	1804841	1684604	1335560	1362394	1424731	1654167	1412483	1652258	1395530
3309144	2864580	2900503	2370058	1767686	1698466	1957384	2459654	2115752	2197652	1922479

The other important parameter to evaluate is how much of the traffic volumes can pass through the Bloom filter and reach the end nodes. As shown in Figure. 3.21 normal traffic detection rate is more than 68% up to 82%. This demonstrates that the performance of signature-based IP address history is highly reliable for identify normal traffic and withholding only little

legitimate traffic. Normal traffic detection rate increase to around 80% to 90% if the first three octets of IP addresses is used to generate IP address history.

In this part we investigate the effectiveness of history accuracy and measure how many of packet traffic can pass the Bloom filter. The results are shown in Figure. 3.22. The x-axis is the traffic volume based on number of packets. From this figure we observe the total number of traffic received is varying from 160 Million to 220 Million packets in each day. We denote the external traffic also increase suddenly on Feb 10th and Feb 16th where the portion of normal traffic can pass the history doesn't change too much. It means for those days part of the incoming traffic to CSU doesn't appear before and it was not include in the history. For almost all the other days the normal traffic that can pass the history has similar behavior. Furthermore, the result shows the signature-based IP address history with the first three octets can perform better than the history with entire IP address as expected. There is one clear deduction which is by using first three octet of IP address history the normal traffic detection rate increase; however, there is tradeoff between protection rate and normal traffic detection rate. When we are using the first three octet of IP address there is some possibility that malicious start the attack from the point with similar first three octets of IP address that exist in the history and the signature-based IP address history couldn't filter them. Therefore the protection rate reduces by using three octets of IP address. In the other side by using first three octet of IP address the size of Bloom filter reduces and normal traffic detection rate increases. Therefore, selecting appropriate approach is based on accepted size of Bloom filter, history accuracy and protection rate. In general, using the first three octets of IP address gives the better history accuracy, normal detection rate and it reduces the size of Bloom filter where protection rate decrease at this point.

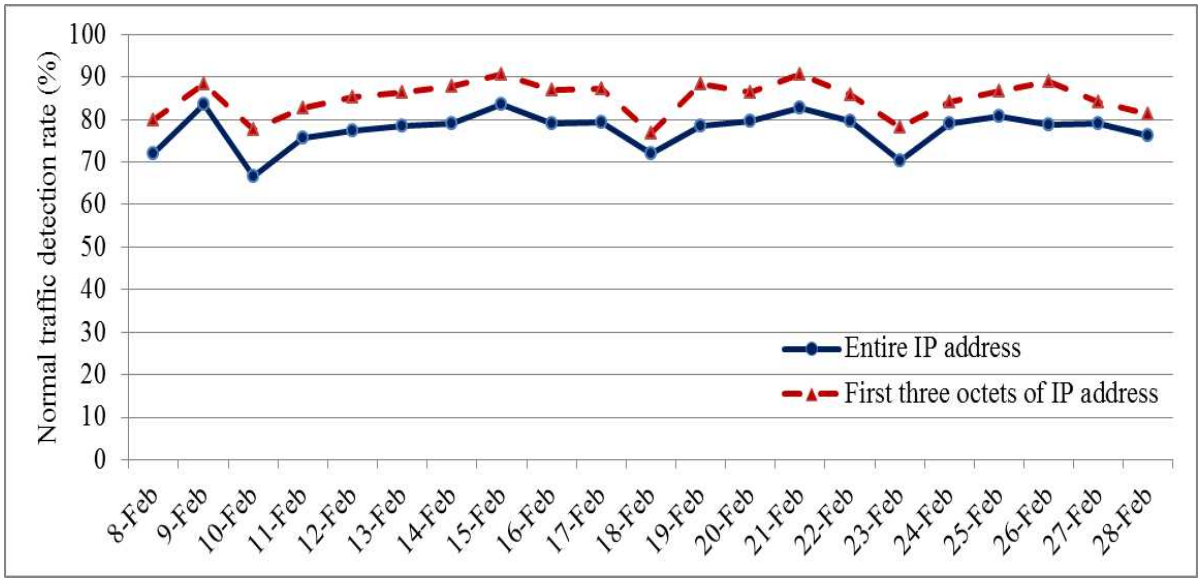


Figure. 3.21. Normal traffic detection rate

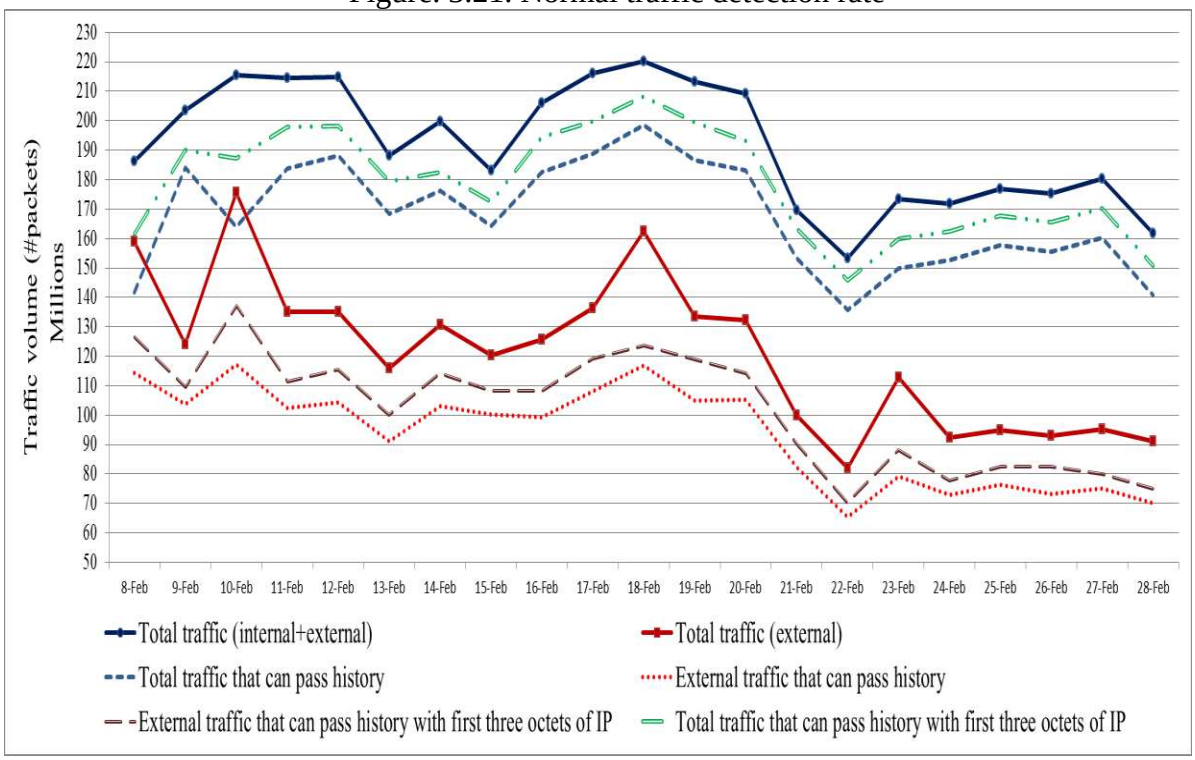


Figure. 3.22. Traffic volume and number of packets can pass history

3.4.3.1.1 Impact of Window Size

In the last part, we studied the effect of window size to determine accurate IP address history. We create history for four different window size and measure history accuracy according to them. The window size is set to 14, 7, 10 and 3 days respectively for creating the history and then we

calculate history accuracy for days from Feb 15th to Feb 28th as shown in Figure. 3.23. The number of unique IP addresses that appeared in the corresponding history also show in Table 3.5. Figure. 3.23 shows the history accuracy improves as long as we increase the length of window size from 3 to 14 days; however, the interesting observation is that history accuracy doesn't change significantly after 10 days. It means most of the legitimate IP addresses have appeared in the history visited CSU less than 10 days ago and very few IP addresses have made more than 10 days prior appearance. The other observation can be notice in Figure. 3.23 that the larger gap is between window size 3 and 7 days. The history accuracy is around 50% for 3 days window size while this value improves to 65% for 7 days. In fact this experience shows the 3 days window size is a short period to create a history with high accuracy.

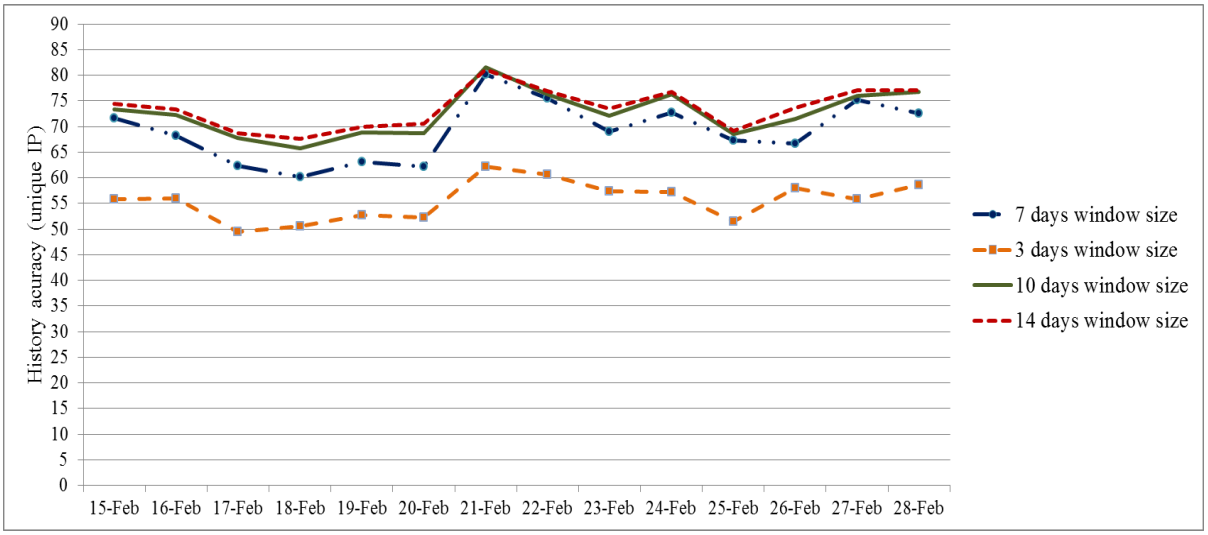


Figure. 3.23. History accuracy for different window sizes

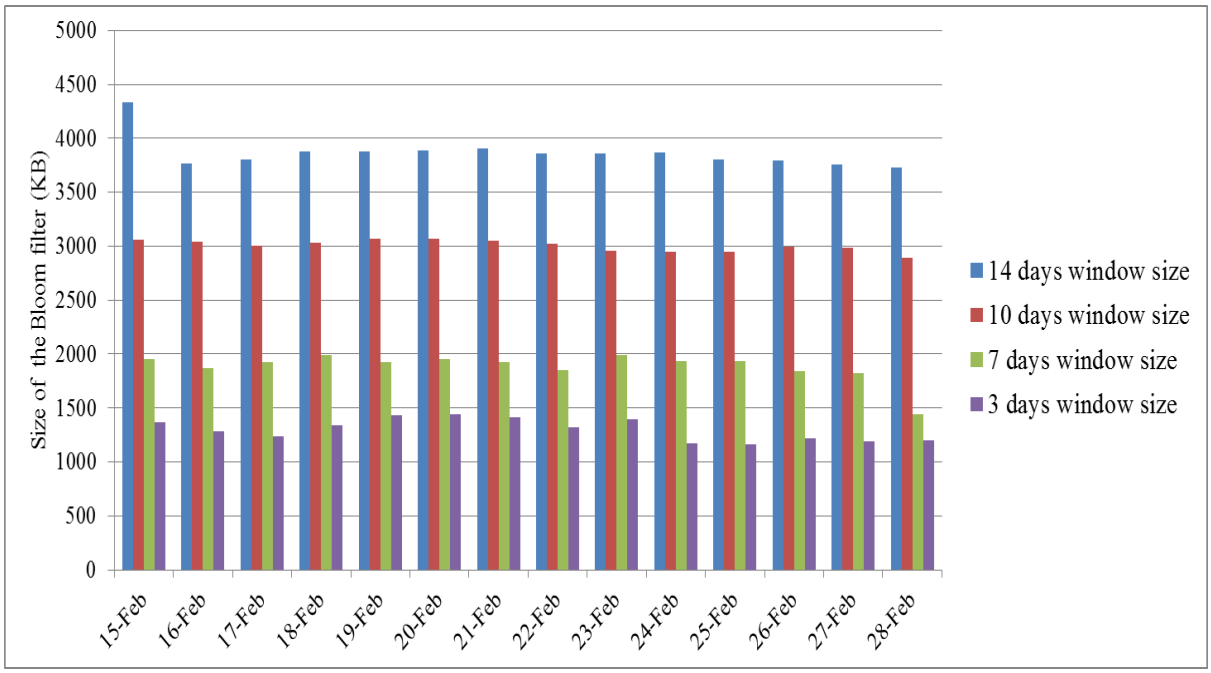


Figure. 3.24. Size of the Bloom filter for different window size

Table. 3.5 Number of unique IP addresses in history with different window size

	15-Feb	**16-Feb**	**17-Feb**	**18-Feb**	**19-Feb**	**20-Feb**	**21-Feb**	**22-Feb**
14 days window size	6930826	6019868	6080193	6206424	6198630	6224286	6244686	6178677
10 days window size	4889103	4858096	4803532	4853005	4910655	4911569	4887286	4832357
7 days window size	3130830	2986955	3077742	3181456	3086945	3131923	3077169	2957518
3 days window size	2189643	2047126	1987207	2141480	2289896	2308403	2259784	2118029

23-Feb	**24-Feb**	**25-Feb**	**26-Feb**	**27-Feb**	**28-Feb**
6166982	6192012	6091122	6069075	6007085	5969252
4735330	4714798	4717731	4785940	4775288	4628090
3181719	3095821	3095821	2940737	2915739	2303440
2239626	1874455	1865488	1944892	1911688	1917622

The other important parameter to evaluate history accuracy is size of Bloom filter. As shown in Table 3.5 and Figure. 3.24 as long as increasing the window size, size of history as well as history accuracy increase. This means that there is tradeoff between history accuracy and size of history. According to Figure. 3.24, size of history increase around 0.7MB in average from 10 days to 14 days window size; however, as shown in Figure. 3.23 history accuracy is almost similar. Therefore between 10 days and 14 days window size, 10 days window size would be appropriate selection. Furthermore, from Figure. 3.24 we can also find that size of Bloom filter reduce around 1MB in average when the window size changes from 10 days to 7 days where, the history accuracy only improve less than 5% by considering 10 days window size. As the result, 7 days window size can be the best choose where it can provide good history accuracy with appropriate size of Bloom filter.

In our work, we show very good results in successful characterizing the network traffic and preserving good traffic with appropriate filtering's size. The experiment results verify that our signature-based mechanism can be deployed in real networks as it has addressed some critical drawback of the previous approaches as well.

3.5 Conclusion

In this chapter we introduced a signature-based filtering mechanism that distinguishes attacks from normal traffic. We have demonstrated how to form a history based filter that takes into account a rich set of header fields of traffic and characterize network traffic. We then presented a filtering mechanism based on Bloom filter structure that can be rapidly distributed to responsive points closer to attack sources, Furthermore; we have shown the effectiveness of filtering structure with the DARPA dataset, University of Auckland and Colorado State University

dataset. Our experiment indicates that our filtering model can protect the victim node from ~95% attack traffic while allowing ~70% of legitimate traffic with appropriate size of Bloom filter. In addition, according to DARPA dataset, it shows that the filtering mechanism can detect 99% of attacks that want to bypass the filtering during the normal traffic. In contrast to existing DDoS attack detection techniques, our scheme is more reliable and accurate, and also has a lower overhead.

CHAPTER 4

THE COMPACTED BLOOM FILTER

4.1 Introduction

This chapter focuses on designing an efficient data structure for filtering attack traffic. A Bloom filter is a space-efficient probabilistic data structure that is used in many domains including networking applications to test for set memberships. Such applications often require passing bloom filters using messages. Consequently, it is important to minimize the size of the filters such that storage and processing costs are minimized. In this chapter, we introduce a novel data structure, which we refer to as the Compacted Bloom Filter (CmBF) that improves performance, uses less storage, and provides the same functionality as a standard Bloom filter for applications, such as IP traceback, web caching, and peer-to-peer networks. Moreover, CmBF works well for endpoints with limited memory. However, unlike the standard Bloom filter, our data structure limits false positives significantly at the expense of some false negatives in membership queries. We derive expressions for the false positive and false negative rates. We conduct simulations that validate the derived expressions and explore the tradeoffs of this data structure.

Standard Bloom filters [42] provide space-efficient storage of sets, at the cost of probable false positives to support membership queries [43]. Although a Bloom filter may yield false positives, which occurs when an external element is recognized as a member of the set, it provides a compact structure that can be configured for sufficient accuracy. In order to achieve a false positive probability of 1%, 10 bits of storage per element is required. Bloom filters have been used in many different distributed applications [44] such as IP traceback [45] [46], web

caching [47] and peer-to-peer networks [48] [49] [50]. Although a standard Bloom filter construct is space efficient for simple membership queries, they are inadequate when the applications need to meet a specific false positive rate and the endpoints have limited storage capacity. Our work is motivated by applications that must transmit Bloom filters over the network and/or endpoint machines have restriction on available memory to meet specific false positive rate.

In recent years, the popularity of Bloom filters has grown and they are now being used in many different areas including peer-to-peer systems, web caches, database system, spell checkers [51] [52] and networking [53]. Consequently, in the past decades lots of research efforts focused on improving Bloom filters. Some variants of the Bloom filter include Counting Bloom filter [47] [54], d-left Counting Bloom filter [55], [56] Spectral Bloom filter [57], Generalized Bloom Filter (GBF) [53], L-priorities Bloom filter [58] and Compressed Bloom Filter (CBF) [41].

The CBF [54] reduces the transmission size of the Bloom filter. However, the tradeoffs are: (1) the increased processing required for compression and decompression, and (2) the increased memory required at the end points. In this work, decompression must be done at the receiving end to achieve the functionality of the standard Bloom filter. The memory savings occur during transmission, but not at the end points. In our work, we achieve memory savings at the end points and also during transmission time. We present a new variant of the Bloom filter, which we refer to as the Compacted Bloom filter (CmBF), which has the same functionality as the standard Bloom filter but requires less memory. For the same fraction of false positives, the CmBF generally requires half the bits over the standard Bloom filter. This memory saving is achieved at the expense of increasing the number of false negatives when CmBFs are used instead of the

standard Bloom filter. A false negative occurs when an inserted element is not recognized by a membership query. Later we demonstrate how the false negative rate depends on the size of the memory and the false positive rates. CmBF is also very simple to implement, which makes it very useful in practice, much like the standard Bloom filter.

The rest of this chapter is organized as follows. Section 4.2 summarizes background material on the Bloom filter. CmBF is described in section 4.3, along with a theoretical analysis. In section 4.4 we provide experimental results that corroborate with our theoretical analysis. In addition, we also present the tradeoffs between false positive rates, false negative rates and memory savings. Finally, section 4.5 concludes the chapter.

4.2 Bloom Filters

A Bloom filter is used to represent a set S of n elements, where $S = \{s_1, s_2, \ldots, s_n\}$. It is implemented by a single array of m bits and k independent hash functions $h_1, h_2, \ldots, h_k$. Each hash function uniformly maps an element of the set S to a value in the range $\{0, \ldots, m\text{-}1\}$. The average number of bits used to represent a single element is $k_0 = m/n$. To construct the Bloom filter, first all m bits are initialized to zero. Then for each element $x \in S$, bit $h_i(x)$ is set to one for $i=1, 2, \ldots, k$. The same bit will be set to one several times if different hash functions map to the same element in the range $\{0, \ldots, m\text{-}1\}$. Figure. 4.1 depicts how an element is inserted into a Bloom filter. In order to query whether an element y belongs to S, we check whether the bits of the array corresponding to the hash functions $h_i(y)$, where $i=1, 2, \ldots, k$ are 1 or not. If at least one bit is 0 then $y \notin S$. Otherwise, $y \in S$ with some probability. This test can result in a false positive. False positive occurs when the bits $h_i(y)$ $i=1, 2, \ldots, k$ are all previously set by other inserted

elements. The probability of a false positive for an element $y \notin S$ is calculated as follows [41]. The probability of a given bit is set to 1 in the array after inserting n elements is:

$$p = 1 - (1 - \frac{1}{m})^{kn} \approx 1 - e^{-k\frac{n}{m}} \tag{4.1}$$

The false positive rate f_p is the probability of finding a bit set to 1 for each of the k positions for non-member elements. It is given by the following equation:

$$f_p = (p)^k \approx (1 - e^{-k\frac{n}{m}})^k \tag{4.2}$$

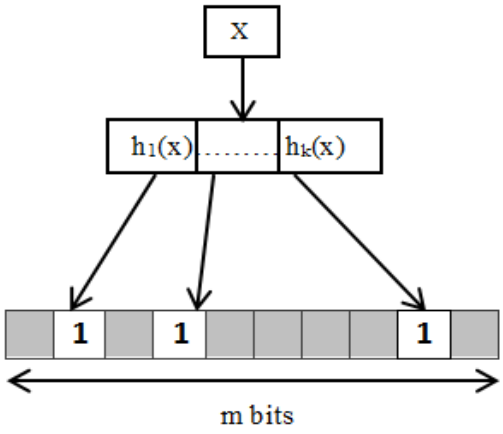


Figure. 4.1. An example of insertion of an element into a Bloom filter

The interesting observation is that increasing the ratio m/n reduces the false positive rate. Additionally, there is an important tradeoff between the number of hash functions k and the false positive rate. Increasing the number of hash functions k leads to higher false positive rates and at the same time the probability of finding every indicated bit is 1 decrease with higher values of k. The optimal value for k that minimizes f_p is found by differentiating f_p with respects to k. The only solution is $p=1/2$ which implies that the minimum false positive rate is achieved when approximately half of the bits equals 1. In this condition $k=m/n\ ln2$ and f_p is $(.5)^k \approx (.6185)^{m/n}$. From this equation, the minimum number of bits to achieve a false positive rate f_p is given by $m = -0.208n \log f_p$.

One problem with the Bloom filter is that its false positive rate strongly depends on the fill ratio, which is the number of bits that are set to 1 and on average is given by $m.p$. Fill ratio, thus, is a function of n, m, and k. In order to have the optimum number of k and f_p, we must have a lower bound on the size of the memory. Thus it is not suitable for applications running on systems with limited memory. Towards this end, we propose a Compressed Bloom filter (CmBF) that requires less memory than the standard Bloom filter to achieve a certain false positive rate f_p.

4.3 The Compacted Bloom Filter

CmBF requires less memory than a Bloom filter to achieve the same false positive rate. When used in a network application, it helps reduce the transmission cost and the storage cost at the endpoints. Moreover, unlike the CBF, compression and decompression techniques are not needed which makes CmBF more efficient.

CmBF is a probabilistic data structure like the standard Bloom filter that is used to represent a set $S= \{s_1, s_2, \ldots, s_n\}$ of n elements. Let the standard Bloom filter used to represent this set be an array of m bits. Let k be the number of independent hash functions used to construct the Bloom filter representation. The outputs of the hash functions are uniformly distributed over the discrete range $\{0, \ldots, m-1\}$. We now describe how to construct the corresponding CmBF from this standard Bloom filter representation.

The standard Bloom filter is first divided into k_0 blocks, denoted by *block_1, block_2, …, block_* k_0. Each block has n bits, the positions of which are denoted by $1, 2, \ldots, n$ as shown in Figure 4.3. A CmBF vector, denoted by CmBFV, is an array of n indices and each index contains m_0 bits, where $\log_2 k_0 \leq m_0 < k_0$. Note that, for m_0 bits, 2^{m0} patterns exist in the CmBF vector and the lower bound for m_0 is $\log_2 k_0$ as it is the minimum number of bits required to cover k_0 patterns.

Index *i* in CmBF vector, denoted by *CmBFV[i]*, corresponds to the value of the i^{th} bit positions in *block_1, block_2, …, block_n* in the standard Bloom filter. For example, the first index in CmBF vector, denoted as *CmBFV[1]*, has information pertaining to the first bit of each block in the Bloom filter. We define some rules about how to populate each index based on the bit pattern in the standard Bloom filter.

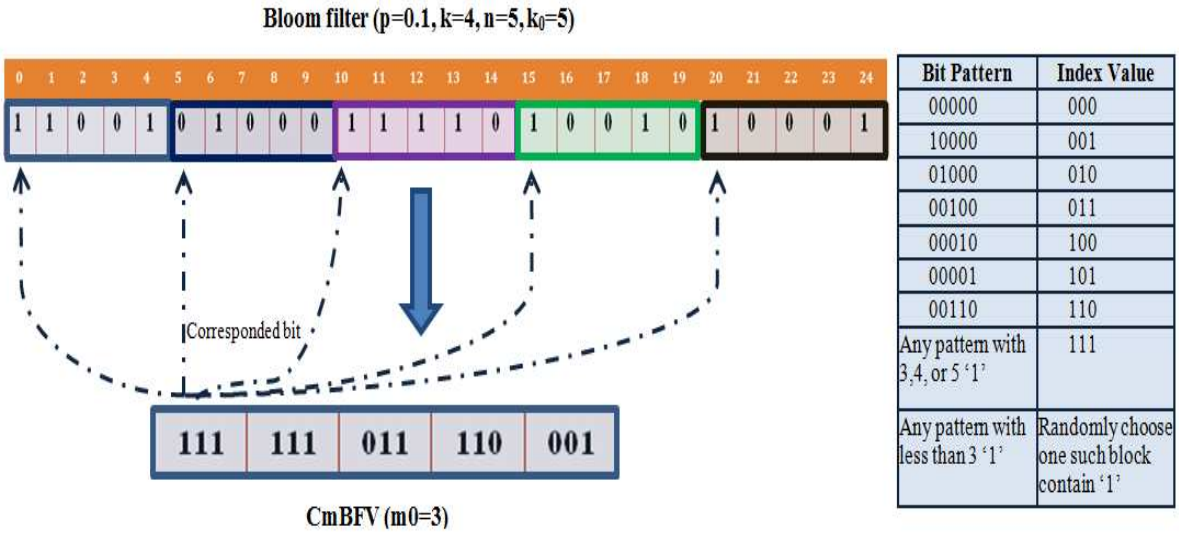


Figure. 4.2. An example for generating the Compacted Bloom filter

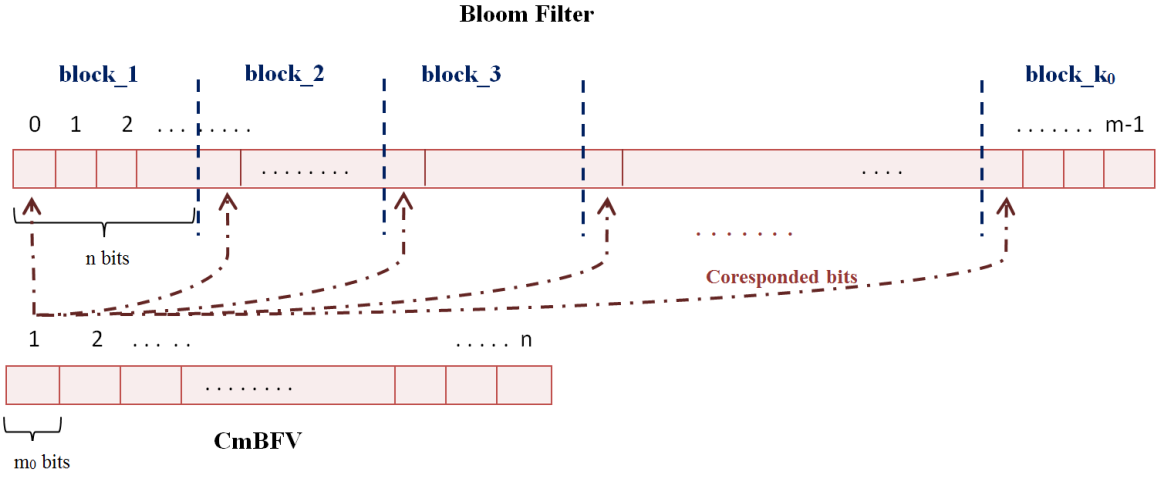


Figure. 4.3. Compacted Bloom filter

Consider the set of i^{th} bits in all blocks in the Bloom filter, which we denote as S_i. If there is no '1' in S_i, the i^{th} bit of any block, we set the corresponding bit in CmBF to 0, i.e., *CmBFV[i]*

$= 0$. If there is exactly one '1' in S_i, which appears in *block_r*, we set $CmBFV[i] = r$. If all the bits in S_i. are '1', we set $CmBFV[i] = 2^{m0} - 1$. If not all bits in S_i. are '1', but half or more than half of them are '1's, we set $CmBFV[i] = 2^{m0} - 1$ as well. In this case, some of the i^{th} bits having '0' may be interpreted as '1', thereby introducing more false positives. If less than half of the bits in S_i. are `1', we randomly choose one such block containing `1' in the i^{th} bit position, say *block_s*, and we set $CmBFV[i] = s$. Thus, all the other blocks that contain a '1' are interpreted as having a '0', thereby increasing the false negative rate compared to the standard Bloom filter. To deal with this issue, as $2^{m0} - 1$ different index values are available to assign bit patterns in CmBF, it is more efficient to use all available index values to cover more bit patterns. It reduces the number of bits interpreted as '1' or '0', which, in turn, decreases the false positive and negative rates. For this reason, one more rule is defined in this structure to cover unused index values. If bit pattern value is less than $2^{m0} - 1$ and has more than one '1', it means that the bit pattern value is in the range of unused index values. In this case, the index is assigned the bit pattern value. This case occurs in the fourth index value in Figure. 4.2. The following pseudo code in Figure 4.4 describes how to assign values to the CmBFV.

Figure. 4.2 illustrates how the CmBF is generated for $p=0.1$, $n=5$, $k=4$, $k_0=5$, and $m_0=3$. These parameters necessitate that the size of the standard Bloom filter be 25 bits and the size of CmBF be 15 bits. In this example, we achieve 40% reduction in size when with CmBF compared to that of the standard Bloom filter. Note that a false negative occurs when an inserted element is not recognized by membership queries, and it depends on how many of the bits assigned to '1' in the standard Bloom filter are interpreted as '0' in CmBF. Figure. 4.5 shows the Bloom filter generated from the CmBF described in Figure. 4.2. Note that, the generated Bloom filter, shown in Figure 4.5, differs from the original one shown in Figure. 4.2. In this CmBF example, three '0'

invert to ‘1’ compared with original Bloom filter that increase bit ratio and introduce more false positive. Moreover, one ‘1’ inverted to ‘0’ that in this case generates false negative in CmBF structure. Note, false negative is generated when a bit that is ‘1’ in the standard Bloom filter is reset to ‘0’ by CmBF rules. The probabilities of false positives and negatives are calculated in the next section and the effect of false positive and negative are shown in the evaluation part.

Algorithm1: Generate CmBFV

INPUT: Bloom filter (n, m, k_0, m_0)
OUTPUT: CmBFV
1: **for** i ← 1, n do
2: **if** |#′1′in Bit pattern[i]| = 0 % Bit pattern[i] is i^{th} bit of block_ (1…k_0)
3: CmBFV[i] ← Binary_value (0)
4: **else if** |#′1′in Bit pattern[i]| = 1 % ’1’ appears in block_r
5: CmBFV[i] ← Binary_value (r)
6: **else if** (Bit pattern[i]) < Binary_value($2^{m_0} - 1$)
7: CmBFV[i] ← Binary_value (Bit pattern[i])
8: **else if** |#′1′in Bit pattern[i]| < $k_0/2$
9: CmBFV[i] ← randomly select one of the blocks contains ‘1’
10: **else** % that means |#′1′in Bit pattern[i]| ≥ $k_0/2$
11: CmBFV[i] ← Binary_value ($2^{m_0} - 1$)
12: **end if**
13: **end for**

Figure. 4.4 pseudo code to generate CmBF

Querying a given element for the set membership is based on the bit patterns that we defined in constructing the CmBF. The advantage of CmBF is that it reduces the number of bits required. Thus for a given size of memory lower false positives are achieved; however, this structure provides some false negative as well. We explore this tradeoff and provide an in-depth analysis of the CmBF later.

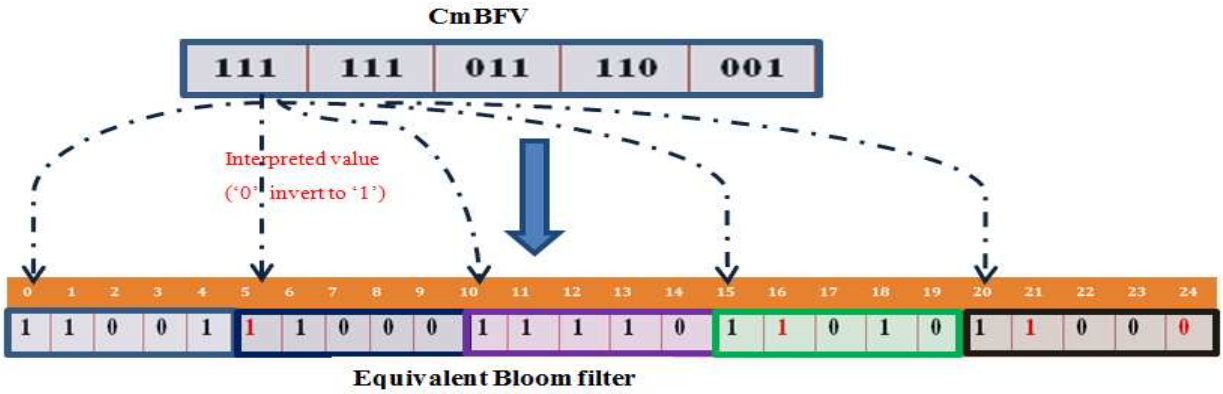


Figure. 4.5. Equivalent Bloom filter for k_0=5 and m_0=3.

4.3.1 False Negatives

False negative only occurs for inserting elements. A false negative occurs for an element if at least one of its marked bits is inverted, that is, a bit that is '1' in the standard Bloom filter is reset to '0' by CmBF rules. This condition happens when more than one but less than half of the bits is set to '1' when the element is inserted in the standard Bloom filter. Thus, the false negative rate depends on the number of '1's in the corresponding bits in the Bloom filter. The false negative rate is calculated from the probability of the number of 1's that are less than half bits in the corresponding index. As we show in Eq. 4.1, the probability of a given bit set to '1' in array after inserting n elements can be expressed by $(1-e^{-k\frac{n}{m}})$ and therefore, probability of a given bits set to '0's is given by $e^{-k\frac{n}{m}}$. Thus $f(i,k_0,p)=P(X=i)$ is the probability of getting exactly i number of '1's in k_0 trials when X follows the binomial distribution as shown in Eq. 4.4. On average, q_0 is the number of bits set to 0 when the bit is initially 1 and (i-1) in this equation shows the number of bits interpreted to '0' when $P(X=i)$ occurs.

$$q_0 = \sum_{i=2}^{\lfloor k_0/2 \rfloor} P(X=i)(i-1)k_0 \qquad (4.3)$$

Where,

$$P(X=i)=\binom{n}{i}(1-e^{-k\frac{n}{m}})^{i}(e^{-k\frac{n}{m}})^{n-i} \quad (4.4)$$

A false negative occurs if at least one of the selected bits is an inverted bit, that is, a bit in '1' is reset to '0'. Hence from the bit distribution, the average false negative probability f_n is calculated as shown below in Eq. 4.5, where q_1 is the number of bits on average set to 1 in the standard Bloom filter. According to Eq. 4.5, a false negative occurs if at least one of the selected bits is an inverted bit and the others have '1'.

$$f_n=\sum_{i=1}^{k}(\frac{q_0}{q_1})^{i}(1-e^{-k\frac{n}{m}})^{k-i} \quad (4.5)$$

$$q_1=m.(1-e^{-k\frac{n}{m}}) \quad (4.6)$$

As the example consider the CmBF is generated in Figure. 4.5 for $p=0.1$, n=5, k=4, k_0=5 and m_0=3. To calculate false positive probability first step is finding the probability of having the number of 1's that are more than one but less than half bits in each block. This condition means we should find the probability of getting 2 number of '1' in block. Therefore, in this example $P(X=2)$ is calculated from Eq. 4.4 where it goes to equal 0.275. In the next step, we should identify the number of bits change from '1' to '0'. q_0 in Eq. 4.3 is defined as the number of bits set to 0 when the bit is initially 1. According to this equation (i-1) is equal to 1. It means the number of bits interpreted to '0' is 1 for $X=2$ in a block. In the other word, the number of bits needs to be converting when we getting 2 number of '1' in a block is 1. So to determine on average number of bits change from '1' to ''0', $P(X=2)$ is multiplied by $1*5$ as shown in Eq. 4.3. q_0 is calculated from Eq. 4.3 and is equal to 1. It means, on average the number of bits set to 0 when it was initially 1 after creating CmBF is 1. In the next step, q_1 is calculate from Eq. 4.6 as

the number of bits on average set to 1 in the standard Bloom filter. In this equation length of Bloom filter *m* multiply by $(1-e^{-k\frac{n}{m}})$. As explained $(1-e^{-k\frac{n}{m}})$ is the probability of a given bit set to '1' in array after inserting *n* elements. So by multiplying it to *m* we determine the number of bits set to 1 on average in the standard Bloom filter. According to this equation q_1 is obtained 13. In the last step the average false negative probability f_n is calculated according to Eq. 4.5.

4.3.2 False Positives

To calculate the false positive probability of the CmBF, we first calculate the probability of a bit being set to 1 when the bit is initially 0. This condition happens when the numbers of 1's are more than half bits in the corresponding index. On average, q_2 bits is set to 1 when the bits are initially 0. q_2 is expressed by:

$$q_2 = \sum_{i=\left\lceil \frac{k_0}{2} \right\rceil}^{k_0-1} P(X=i)(k_0-i)k_0 \tag{4.7}$$

From the bit distribution, the probability of a false positive is the probability that we find bit in '1' for each of the *k* positions. Since on average q_2 bits are set to 1 when the bit is initially 0 and q_0 bits are set to 0 when the bit is initially 1. Hence the number of 1 (i.e. fill ratio) in CmBF is:

$$q_3 = q_1 + q_2 - q_0 \tag{4.8}$$

As a result, the probability of f_p for the CmBF is calculated as:

$$f_p = (\frac{q_3}{m})^k \tag{4.9}$$

Consider the CmBF is generated in Figure. 4.5 for $p=0.1$, $n=5$, $k=4$, $k_0=5$ and $m_0=3$ as the example. To calculate the false positive, we first calculate the probability of a bit being set to 1 when the bit is initially 0. This is equal to when the numbers of 1's are more than half of bits in the blocked Bloom filter. This condition means we should find the probability of getting 3 or 4 number of '1' in block and it is equal to $P(X=3)$ and $P(X=4)$ in this example. From Eq. 4.4 these probability can be calculated and are equal to 0.337 and 0.2065 for $P(X=3)$ and $P(X=4)$ respectively. Therefore, in the next step we can obtain q_2 as the average number of bits is set to 1 when the bits are initially 0 from Eq. 4.7. That is equal to 4. This means on average 4 bits convert to 1 from 0 after creating CmBF in this example. Hence the total number of 1 in CmBF is equal to 16 from Eq. 4.8 where q_1, q_2 and q_0 are equal to 13, 4 and 1 respectively. At the end, the average false positive probability f_p is calculated according to Eq. 4.9. q_3/m in this equation identify fill ratio and f_p as the probability of finding a bit set to 1 for each of the k positions is equal 0.162 accordingly.

4.4 Evaluation

In order to analyze the behavior of the CmBF, we simulated it in Matlab with functions for inserting and checking elements in a bit array. For each simulation round, we selected the number of hash functions based on the desired false positive rate and inserted the elements into the Bloom filter. The next step involves compressing the Bloom filter. The CmBF algorithm is run and then the false positive rate is computed according to the number of bits that are inverted (i.e., a bit in 1 is reset or a bit in 0 is set). To estimate the false negative, we check whether CmBF recognizes the inserted elements or not. Each inserted element that is not recognized by the CmBF is reported as a false negative. We run 500 simulation rounds. For each simulation round, we select new hash functions and a new dataset to insert into Bloom filter. We used traffic

dataset collected from University of Auckland in New Zealand [6] and hash source IP addresses into the Bloom filter. Simulation results are very close to the analytical results as shown in Figure. 4.6, which serves to validate the analytical expressions derived in the previous section. In Figure. 4.6, we show the false positive probability of CmBF as a function of the number of hash functions for three different sizes of the Bloom filter for both the analytical and the empirical solutions. Thus, the rest of the graphs presented are based on theoretical assessment only.

We test the theoretical framework above by examining a few specific examples of the performance improvements possible using CmBF. We consider cases where 10 bits are used in the Bloom filter for each element that is $m/n = k_0 = 10$. Then, using the optimal number of hash functions $k=7$ yields the false positive rate of 0.0081. According to CmBF algorithm the number of bits per index in this case can be in the range of 4 to 9. Therefore the memory size is reduced from 60% to 10%. Our optimization is based on the assumption that we wish to minimize the memory requirement m as functions of k and p. The primary point of this theoretical analysis is to demonstrate that compression is a viable means of improving performance, in terms of reducing Bloom filter size for the desired false positive and negative rates. The other important parameter is false positive rate in CmBF structure. In membership queries, a false positive implies recognizing an element that does not belong to the set. As the false positive probability increases, more and more non-members are incorrectly identified as belonging to the set. Therefore, a low false positive probability is desired.

First, we note that the false positive probability of CmBF reduces as the number of hash functions decrease for specific n and m. This assumption is reasonable since the false positive probability of a CmBF f_p is a function of fill ratio $(1-e^{-k\frac{n}{m}})$, according to Eq. 4.4. As the

number of hash functions decrease, the fill ratio of the Bloom filter also decreases. Note that, on average, the number of bits that are set to '1', when the bits are initially '0', decreases as we reduce the number of hash functions. Moreover, when the fill ratio decreases, it is easier to find the number of 1's in the blocked Bloom filter that are less than half of the bits in the corresponding index, but harder to find number of 1's that are more than half of the bits. When less than half of the bits are set, our CmBF algorithm randomly represents one block and resets the bits corresponding to the other blocks. Moreover, when half or more of the bits are set, it sets the bits corresponding to all blocks as '1'. Thus, when fill ratio decreases, the probability of bits being reset increases and the probability of bits being set decreases, contributing to a lower false positive rate. In Figure. 4.7, false positive probability of a CmBF f_p as a function of k for m/n=10 is shown. For all different number of bits per index m_0, we have a similar behavior. The false positive probability of the standard Bloom filter always increases when the number of hash functions decreases, which is in accordance with Eq. 4.2. As Figure 4.7 depicts, for k=7 the false positive probability of standard Bloom filter and CmBF have a noticeable difference. However, for k=5 false positive rate for CmBF falls over 40% and CmBF and Standard Bloom filter false positive rates are very close to each other and for k=3 and 2 the CmBF have a lower false positive probability. Thus, the results demonstrate that the CmBF allows an equal or lower false positive rates for a given n and m compared with the standard Bloom filter.

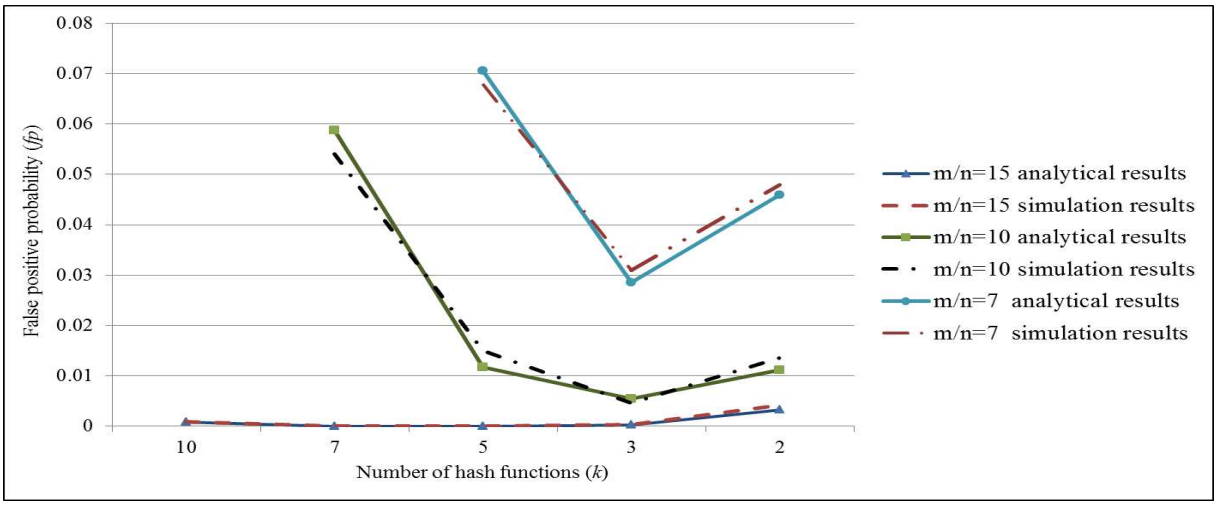


Figure. 4.6. False positive probability of CmBF as a function of the number of hash functions for m/n=7, 10 and15.

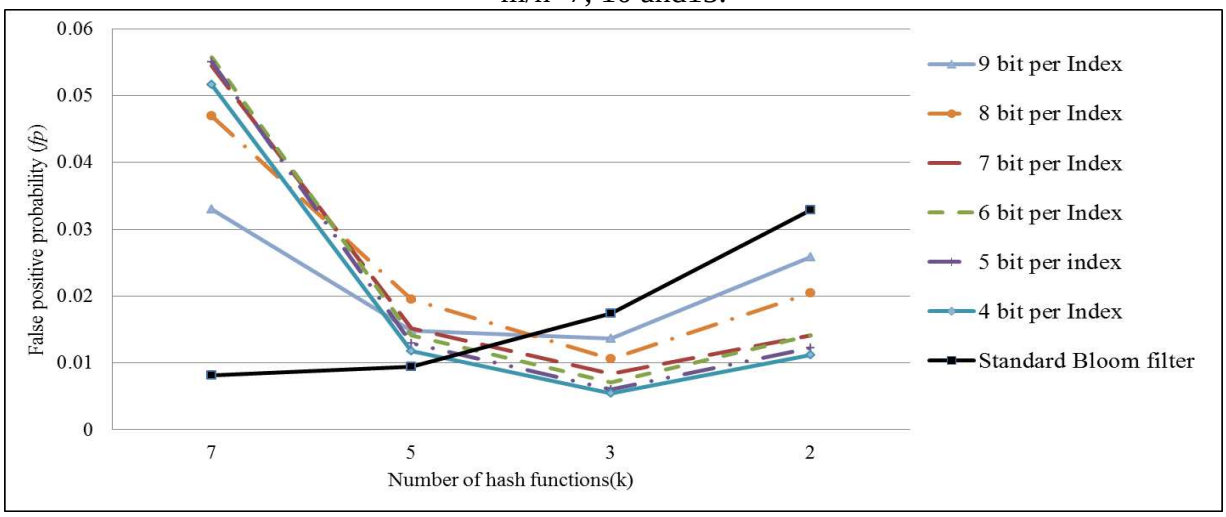


Figure. 4.7. The effect of increasing bits per index on the false positive probability of CmBF as a function of the number of hash functions for m/n=10.

The advantage on CmBF comes at a cost. The cost in this case is the introduction of false negative probability in membership queries for CmBF. A false negative implies not detecting an inserted element. In Figure. 4.8 we show the false negative probability for different number of bits per index for *m/n*=10 and *k*=7, 5, 3 and 2. We notice that the false negative probability always equals zero for the standard Bloom filter, as expected. Since the standard Bloom filter never resets bits, it is impossible to have a false negative. For the CmBF, however, this behavior occurs because some of the bits marked as '1' is inverted to '0' during construction. In Figure

4.8 we see that the false negative rate increases as the number of hash functions decrease for a specific number of bits per index. This happens because with less hash functions it is more likely that less than half of the corresponding bits for each index will be '1'. As a consequence, the probability of an inversion of bit to '0' is higher and false negative rate increases. From Figures 4.7 and 4.8 we see that the number of hash functions used in the CmBF has a dual effect. On one hand, decreasing the number of hash functions reduce the false positive probability as shown in Figure. 4.7. On the other hand, it increases the false negative probability as shown in Figure. 4.8. Some applications, such as IP traceback [45] [46], require smaller false negative rate, but can tolerate higher false positive rate. However, for other applications such as P2P, the opposite can be true [48] [49] [50]. As a result, it is important to select appropriate parameters so that we can provide an ideal tradeoff.

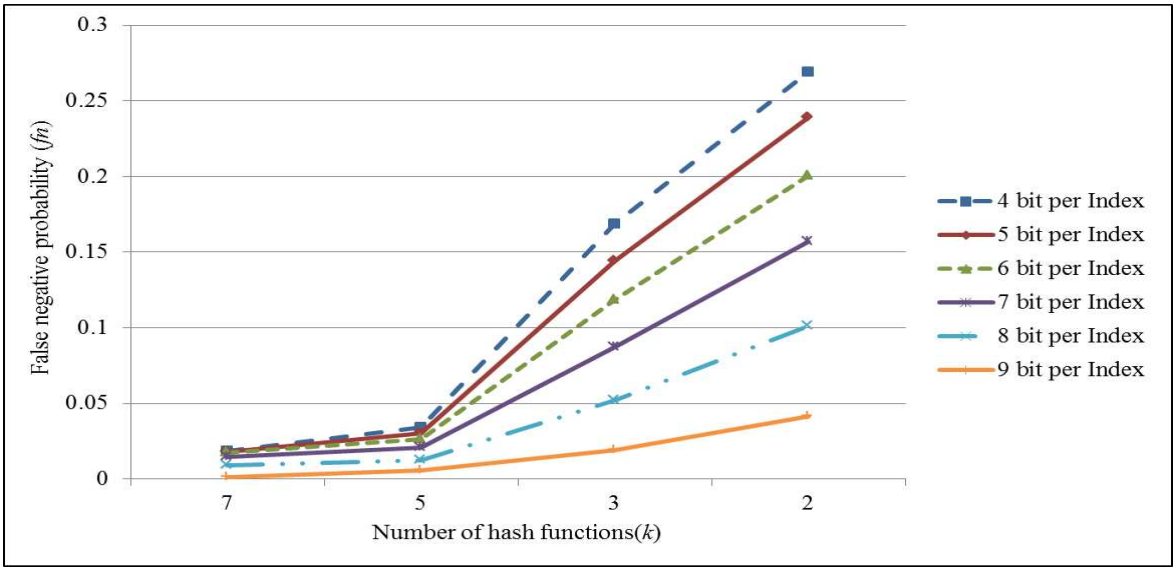


Figure. 4.8. The effect of decreasing the number of hash functions on the false negative probability of CmBF for m/n=10.

To deal with adjusting desired properties, it is possible to degrade the false negative rate as a tradeoff cost with reduction of memory size. As seen in Section 4.3, the false negative in CmBF arise because bits set as '1' are reset when the numbers of 1's are less than half of the total

number of bits in the standard Bloom filter. Note that, this behavior can be somewhat mitigated by increasing the number of bits per index as more bit patterns can be used to cover more conditions, which, in turn, will reduce the false negative and false positive rates. These extra bits can represent more patterns that can be used when the number of 1's is less than half bits in the standard Bloom filter. Note that, as stated in Section 4.3, when less than half of the bits are '1' in corresponding index, the rule defined is one of the block numbers are randomly chosen and indicated as '1' and others are reset to '0'. However, with the increased number of bits, it is possible that some pattern can be used to indicate accurately the exact blocks containing '1', thereby reducing the false positive and false negative rates. The number of such patterns that can be covered depends on the number of bits per index. Increasing the number of bits per index requires more memory for CmBF but reduces the false negative rate. Depending on the application it may be important to reduce the probability of false negative at the expense of higher memory requirement. Figure. 4.9 and 4.10 depict the f_n, f_p and memory reduction size as a function of the number of bits per index for different values of k, for m/n=10.

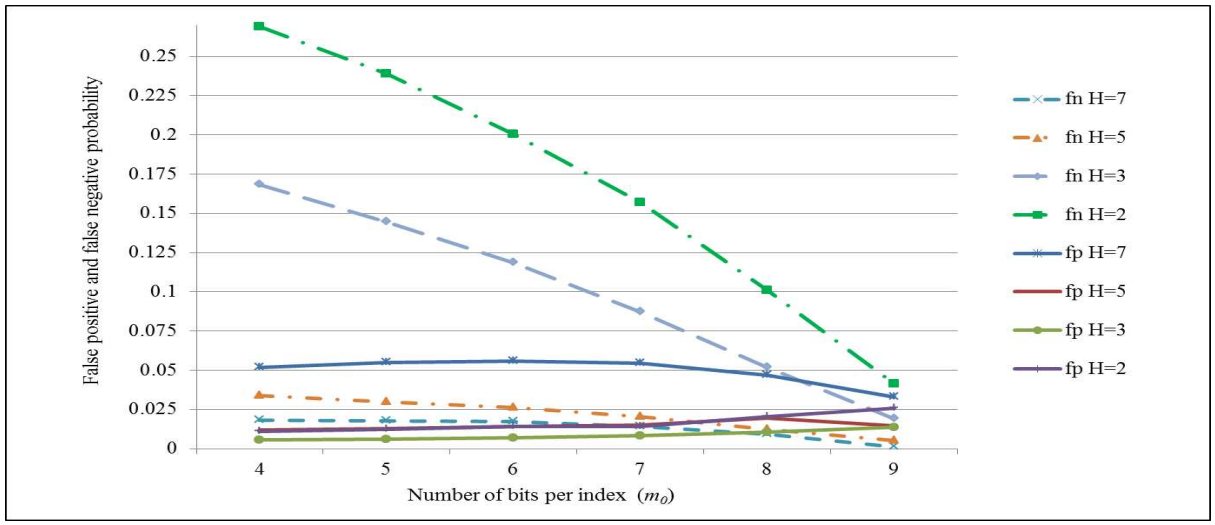


Figure. 4.9. The effect of decreasing the number of bits per index on the false positive and negative probability of CmBF for m/n=10.

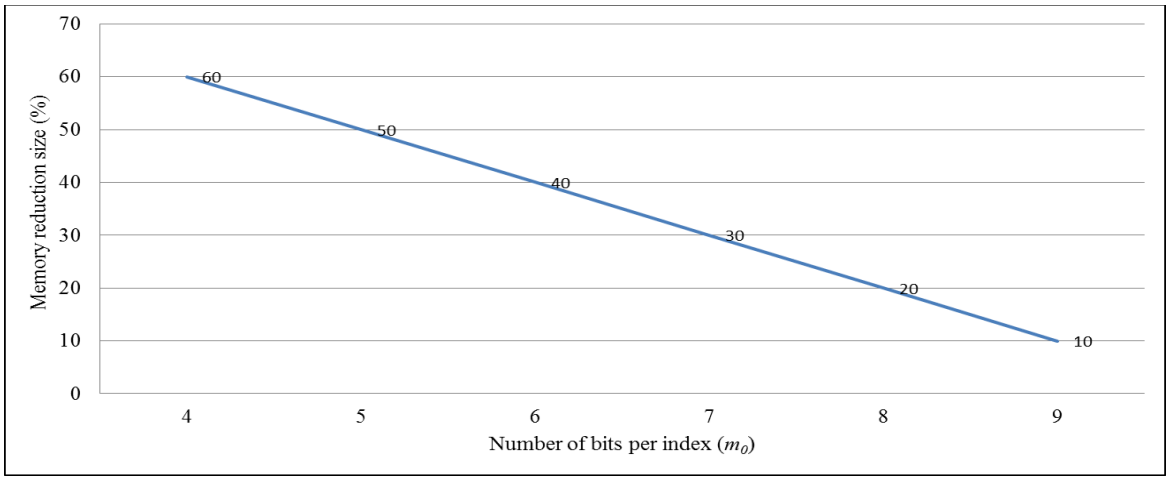


Figure. 4.10. The effect of decreasing the number of hash functions on the false negative probability of CmBF for m/n=10.

We see that the probability of false negatives reduces dramatically as we increase the number of bits per index and it reaches close to 0 (0.0012%) with increase in the number of bits per index. As an example, based on our result for m/n=10, h=5 and 5 bits per index, f_p is 0.011(standard Bloom filter has f_p = 0.0094 in this condition), f_n is 0.03 and we have 50% memory reduction size. In the other worlds, in CmBF by accepting 0.03 false negative rates we can reduce 50% memory requirements with almost similar f_p compared with the standard Bloom filter.

In short, we can do tradeoffs between the false positive rate, false negative rates, and memory reduction using our CmBF data structure and use it for diverse kinds of applications.

4.5 Conclusion

In this chapter, we have introduced CmBF, a new data structure that uses less memory but provides the same functionality as the standard Bloom filter. CmBF reduces the memory requirements by introducing the false negatives in membership query. We derived expressions and provided tradeoffs between false positive, false negative and memory reductions. Such knowledge of tradeoffs can be used for different applications that require the functionality of

Bloom filters but aim to minimize the communication costs and have less memory at the receiving end.

CHAPTER 5

A DISTRIBUTED MECHANISM TO PROTECT AGAINST DDOS ATTACKS

5.1 Introduction

In this chapter we propose one such mechanism that when deployed is able to filter out attack traffic and allow legitimate traffic in the event of an attack. Our goal is to place such filters that block attack traffic and allow legitimate traffic as close to the source node as possible so that network resources are not wasted in propagating the attack. Our scheme introduces an algorithm that identifies the most effective points where the filter can be placed so as to minimize attack traffic and maximize legitimate traffic for the victim node and the other end users during the attack period. We evaluate the effectiveness of our scheme using extensive simulation in OPNET© with real network topology. The experimental results demonstrates the efficacy of the proposed scheme in blocking attack traffic at the upstream routers close to the source of the attack and also shows the effectiveness of the scheme in allowing the legitimate traffic from reaching the victim node. The results prove that presented algorithm accurately selected desirable filtering points where all the filtering routers are selected at first three routers from attacker node and stop 95% of attack traffic while allowing 77% of legitimate traffic to reach to victim node for CAIDA attack dataset.

Denial of Service (DoS) attacks, which are intended to make the service unavailable for legitimate users, have been known to the network research community since the early 1980s. DDoS attacks against commercial websites like Yahoo, Ebay and E*Trade have provided evidence of how legitimate user access may be blocked by DDoS attacks and cause financial loss [1] [2]. Moreover, emergency and essential services rely on the network infrastructure, and thus

DDoS attacks may have severe consequences to the public as well. Consequently, defense against DDoS attacks has become one of the most important research issues, with techniques for preventing, detecting, and surviving such attacks being developed [3]. Despite significant research into countermeasures, DDoS attacks still remain a major threat today [27]. Recent examples include a record 400 Gbit/s DDoS attack against CloudFlare, a rate about 100 Gbit/s more than the largest previously seen DDoS attack [7].

The frequencies and the impact of DDoS attacks have motivated researchers in the Internet security community to provide techniques for preventing, detecting, and surviving such attacks [3]. Most approaches have failed to provide service availability in the presence of DDoS attacks. In this chapter we address this issue and present a distributed responsive defense approach. We demonstrate how to distinguish attack traffic from legitimate traffic. Moreover, we aim to stop the attack traffic as near as possible to the source of the attack while allowing the legitimate traffic access to the victim node.

In general, there are several factors that make defending against DDoS attacks hard. First, the traffic flow volume through the victim node can reach up to 400Gbps [7]. Second, the attack occurs in a highly distributed manner – this gives the illusion of legitimate traffic and makes detection difficult. In addition, a flooding attack can appear like a flash crowd, which occurs when a large number of legitimate users connect to a server simultaneously [8]. Differentiating the attack traffic from flash crowd traffic is one of the hardest steps in detecting and protecting against DDoS attacks. The presence of zombies that spoof the IP address of the source of the attack make it even more difficult to identify and trace the attacks back to their actual sources.

DDoS attack countermeasures can be classified into three classes depending on when they are applied with respect to the attack. Some are preventive techniques that are applied in order to

stop the occurrences of attacks [2], [59], [60], [61]. Some are detection mechanisms that aim to identify attack traffic [62], [5]. Other techniques attempt to identify the sources of attack and respond to those [63], [5], [64], [65]. However, a comprehensive DDoS defense mechanism should include all three categories of defenses since there is no one-size-fit-all solution to the DDoS problems [66]. Attack source identification and responsive techniques actively try to mitigate DDoS attacks by filtering or limiting attack packets [27]. Such schemes have two components, namely, attack detection and packet filtering. The characteristics of attack packets, such as source IP address or marked IP header values [67], [59], [68], are often used to detect and identify attack traffic. These characteristics are used for packet filtering. Note that packet filtering can be applied either close to the attack node [62], [5], [69] or close to the victim node [67], [59], [68], [62], [70] where all the attack traffic aggregate. However, applying attack traffic filtering at the victim node limits the effectiveness of filtering as the victim may crash while dealing with an overwhelming volume of attack traffic. In addition, the high volume of attack traffic may still overwhelm upstream Internet resources when the filter is applied close to the victim node. Thus, it is desirable to filter attack traffic as close as possible to the attack sources. However, it is difficult to anticipate and identify such nodes as the attack may originate at widely distributed nodes and spread through various routes [66]. Therefore, a distributed approach for filtering packets as close as possible to the source may be needed. Detection and response can be performed in different places on the path between the victim and the source of the attack as it is desirable for any DDoS defense mechanism to detect attacks quickly and stop them as close to the source as possible.

Since attackers cooperate to perform successful attacks, defenders must also form and collaborate with each other to defeat the DDoS attack. Consequently, a lot of work appears in

distributed defense mechanisms for DDoS attacks [22], [23], [24], [25], [26]. Some schemes [22], [23], [24] detect attacks downstream close to the victim and this attack signature is propagated upstream to the routers doing filtering located close to the source of the attack. The drawback of these approaches is the difficulty of securely forwarding the attack signature to the upstream routers. Some schemes [23], [24] need a global key distribution infrastructure for authenticating and verifying the attack signature [27] which is used for differentiating the attack traffic from the legitimate one. Note that, creating an accurate attack signature is difficult due to the presence of zombies who can spoof the IP address of the attack source. Firecol mechanism [25] also presents a distributed collaborative system to detect flooding DDoS attacks. This mechanism relies on an Intrusion Protection System (IPS) located in the Internet service providers. IPSs are effective in stopping attack traffic, but they have very large communications overhead and also significantly decrease the performance of the routers.

An alarming trend associated with individual DDoS attacks has been the dramatic increase in traffic associated with attacks. Recent examples include a record 400 Gbit/s DDoS attack against CloudFlare, a rate about 100 Gbit/s more than the largest previously seen DDoS attack [7]. At these traffic intensities the network infrastructure upstream from the intended victim becomes severely affected, making it essential to filter traffic close to the sources. Yet, the defense scheme should not consume significant resources at the routers to identify packets to filter, especially at those routers not carrying significant volumes of attack traffic. In this chapter we propose a novel distributed DDoS defense mechanism for achieving these goals, which does not consume any router resources in the process of identifying routers for up-stream filtering. Moreover the approach tries to minimize the modifications required to the routers and the current protocols to combat DDoS attacks with a low complexity and scalability. The mechanism aims to maximize

the arrival rate for the legitimate traffic and minimize the attack flow during the attack time. According to our approach, the first part involves developing rules to create a history-based profile on the high confidence IP addresses that serve to differentiate the good traffic from the malicious ones during the attack time. The history is a collection of legitimate IP addresses that have appeared in the past. The second part is capture IP address history in the form of a Bloom filter for efficient transfer. The third part of our approach which is the main contribution of this chapter identifies how and where this history is used to prevent the attacks causing network congestion. Placing the filter in the upstream routers incurs storage and performance costs since the filter must be applied to multiple routers. Placing the filter closer to the victim causes the link capacity to become saturated and wastes network resources. Our scheme introduces an algorithm that identifies the routers where the filtering can be placed. We do this by monitoring the network traffic during the attack period. To the best of our knowledge, we do not know of any work that considers the optimal placement of the filters to detect DDoS attacks. The rest of this chapter is organized as follows. Section 5.2 enumerates some related work. Section 5.3 describes our responsive defense mechanism. Section 5.4 presents our simulation results. Section 5.5 concludes the chapter.

5.2 Related Work

DDoS attacks were first recognized around 2000 and researchers have worked on providing countermeasures to these attacks since that time [28]. Some of them use a centralized approach [67], [59], [62], [60], [5], [71] while others are distributed mechanisms [22], [23], [24], [25], [22], [72], [60], [73], [64], [74]. Some research focused on developing cooperative mechanisms using a set of nodes to thwart DDoS attacks. Pushback [22] is one of the earlier efforts that mitigate DDoS attacks using a cooperative mechanism. The approach has several shortcomings. First,

legitimate traffic is severely penalized. Second, in this approach we need to install the mechanism in all the routers – this may be unacceptable to some routers because of the high computational and memory overheads. Other techniques, such as K-Max-in [75], Secure Overlay Services (SoS) [72], Hop Count Filtering (HCF) [60], ANTID [73] and Decom [64], have tried to detect attacks based on observing some of packet's information such as TTL value and path fingerprints. However, a number of drawbacks exist in these techniques [72], [75], [60], [73], [64], [76], [77] including the cost associated with them and universal deployment required at the ISP level to be effective. While those approaches are considered costly in terms of resource consumption, other proposed mechanisms using simple statistics are not distributed [62]. In [68] each router between the source and destination marks the path to detect spoofed addresses. H. Wang et al. in [78] also analyzed the correlation between the request and replies to detect flooding attacks. Detecting the DDoS attacks at the ISP level was studied in [79] and [80] as well. Francois et al. [25], [81] recently presented a method called Firecol that consists of collaboration mechanisms of Intrusion Protection System (IPS) organized in the form of a virtual ring at the ISP level. The IPS forms a virtual protection ring around the hosts that are being protected and the members in the ring collaborate by exchanging selected traffic information. The detection system protects its clients based on parameters such as IP traffic patterns, ports or protocols in use. The evaluation part demonstrates that this technique performs well with respect to protecting the hosts, but has a high overhead and performance cost for routers. In addition, updating the score list and ring communications increase the overhead cost, especially during the DDoS attacks, where lack of bandwidth may hinder the protocol communication. The other recent cooperative approach TDFA [26] also presents a distributed defense mechanism that consists of three main components: detection, traceback and traffic control. The scheme tries to identify

source of attack by traceback scheme and reduces the rate of forwarding the attack packets and therefore improves the throughput of the legitimate traffic. TDFA consumes a lot of resource during the attack time. One of the fundamental deployment and operational challenges is ensuring a sufficient number of routers to support traceback before it is effective. Moreover traceback mechanism has computational and network overhead specially during the attack time. Our proposed distributed approach tries to minimize resource consumption, detects DDoS attacks early, identifies the attack source, and finally stops the attack as close as possible to the attack source.

5.3 Distributed Responsive Defense Approach

This section presents our scheme to identify upstream routers, and blocking the DDoS attacks at these routers to minimize the impact both to the victim and to the upstream network during the attack time. Proposed DDoS mitigation mechanism consists of the following components: (1) identification model to discriminate attack traffic from legitimate traffic based on a history-based profile, (2) capture the history-based profile in the form of a Bloom filter for efficient transfer, (3) identify the responsive points (router/switch) which carry the attack traffic, and 4) activate packet filtering at selected points. First two items are addressed in detail in [82] and only briefly reviewed here. The main contribution of this chapter is the overall scheme and the last two components.

5.3.1 Identification Model

To continue to provide services under DDoS attacks, it is essential to distinguish attack traffic from legitimate traffic. History based profiles are effective in identifying significant fractions of legitimate traffic as well as attack traffic. We addressed this problem in [82] by investigating the

specific attack features as well as normal traffic characteristics to identify the attack traffic from legitimate traffic. Our parameter set for establishing the identification model are source IP address and port number, destination IP address, size of packet, packet type (ICMP, UDP, TCP), frequency of IP address and for TCP traffic, those IP addresses with a successful TCP handshake. Then accurate normal traffic signatures with a scoring mechanism based on these features developed. The effectiveness of this approach in blocking attack traffic and allowing the legitimate traffic at upstream routers has been demonstrated using DARPA data set in [82] as well. Our experiment indicates that our filtering model can protect the victim node from 95% of attack traffic while allowing 70% of legitimate traffic.

5.3.2 Bloom Filter Mechanism

Since the filtering mechanism need to be applied in upstream routers, and the network bandwidth may already be saturated during an attack, transferring the entire history and looking it up in the upstream routers is expensive where upstream routers must process all packets targeted towards the victim node. A Bloom filter based mechanism is used [82] to efficiently implement and disseminate the proposed identification model; it helps reduce the communication and computation costs and the storage requirements of the upstream routers that check for malicious traffic. In general, Bloom filter [40] is a space efficient probabilistic data structure at the cost of probable false positives for presenting whether an element is a member of a set or not since in our approach the Bloom filter represents the contents of the IP address history. There are three fundamental performance metrics for Bloom filters: false positive rate, size of the Bloom filter array and number of hash function. In our analysis, we limit the false positive rate to 0.1. In order to achieve this false positive rate, the Bloom filter array is 5 times the number of IP

addresses kept in the history. Moreover, 4 hash functions require providing a Bloom filter from the IP address history.

5.3.3 Responsive Points' Identification

The third step build up the main contribution of this chapter and it is how and where to use this filtering to minimize the impact of the attack. Placing such filters at the upstream nodes may be expensive because all packets passing through such a node must be processed irrespective of whether a significant amount of attack traffic passes through it or not. Placing filters close to the victim nodes, in contrast, causes resource wastage as the attack traffic passes through the network before it is stopped. Our proposed solution addresses this problem by using a recently developed technology, typically implemented as Small Formfactor Probes (SFP) using FPGAs (Field Programmable Gate Array). Proposed approach monitor traffic by SFProbes to efficiently identify router/switch which carry the attack traffic and then apply packet filtering at selected routers as the responsive point of defense mechanism. An example of such hardware is JDSU SFProbes and Packet Portal [30]. Packet Portal is a new approach for gathering, distributing and analyzing information from distributed Ethernet ports [30]. SFProbes can plug into any SFP compatible elements such as switches and routers in such a way that it taps into the normal fiber without interfering with the traffic flow. It can be programmed, over the network using the same fiber, to do tasks such as counting the number of packets with certain values in header fields and forward information about link traffic to a remote base station. These packet portals have been deployed on operating networks for network monitoring functions. Our approach uses these probes, at a subset of ports in the network, to identify the upstream links, and thus nodes, which carry attack traffic. The probes connected to the router/switch ports, e.g., JDSU Packet Portals, communicate with a remote base station. It is this feature that we use, as described below, to

send the history-based profiles to identify the paths with high intensity of attack traffic. Moreover the portal base station (PBS) has knowledge about SFProbes attached to the routers throughout the network and can collect data from SFProbes and perform the computation needed for our scheme, obviating the need for routers performing such computations. The portals can be globally time synchronized to less than one millisecond, thus enabling synchronized measurements that were not previously possible [83].

When an attack is detected on the victim network, our algorithm starts to protect a victim node as illustrated in Figure. 5.1. At this point the victim network sends the Bloom filter that created in section (ii) to PBS. As we mentioned the Bloom filter represents the content of IP address history and it causes a significant reduction of message traffic and introduces an efficient IP address look up mechanism. The PBS sends Bloom filter to those SFProbes that have plugged into the routers as shown in Figure. 5.1. SFProbe starts monitoring the intensity of traffic directed toward the victim node in defining time slots.

Suppose $t_1, t_2, \ldots t_m$ be discrete time slots and $X(t_m, i)$ be the number of packets received by a router during time slot m at SFProbe i toward the victim node. Eq. 5.1 is defines the historical estimate of the average number of packets received by a router $\bar{X}(t_m, i)$, where α is a weighted value between 0 and 1.

$$\bar{X}(t_m, i) = (1-\alpha)\bar{X}(t_{m-1}, i) + \alpha X(t_m, i) \tag{5.1}$$

Let $A_j (t_m, j)$ represent a Boolean variable which equals 1 if the packet Pj is received at router i at time slot t_m and belong to the corresponding Bloom filter $B(v)$ for victim point v and is 0 otherwise.

$$A_j(t_m,i)=\begin{cases}1 \;\; if \;\; P_j(t_m,i)\in B(v)\\ 0 \qquad Otherwise\end{cases} \tag{5.2}$$

Let $\overline{W}(t_m,i)$ define the historical estimate of the average number of packets received by the router at SFProbes *i* during time slot *m* that match with the Bloom filter. In the other word, $\overline{W}(t_m,i)$ shows the average number of packets that is considered as the legitimate traffic by the Bloom filter directed towards the victim and is given by the following equation where, *n* is the total number of packets follow toward the victim node at time slot t_m :

$$\overline{W}(t_m,i)=(1-\alpha)\;\overline{W}(t_{m-1},i)+\alpha\;\sum_{j=1}^{n}\frac{A_j(t_m,i)}{n} \tag{5.3}$$

During the attack time, if the number of IP addresses that do not match with the Bloom filter is high the router is likely to be carrying significant attack traffic. This makes it possible to inform the router to drop packets directed towards the victim node. We use the following parameter in Eq. 5.4 that is called Attack Estimation Rate (ASR) $R(t_m,i)$ to determine the average number of packets that do not match with Bloom filter.

$$R(t_m,i)=\frac{\overline{X}(t_m,i)-\overline{W}(t_m,i)}{\overline{X}(t_m,i)} \tag{5.4}$$

When the attack starts, SFProbes start monitoring traffic going towards the victim node and sends $R(t_m,i)$ to the PBS. Next we decide the points at which the filters are to be placed. In order to save network resources, the best routers to apply the filtering mechanism must be as far away as possible from the victim node. For instance, if routers with 2 and 3 hop distances have the same attack detection rate, it is more efficient to place the filter on the router which is at hop

distance 3 from the victim node instead of the one which is at hop distance 2. We take this into account while considering the placement of the filtering mechanism in the routers. Moreover, we also take into account the volume of potential attack traffic passing through a router when considering the placement of routers. Thus, we use two factors to determine the best routers to place the SFProbes – one is the Attack Estimation Rate based on Eq. 5.4 and the hop distance $H_i(v)$ that shows how far the SFProbe i is from the victim node v. For each SFProbe i computer the weighted attack estimation rate $S(t_m,i)$ is given by the following equation.

$$S(t_m,i) = \frac{\overline{X}(t_m,i) - \overline{W}(t_m,i)}{\overline{X}(t_m,i)} * H_i(v) \tag{5.5}$$

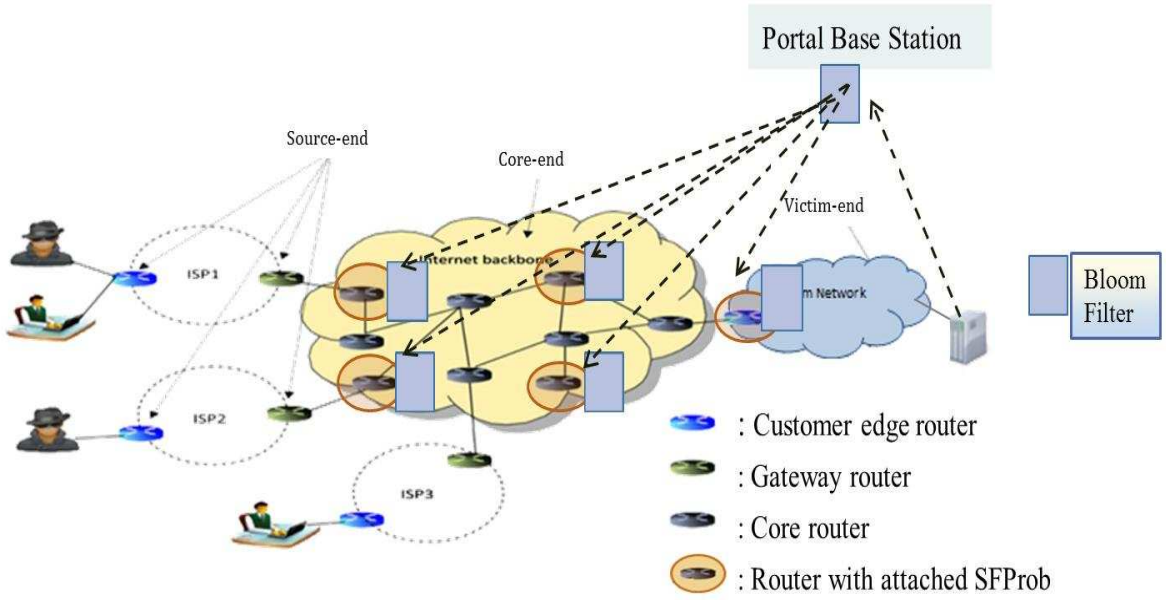


Figure. 5.1. Responsive defense mechanism

The routers with higher value of $S(t_m,i)$ will be selected as the more effective routers to apply the filtering mechanism. We evaluated this presented approach called as the Responsive Point's Identification algorithm by considering Hop distance, and Attack estimation rate (RPI-HA). The results are shown in the next section.

In addition to the hop distance and the volume of attack traffic factors, other issues must also be considered while placing the filters. One such additional factor is the number of routers on which filters are placed based on the distribution of the attack traffic. We present a new formula that adjusts the number of routers according to the attack traffic distribution that we refer to as the Responsive Point's Identification algorithm by considering Hop distance, Transmission rate and Attack estimation rate (RPI-HTA). In this scheme, the transmission rate of traffic directed towards the victim node as well as hop distance is considered to determine the best filtering points. SFProbes collect the information of attack estimation rate $R(t_m,i)$ and traffic transmission rate $T(t_m,i)$ during m time slots and send this information to the PBS.

To select the filtering points the attack estimation rate $D(t_m,i)$ computes as follows:

$$D(t_m,i) = \frac{\overline{X}(t_m,i) - \overline{W}(t_m,i)}{\overline{X}(t_m,i)} * \frac{\overline{T}(t_m,i)}{C(v)} * H_i(v) \tag{5.6}$$

$$\overline{D}(t_m) = (1-\alpha)\overline{D}(t_m) + \sum_{i=1}^{n} \frac{D(t_m,i)}{n} \tag{5.7}$$

In Eq. 5.6 we consider the distribution of traffic towards the victim node as an important factor that helps to determine how the attack has followed, where traffic transmission rate $T(t_m, i)$ and capacity of the victim node $C(v)$ are considered. $D(t_m,i)$ in Eq. 5.6 determines if the attack is distributed or centralized. Note that, high value of $D(t_m,i)$ indicates that the attack is centralized whereas a low value of $D(t_m,i)$ denotes a distributed attack. If the attack is highly distributed we need to consider more filtering points to stop the attack whereas if the attack is more centralized we apply filters on few of the routers. The average attack estimation rate is given by $\overline{D}(t_m)$ in Eq. 5.7. We select only those routers whose attack estimation rate $D(t_m,i)$ is higher than average

attack estimation rate $\overline{D}(t_m)$ as filtering points. Thus, in the RPI-HTA algorithm the number of filters to apply depends on traffic transmission rate as well as hop distance.

5.3.4 Packet Filtering

The last step of proposed approach is activating packet filtering at selected points. According to the previous section PBS identify those routes which carry the attack traffic to apply Bloom filtering. So PBS send created Bloom filter to those routers and the routers start to filter incoming traffic toward the victim node. This process continues during the attack event. Table 5.1 summarizes our responsive defense approach by describing the various tasks and the order in which they must be performed from beginning to end. Moreover, the table represents which point is responsible for which task.

In the next section we evaluate the efficiency and accuracy of the two approaches to determine a cooperative responsive mechanism to stop the attack traffic.

Table. 5.1 Tasks in the responsive defense approach

Task#	Responsible point	Task description
1	Campus Network	Establish the identification model and create a history-based profile during the normal traffic condition
2	Campus Network	Create Bloom filter according to task #1
3	Victim Subnet	Send the created Bloom filter to PBS when the attack is detected
4	PBS	Distributing the created Bloom filter to the SFProbes
5	SFProbes	Follow the network traffic and calculate $X(t_m, i)$, $W(t_m, i)$, and $R(t_m, i)$
6	SFProbes	Send the calculated parameters from task #5 to PBS
7	PBS	Analyze the traffic follow toward the victim node based on $D(t_m, i)$, $S(t_m, i)$, $T(t_m, i)$, $C(v)$and $h_i(v)$
8	PBS	Select the best routers to apply filtering
9	Router	Apply Bloom filter to stop the attack

5.4 Evaluation

In this section we demonstrate the performance of our approach using a real network topology from Oregon route-views between March 31 2001 and May 26 2001 [67] and set it up on OPNET© as shown in Figure. 5.2. We test the effectiveness of the responsive defense mechanism using the DARPA 1998 intrusion detection dataset [19] in this experiment. The dataset contains 7 weeks of training datasets that we use to establish an IP address history and 2 weeks of testing dataset to evaluate our techniques. The training part of the DARPA dataset contains normal traffic as well as labeled attacks which helped validate our feature selection work [82] and imported to the OPNET© for our simulation. The first step is creating the IP address history from

the DARPA training dataset and then create corresponding Bloom filter, the details of which were part of our earlier work [82].The next step, which is part of this chapter, evaluates the responsive defense approach based on the created IP address history in OPNET©. We also have validated our model by real network traffic collected at University of Auckland [6] and CAIDA attack dataset [21] in section 5.4.4.

5.4.1 Metrics

Responsive defense mechanism is evaluated on the basis of three parameters, namely, (i) *Attack Traffic Detection Rate* (ii) *Normal Traffic Detection Rate*, and (iii) *Link Utilization Rate*. We now define each of these parameters. (i) *Attack Detection Rate:* Attack traffic detection rate is considered as the percentage of attack dataset that is correctly detected as the attack and cannot pass through the Bloom filter to reach the victim node during the attack time. In the other word, we count total number of packets P_j that belong to attack traffic dataset and cannot pass the Bloom filter and define as True Positive (TP). TP_j shows if the packet number j is correctly detect as the attack or not. Then attack traffic detection rate is defined in Eq. 5.8 according to that where total of TP_j is divided by total number of attack traffic comes to victim node (TP+FN). False Positive Rate, also defined in Eq. 5.9, is the percentage of normal traffic that incorrectly detected as attack and is prohibited from passing the Bloom filter.

The True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) are defined below where k is the total number of incoming packet toward the victim point.

$$TP_j = \begin{cases} 1 & if\ P_j \notin B(v)\ and\ P_j \in |\ attack\ traffic\ | \\ 0 & otherwise \end{cases}$$

$$FN_j = \begin{cases} 1 & if\ P_j \in B(v)\ and\ P_j \in |\ attack\ traffic\ | \\ 0 & otherwise \end{cases}$$

$$FP_j = \begin{cases} 1 & if\ P_j \notin B(v)\ and\ P_j \notin |\ attack\ traffic\ | \\ 0 & otherwise \end{cases}$$

$$TN_j = \begin{cases} 1 & if\ P_j \in B(v)\ and\ P_j \notin |\ attack\ traffic\ | \\ 0 & otherwise \end{cases}$$

$$Attack\ \ Detection\ \ Rate = \frac{\sum_{j=1}^{k} TP_j}{\sum_{j=1}^{k} TP_j + \sum_{j=1}^{k} FN_j} \tag{5.8}$$

$$False\ \ Positive\ Rate = \frac{\sum_{j=1}^{k} FP_j}{\sum_{j=1}^{k} FP_j + \sum_{j=1}^{k} TN_j} \tag{5.9}$$

(ii) *Normal Traffic Detection Rate*: We use the same logic that was used to identify Attack Traffic Detection Rate to compute these parameters. Normal Traffic Detection Rate is defined as the percentage of normal traffic that can correctly pass through the Bloom filter during the attack period. False Negative Rate is defined as the percentage of attack traffic that is incorrectly marked as normal traffic and therefore can pass through the Bloom filter (iii) *Link Utilization Rate*: This is defined as the percentage of the network's bandwidth that is currently being consumed by the network traffic.

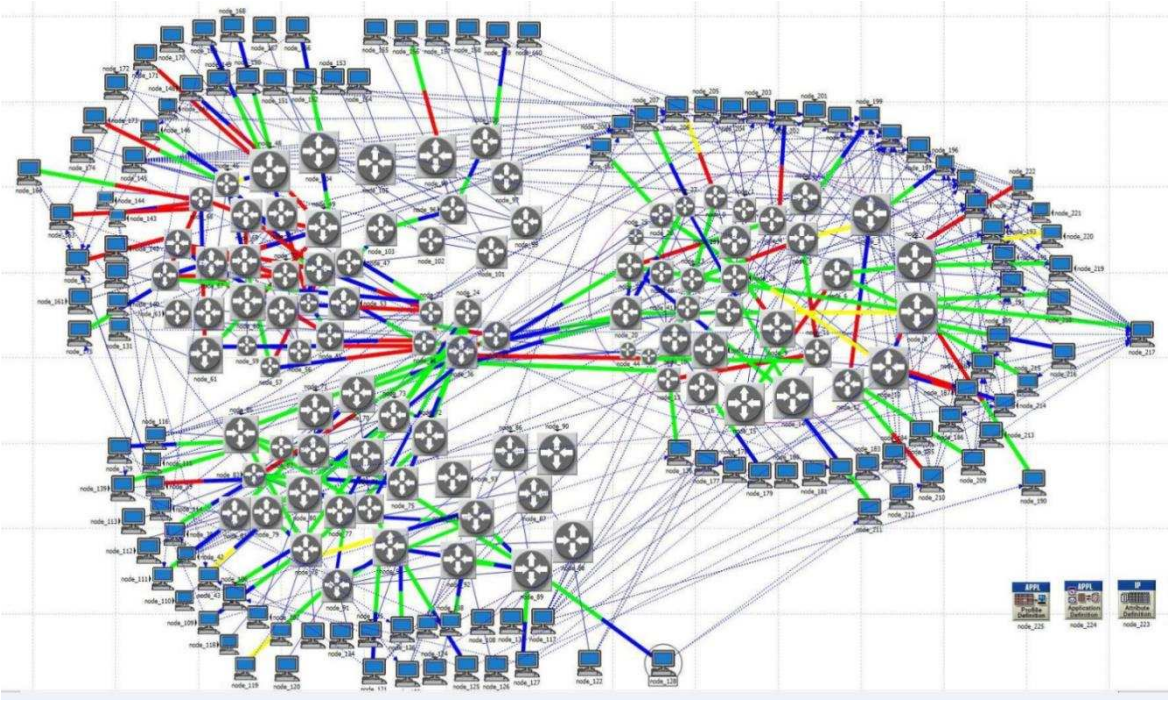


Figure. 5.2. Simulation into OPNET©

5.4.3 Percentage of Collaborative SFProbes

The effectiveness of the responsive defense mechanism relies on the collaboration of SFProbes through the network. Increasing the number of SFProbes attached to the routers enables more close monitoring of the network traffic and providing more effective filtering mechanism. We look at four different scenarios to validate this. We assume a different percentage of routers (80% to 25%) have SFProbes attached to them to monitor network traffic. The number of filtering routers is either fixed according to the RPI-HA or variable based on RPI-HTA algorithm. In the RPI-HA algorithm, the number of filtering routers considered 8, 5 and 3 as 20%, 12.5% and 7.5% of the total routers through the network. We also looked at a fifth scenario where the filters are placed in random locations throughout the network without applying any algorithm to stop the attack traffic. Note that, we considered this scenario in order

to explore how the SFProbe can effectively stop attack traffic and also to evaluate the importance of placing the Bloom filter in the appropriate location.

Figure. 5.3 shows the average attack traffic detection rate of 5 runs and also their standard deviation. We use a time slot of 60 seconds. (The reason for this value is explained in Subsection 5.4.6). The results demonstrate the effectiveness of using the RPI-HTA algorithm, which considers all the three features (hop distance, transformation rate and attack estimation rate) together. The RPI-HTA algorithm produces an attack traffic detection rate of 91% when 80% of routers use SFProbes. Note that, even with 25% probes in the network the RPI-HTA algorithm can stop up to 80% of the attack traffic. This value decreases due to a reduced number of participating SFProbes as expected. Moreover, by applying the random selection to determine placement of filters the attack traffic detection rate reduces by more than 14% and up to 23%. These results show how the location of filters plays an important role in protecting the victim node.

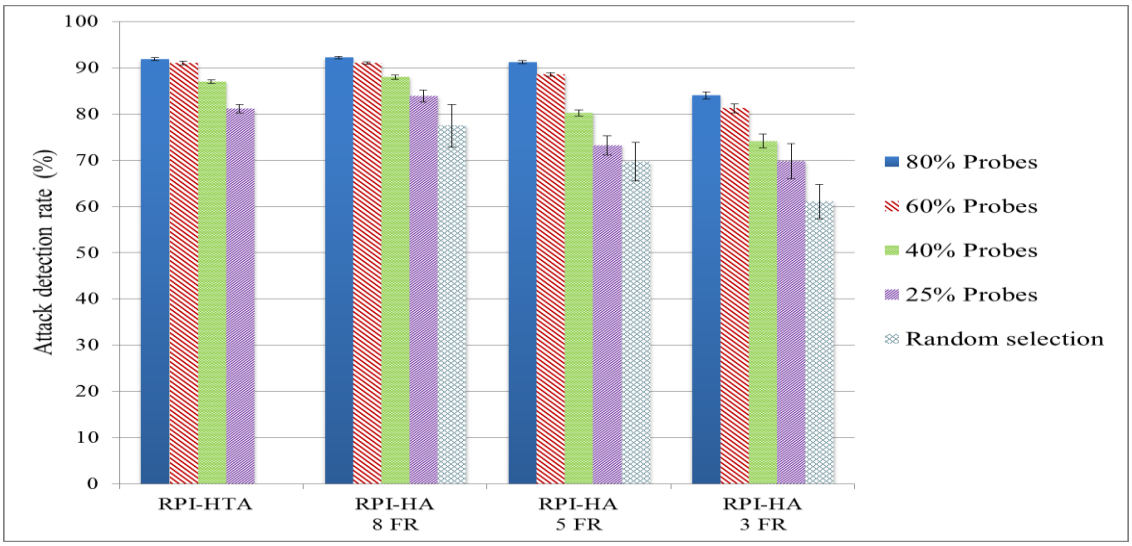


Figure. 5.3. Attack detection rate

Recall that the number of filtering points for RPI-HTA algorithm is variable and depends on attack distribution, transmission rate, and hop distance. 7 filters for networks having 80% and 60% probes and 6 filters for those having 40% and 25% probes are selected according to the RPI-HTA algorithm. As shown in Figure. 5.3 the attack traffic detection rate for RPI-HTA algorithm for 80% and 60% probes is equal or higher than when RPI-HA algorithm is applied by 8 filtering routers through the network. This means that the RPI-HTA algorithm can provide comparable or better filtering mechanism by using lower number of filtering routers by placing the filters in the appropriate locations.

The other important parameter to evaluate is how many normal packets can reach the victim node. As shown in Figure. 5.4, normal detection rate is around 72% for RPI-HTA algorithm with 80% probes and it increases to around 80% if fewer filters are used to stop attack traffic. False negative also was around 3% in this part. Thus, there is a trade off between accuracy of the attack traffic detection rate and the normal detection rate. For instance, by applying RPI-HTA algorithm with 80% probes we have around 90% attack traffic detection rate and at the same time 72% of normal traffic can pass the Bloom filter and reach the victim node. In contrast, applying RPI-HA algorithm and 3 filtering routers with 25% probes can stop 69% of the attack and permit 90% of normal traffic.

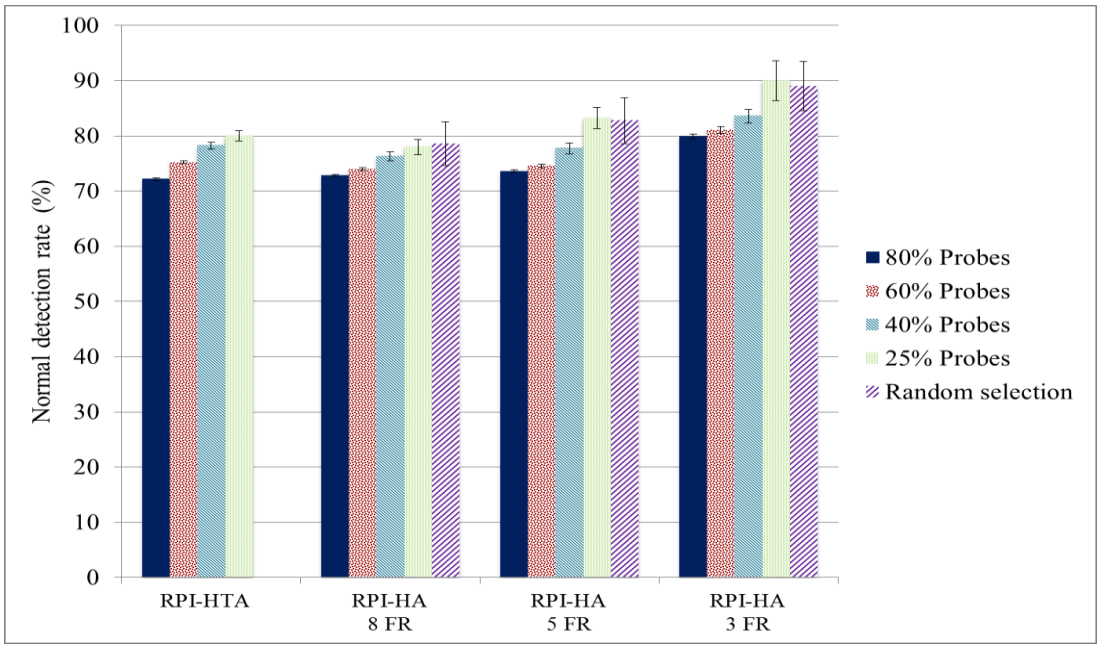


Figure. 5.4. Normal detection rate

5.4.4 Efficiency of Distributed Approach

Figure. 5.5 depicts the fraction of attack traffic dropped at different hop distances from the victim, and thus the source. It shows that 60% of the attacks in total are detected and blocked in the first two routers from attacker, with 23% in the first and 37% in the second. Thus, it shows that RPI-HTA algorithm can effectively select routers further from the victim node and close to attacker. Furthermore, it shows that having more probes is more effective and we can select farther routers as well. For instance with 80% probes all the filtering points are selected at least 3 hop distance away from victim node, while with 25% probes the filtering points must be within 1 and 2 hop distances from victim node.

In this experiment, the 25th, 50th (median), 75th percentiles, minimum and maximum value of attack traffic detection rate for all 3 scenarios are computed as well. The attack traffic detection rate is detailed for each approach. As shown in Figure. 5.6(a), the first layer of router from attacker (fifth hop distance from victim) has relatively good attack traffic detection rate

close to 20% for 50% of simulations in the RPI-HTA algorithm, whereas, this detection rate reduces to less than 10% for random selection scenario. This proves that RPI-HTA algorithm can accurately select desirable filtering points during the attack period and can stop the attack at upstream routers close to the attacker point.

The attack traffic detection rate of 1 and 2 hop distances from victim node is low for RPI-HTA algorithm. It is because the upstream routers have already detected and filtered most attacks. Thus, it can be observed that the core of detection and prevention mechanism is located at the router with hop distance 4, 3, and 5 from victim node in that order. In the other world, all the filtering routers are selected at first three routers from attacker node and the algorithm stop the attack before reach to the routers close to victim node.

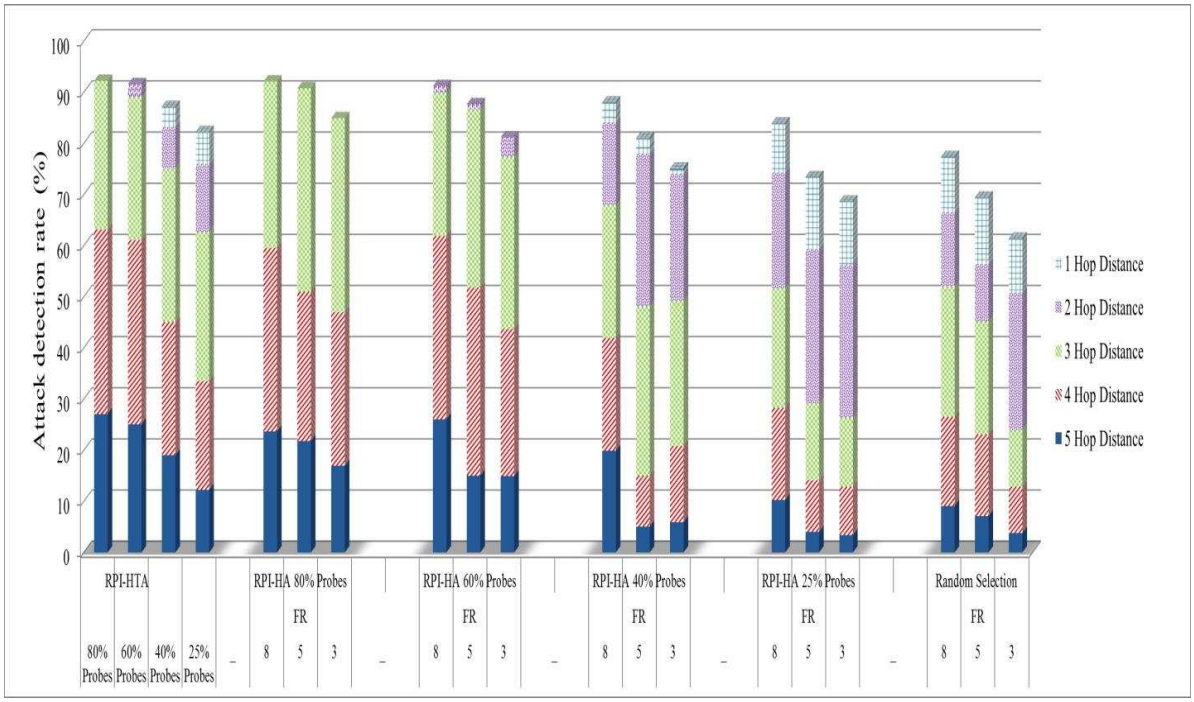
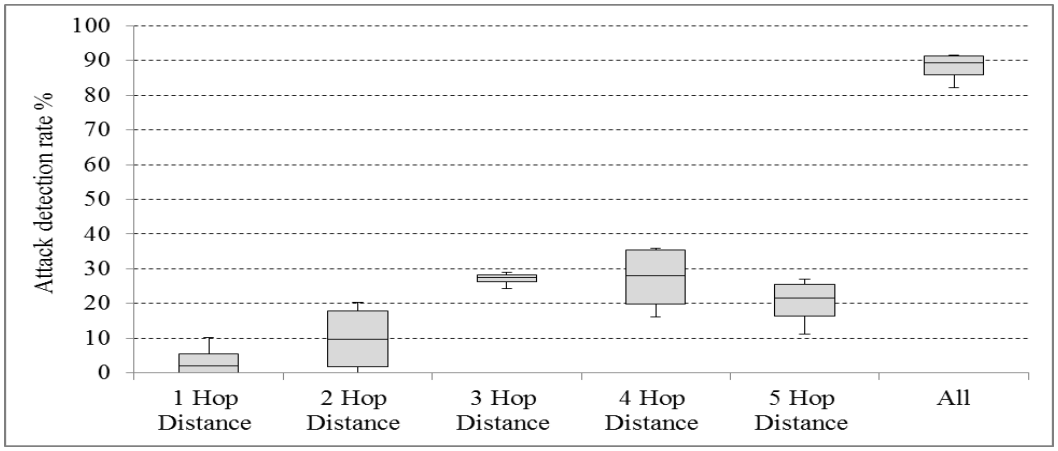
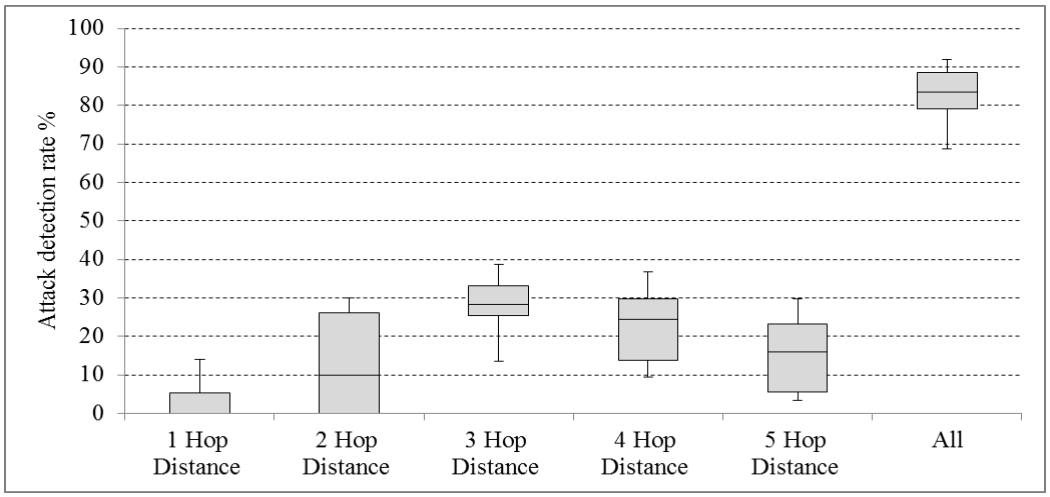


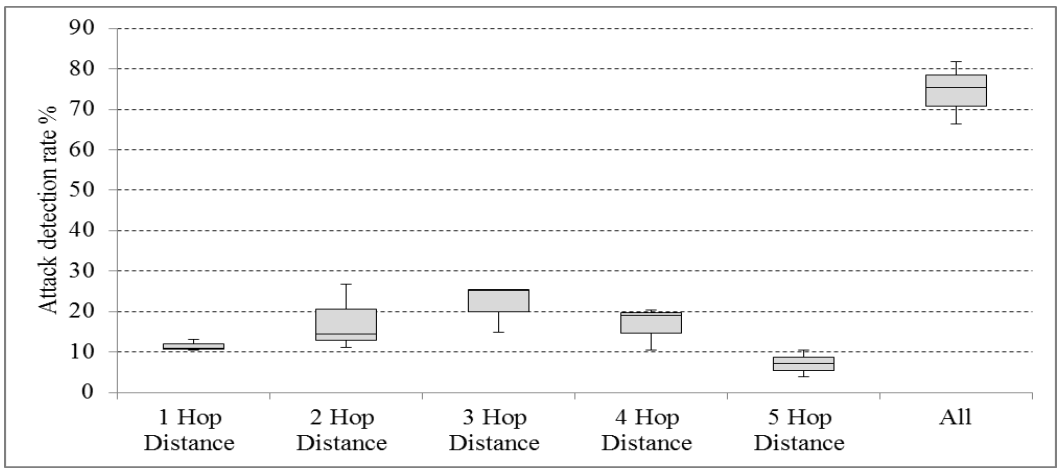
Figure. 5.5. Attack traffic detection rate and location of selected routes



(a)



(b)



(c)

Figure. 5.6. Result of attack traffic detection rate (a) RPI-HTA algorithm. (b) RPI-HA algorithm. (c) random selection

5.4.5 End User's Utilization

The other part of our evaluation is specifying utilization of the victim node and other end-users before and after applying filtering mechanism. In Figure. 5.7, the last link utilization of victim node without applying filtering approach shows that it is fully utilized during the attack time. However, the link utilization reduces to around 60% after deploying Bloom filter through the network based on RPI-HTA Algorithm. The result shows that 80% probes through the network give 50% link utilization rate for the victim node – this is the least link utilization rate that we get in our experiments. Moreover, this result also shows that the attack is detected and is stopped before reaching the victim node.

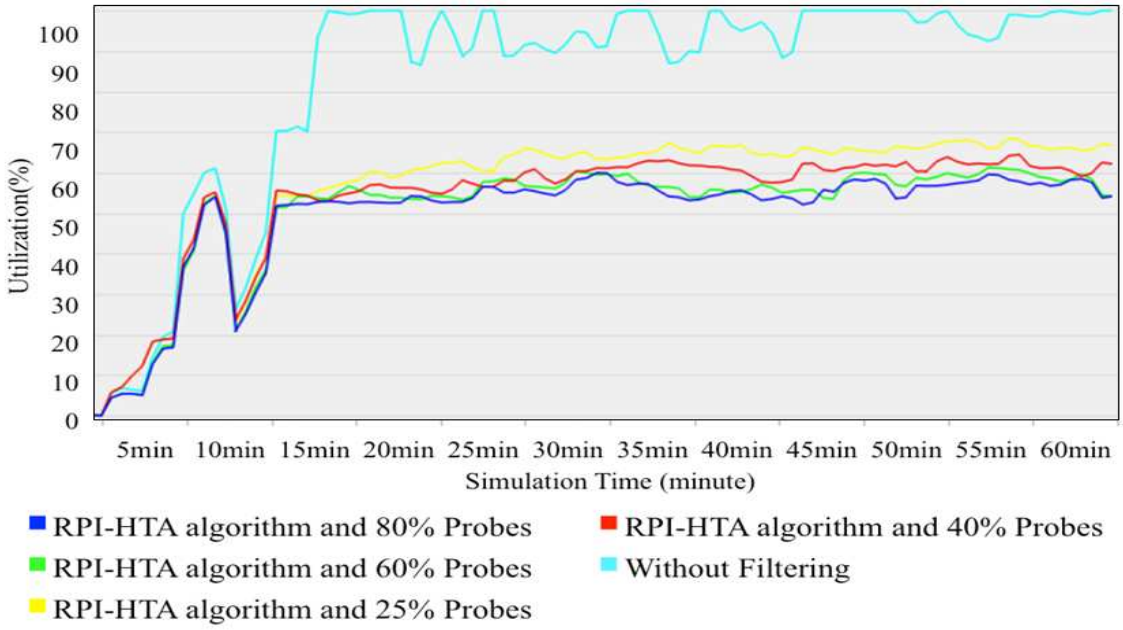


Figure. 5.7. Victim node's utilization for RPI-HTA algorithm

In Figure. 5.8., we compare the last link utilization of the RPI-HTA algorithm that selects 6 points to apply the Bloom filter with that of the RPI-HA algorithm where 8 selected points are chosen to apply bloom filter. It shows similar utilization for both with 80% and 25% probes

through the network. This demonstrates that the RPI-HTA algorithm with lesser number of Bloom filters can determine the best routers to apply the filters and achieve the same result as the RPI-HA algorithm. In the last part of the results, we investigate the effect of the proposed scheme for the last link's utilization of all end-users in the network during the attack time. It is important that the filtering approach can block the DDoS attacks at upstream routers to reduce network congestion during the attack time.

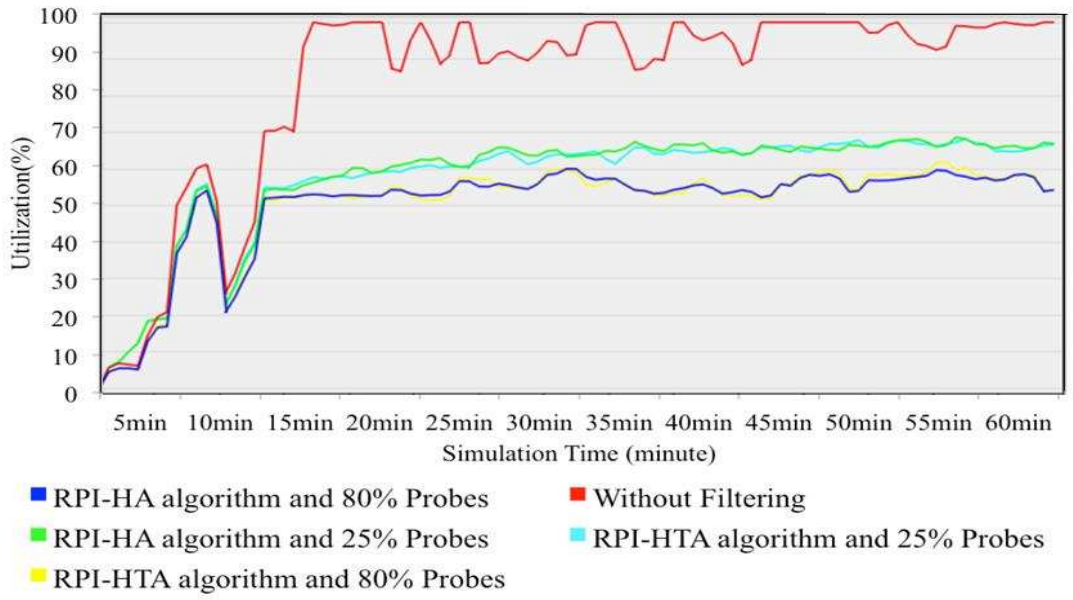


Figure. 5.8. Victim node's utilization comparison

Figure. 5.9 shows the last link's utilization for other end-users increases with applying the Bloom filters. It means the other end-users can receive normal traffic during the attack time. This is good as it provides service availability in the presence of DDoS attacks and minimizes the attack impact during an attack time. Detecting the DDoS attacks at points closer to the source attack protects the victim and other end-users and also saves valuable network resources.

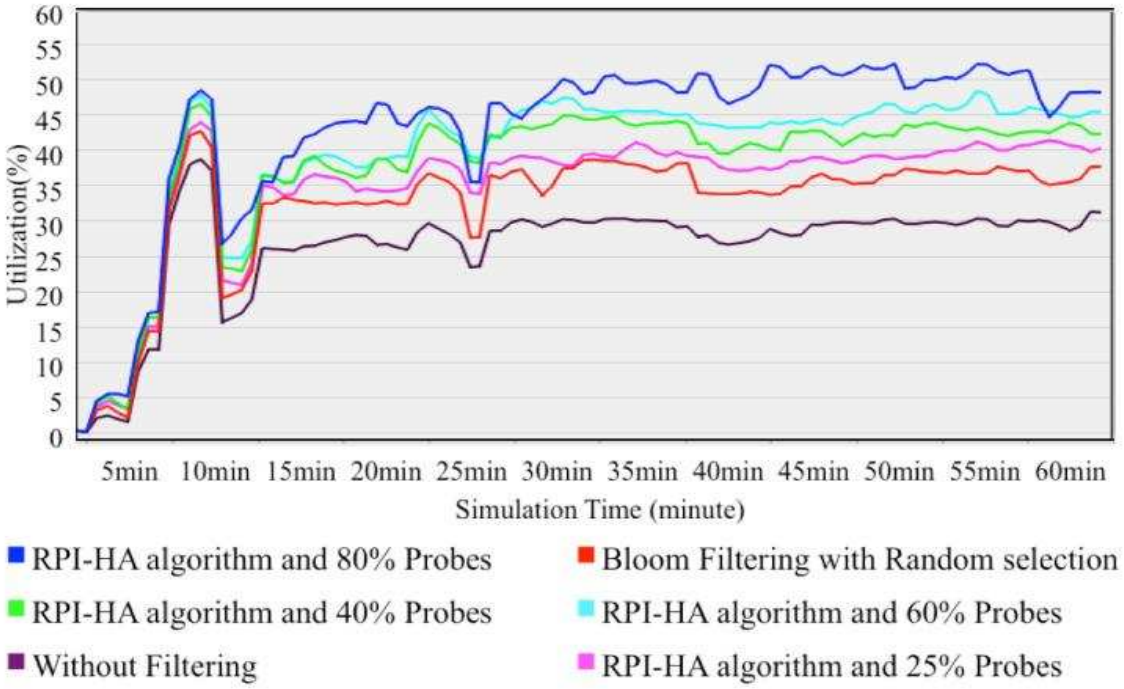


Figure. 5.9. Average of last link utilization of end-users

5.4.6 Impact of Window Size

Figure. 5.10 reports the effect of time slots t_m on attack traffic detection rate where each point represents attack traffic detection rate for a network with 40% probes. When t_m becomes bigger, the SFProbe has longer time to collect and analyze the network traffic. As a result, the algorithm can select filtering point farther from victim node more accurately. For example, as shown in Figure. 5.10 when t_m is 40 seconds, the attack traffic detection rate increases around 5% in 4 hop distance compared to when t_m is 20 seconds.

As shown in Figure. 5.10, attack traffic detection rate is relatively similar when t_m values are 60 and 80 seconds. It means 60 seconds is enough time to evaluate network traffic and increasing the time slot longer will not affect the attack traffic detection rate significantly, which is the value we selected for the other simulation parts as well.

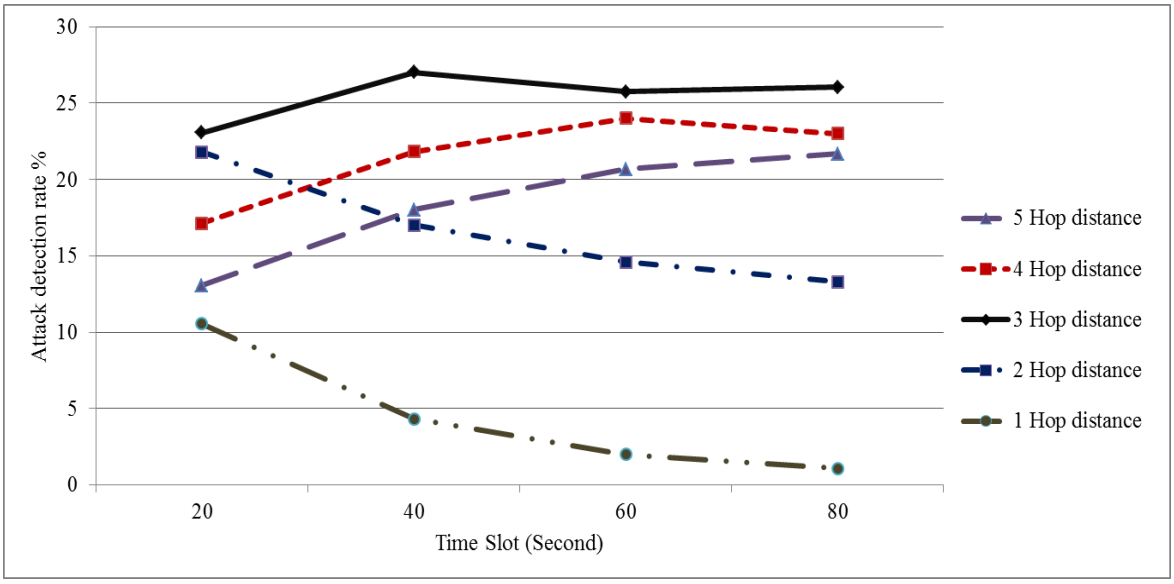


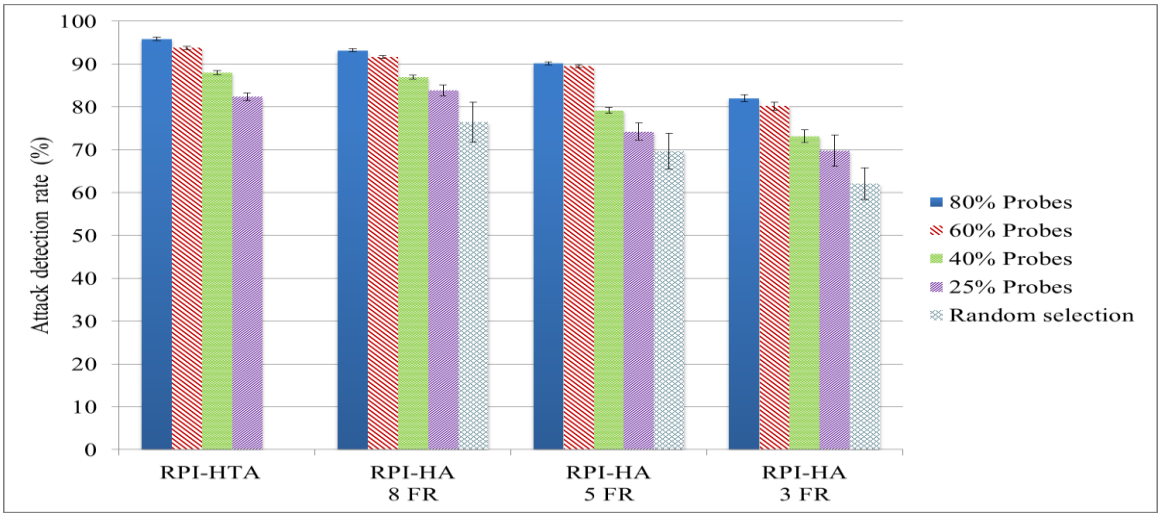
Figure. 5.10. Effect of time slot on attack detection rate

5.4.7 Validation with Real Network Dataset

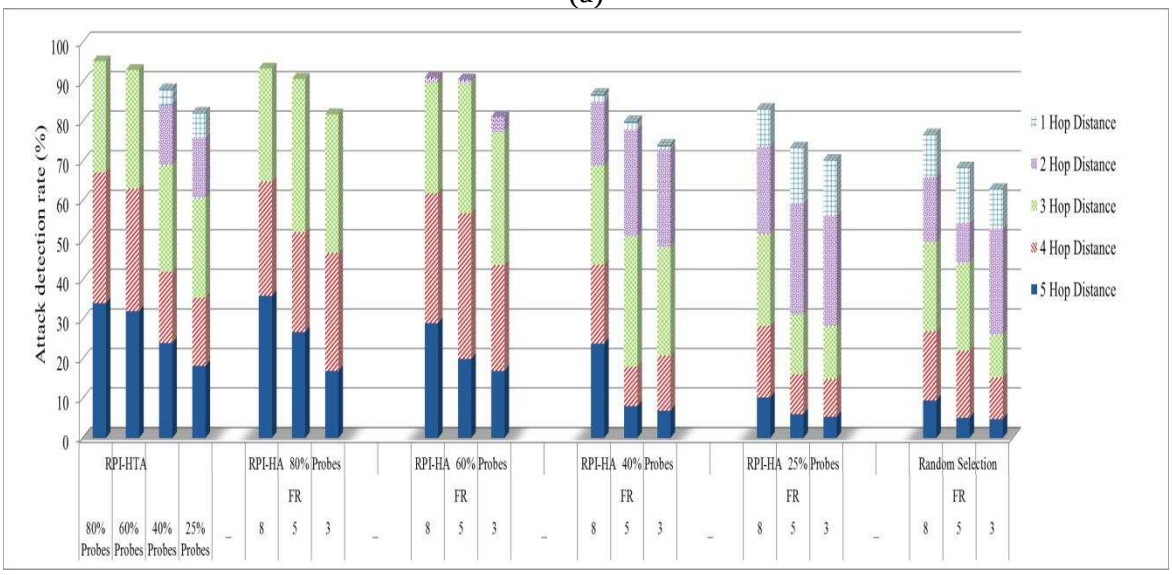
In this experiment, the effectiveness of responsive defense approach is tested using real network trace from University of Auckland in New Zealand. The packet trace contains 6.5 weeks IP header trace taken with 155 Mbps Internet links [6]. The total uncompressed data is 180 GB and all IP addresses have been mapped into 10.*.*.* using one to one hash mapping for privacy. The Auckland traces record three type of IP traffic (TCP, UDP, ICMP). We use "The CAIDA attack dataset 2007" in the experiments as attack traffic. Attack dataset contains approximately one hour of anonymized traffic traces from a DDoS attack on August 4, 2007 that contains 359,656,205 attack packets from 9,066 unique IP addresses [21]. The dataset is run with same topology that was used in previous part by distributing the dataset traffic over the network.

As we discussed earlier, our responsive defense model contains four steps. In the first, history-based profile according to normal traffic is coming to the victim node is created where collected trace from University of Auckland is used at this step. The corresponding Bloom filter will be created based on our scheme [82] in the second step and then responsive Points'

identification approach applies during the attack time. Figure. 5.11 (a) shows the attack traffic detection rate against the CAIDA attack traffic. Attack traffic detection rate is around 95% with 80% probes in the RPI-HTA algorithm where it was around 91% for DARPA dataset. False negative rate for this situation is around 3% as well. Overall the attack traffic detection rate increases slightly compared with DARPA dataset, which the Bloom filter accuracy played a role in this situation.



(a)



(b)

Figure. 5.11. (a) attack traffic detection rate (b) location of selected routers

Figure. 5.11 (b) depicts the fraction of attack traffic dropped at different hop distances from the victim. Similar to DARPA dataset, it shows that RPI-HTA algorithm can effectively select routers further from the victim node and close to attacker where all the filtering routers with 80% probes are selected from first three routers from attacker.

In Fig 5.12 The 25^{th}, 50^{th} (median), 75^{th} percentiles, minimum and maximum value of attack traffic detection rate of RPI-HTA algorithm shown for CAIDA dataset traffic that can be compared with Fig 5.6 (a) for DARPA dataset as well. As shown for 75% of simulations the most portion of attack is detected and block at the first layer of routers from attacker (hop distance 5 from victim node) followed by the one in the second layer of routers from attacker (4 hop distance for victim). Compared with DARPA dataset the router with fifth hop distance from victim has a better attack traffic detection rate close to 25% for 50% of simulations in the RPI-HTA algorithm, where this detection rate was 20% for DARPA dataset. Moreover, it shows that router with hop distance 5, 4 and 3 in that order were the core of attack detection. It means the most of the attack stop before reach to victim node and effect through the network. This proves that RPI-HTA algorithm accurately selected desirable filtering points during the attack period for CAIDA attack traffic as well.

As shown in Figure. 5.13, normal detection rate also is around 77% for the RPI-HTA algorithm when the percentage of participated SFProbes is 80%. It can be noticed that this value increases by 5% compared with DARPA dataset. As we discussed, this value increases as the number of filters or participated SFProbes decrease through the network.

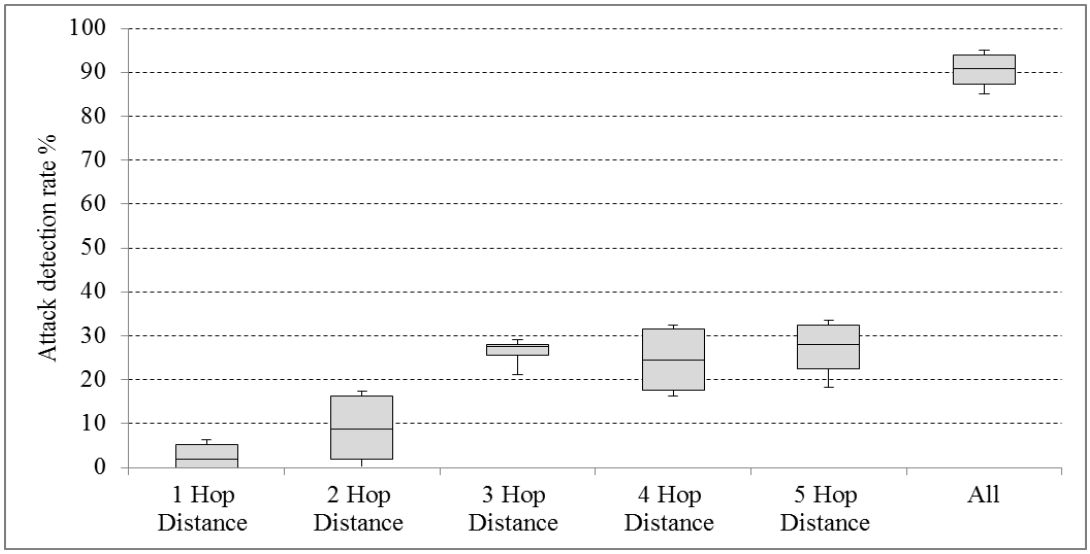


Figure. 5.12. Attack traffic detection rate for CAIDA attack traffic

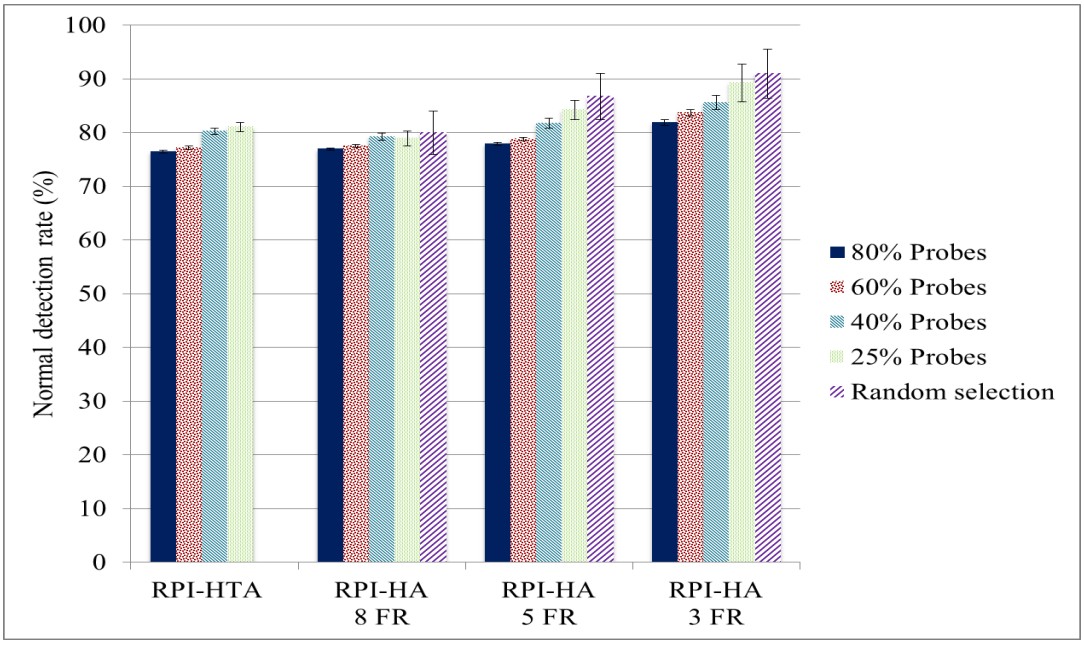


Figure. 5.13. Normal detection rate

5.4.8 Complexity and Processing Overhead

In this section, we discuss the communication and computation overhead of network resources in our scheme. The communication overhead of our scheme involves:

1. According to our algorithm, communication starts between the victim node and PBS when the attack is detected; therefore, there is no communication overhead for the network prior the

attack. When the attack is detected PBS sending the Bloom filter to the SFProbes and the traffic collected in the SFProbes is sent back to the PBS. The number of messages sent in this case depends on the number of participating SFProbes. In our simulation, the percentage of routers having SFProbes attached varies between 80% to 25% and the number of messages exchanged in this case is between 20 to 72.

2. PBS sends the Bloom filter to appropriate routers for filtering. In this case, the number of messages depends on the number of selected routers. For example, the RPI-HTA algorithm selected at most 8 filtering routers – the communication overhead in this case is negligible compared to the traffic going through the network.

The computation overhead of our scheme is also minimal. The SFProbes collect data without disrupting the original traffic, so there is no extra processing overhead on network resources for data collection. Moreover, the PBS makes the filtering decision and so no computation overhead is imposed on the network resources. The following is the computation overhead of our scheme.

The Bloom filter is applied in selected routers during the attack time. The overhead of filtering on the upstream routers is as low as computing hash value of incoming IP address that checks whether IP address is a member of the filter or not (in our experiment a few millisecond). We used MD5 algorithm that is a widely used cryptographic hash function producing a 128-bit hash value. It contains addition, bitwise, shift, and rotation operation where the time complexity is *O(n)*. Moreover, the other type of hash functions such as SHA-2 or SHA-3 can be applied for security and collision resistant purposes. In addition, the number of hash functions k in Bloom filter depends on false positive p given by following equation.

$$f = (1 - e^{-kn/m})^k = (1 - p)^k \tag{5.10}$$

There are three fundamental performance metrics for Bloom filters: the probability of error (corresponding to the false positive rate *f*), size of the Bloom filter array (corresponding to the array size *m*) and the number of hash function *k*. Those parameters can justify based on accepted false positive. In analysis, the false positive rate is limited to 0.1. In order to achieve this false positive rate 4 hash functions require providing a Bloom filter from the IP address history. Moreover, the Bloom filter array is 5 times the number of IP addresses kept in the history. According to our algorithm the Bloom filter is created at victim node prior to the attack; therefore, there is no computation overhead at victim node during the attack time. The only computation overhead is applied on selected routers during the attack to computing hash value of incoming IP address where MD5 algorithm is used and takes a few millisecond to check whether IP address is a member of the filter or not. Note that, the Bloom filter is applied to only a select few of the upstream routers so processing overhead is also minimized.

Different cooperative techniques have been proposed to coordinate attack detection and respond to it [16], [23], [25], [26]. However, most have a large overhead of network communication and resources as briefly compare in Table 2. Moreover, communication during attack time must be minimized to the extent possible, as there is lack of bandwidth. Our scheme alleviates such problems. As a comparison, TDFA [26] using CAIDA attack dataset in evaluation and succeeds in lowering the attack traffic by 88% with traceback mechanism with more than 1000 to 4000 messages passing after detecting the attack to reduces the rate of attack packets whereas our scheme lowers the attack by 95% with very low overhead cost. The total number of message passing is less than 100 in our simulation setup as discussed above. Moreover, in general the traceback mechanism has high computational and network overhead regarding to mark the packets and collect sufficient number of packets to trace back up the

attacker where the number of ISP involvement to support traceback is important parameter and makes it difficult to be deployable. Compared with recent cooperative approaches, our scheme reduces processing overhead as discussed while maintaining the Quality of Service (QoS) for legitimate traffic during the attack period and making the scheme more deployable.

Table 5.2. Comparison of distributed defense approach

Technique	**Features**	**Defense limitation**	**Scalability**	**Implementation complexity**
RPI_HTA	Identifies the most effective points to place filters so as to minimize attack traffic and maximize legitimate traffic	Plug in SFProbes into the routers to support approach	High	Low
TDFA [26]	A traceback based defense system to reduces the rate of forwarding the attack packets	Support sufficient number of routers to be effective / High communication overhead	Medium	High
D_WARD [16]	Gathers two-way traffic statistics and applies a rate limiting	Universal deployment/Performance degradation/high Collateral damage	Low	Medium
FireCol [25]	IPSs form virtual protection rings around the hosts and collaborate by exchanging traffic information	Overhead and performance cost for routers and rings configuration	Low	High

5.5 Conclusion

In this chapter we proposed a responsive defense approach to defend against denial of service attacks. We introduce a cooperative mechanism that specifies best response points where filters can be placed so as to minimize attack traffic and maximize legitimate traffic during the attack.

Our approach has been validated on real-world data sets. Our experiments indicate that our responsive model can protect the victim node from 90% of attack traffic while allowing 70% of legitimate traffic with light computational as well as communication overhead. In addition, the results show how our model can preserve valuable network resource and increase link utilization of other end-users during the attack time to maximally preserve the service availability and minimizing the attack impact.

CHAPTER 6

PACKET-PAIR DISPERSION SIGNATURES IN MULTIHOP NETWORKS

6.1 Introduction

In this chapter we consider packet-pair technique as a well-known mechanism for characterizing end-to-end network paths. An analytical model is presented for the packet-pair based signature of multi-hop network paths. The model does not require a single bottleneck link assumption, and it accurately describes the behavior of packet-pairs. The analytical relationship between the input and the output gaps of packet pairs is derived, and the corresponding distribution of end-to-end packet-pair dispersion is used to derive the path signature, which characterizes the path. The model is verified via OPNET® based simulations. We also analyze how the signature is shaped by factors such as the number of the hubs, initial dispersion and cross traffic. The analytical model provides deeper insights to path signatures, thus enabling efficient and accurate network monitoring, problem diagnosis, and estimation of parameters such as end-to-end network bandwidths and link capacities.

Network monitoring, tomography, and overlay based QoS provisioning are among network operations and applications that rely on the use of end-to-end path measurements to infer operational conditions of network paths. Dispersion of packet pairs or packet trains as they traverse is the basis of many inference tools used for characterization of network paths [84], [85], [86], [87], [88], [89], [90], [91]. Packet-pair technique is used to obtain crucial network properties such as bottleneck capacity, available bandwidth and common congestion links in a wide array of network monitoring tools [92], [93], [31], [94], [29], [95], [96], [97], [98], [99], [100], [101]. A packet-pair typically is two equal-length packets sent with an initial dispersion.

By the time the packet-pair reaches the destination, the final dispersion is changed due to link characteristics and cross traffic along the probing path. Cross traffic refers to all the traffic on the path except for that due to probing packets. Packet-pair technique essentially estimates the network parameters, e.g., the end-to-end network bandwidth and the bottleneck link capacity, from the relationship between the initial and final dispersions.

Packet-pair dispersions can be used to generate path signatures. Dispersion fingerprint [102] uses the Cumulative Distribution Function (CDF) of packet-pair dispersion as the path signature. Internet path signatures are distinct, in general, and they persist over periods of time [102], [103], [104], [105]. Thus the signatures are used for distinguishing paths from one another, monitoring networks, diagnosing problems, developing deeper understanding of network behavior, testing protocols for realistic network conditions, and for determining if two paths share common links. An accurate model of the packet-pair technique is important not only for creating accurate path signatures for different scenarios, but also for enhancing the accuracy and efficiency of measurement techniques for parameter estimation. The main contribution of our scheme is the derivation of a model for the packet-pair technique for multihop paths with multiple tight links, thus more accurately capturing the stochastic nature of cross traffic and its interaction with packet-pairs. In contrast, the existing stochastic delay model, presented in [29], describes the analytical relationship between the input and output dispersions under the assumption of a single tight link. We validate the proposed model via OPNET® simulations and discuss the accuracy of the proposed model. We investigate the effects due to initial dispersion, cross traffic and the number of the hops on end-to-end signatures using the analytical model. We also show how link properties such as available bandwidth, arrival rate of cross traffic can be estimated based on link signatures. In addition, our results are directly applicable for deriving the path signature when

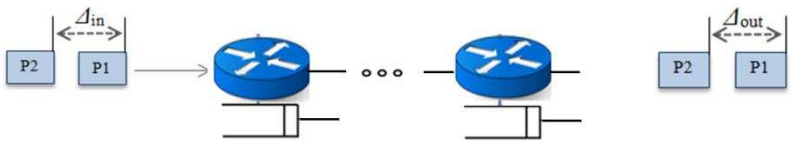
measurements are done using packet-pair with variable packet gaps. This is the case with passive techniques that rely on existing network traffic, thus not consuming network resources for measurements [31].

Section 6.2 reviews the packet-pair technique and the packet-pair delay model for the single-hop case. Section 6.3 presents the queuing model to generate path signatures for the multi-hop case and determine path characterization. This model is validated through OPNET® simulations, and the impact of network and traffic characteristics on signature is evaluated in Section 6.4. Finally, the conclusion follows in Section 6.5.

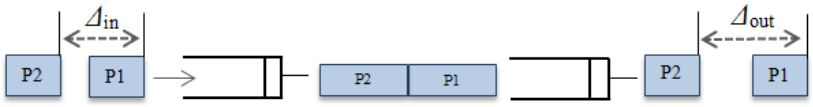
6.2 Packet-Pair Delay Model

In this section, the packet-pair technique is described and the single-hop stochastic model of packet-pair technique proposed in [29] is outlined. The packet-pair technique typically employs two equal sized packets that a source sends to a receiver as shown in Figure. 6.1. The initial dispersion Δ_{in} at the sender is defined as the interval between the departure of the first bit of the first probing packet P_1 and the first bit of the second probing packet P_2 from the sender [102]. The dispersion between these two packets changes according to the path characteristics such as link capacities and cross traffic. The final dispersion Δ_{out} at the receiver is defined as the interval between the arrivals of the first bits of the first and second packets. The main goal is to establish an analytical relationship between Δ_{out} and Δ_{in}. Figure. 6.1 shows how parameters such as network link capacities and cross traffic affect packet-pair dispersion. The dispersion may increase due to multiple causes. Case (b) shows the final dispersion increasing as the packets move from a high bandwidth link to a low bandwidth link, while in Case (c) the final dispersion increase is due to one or more cross traffic packets that come between the packet-pair. In Case

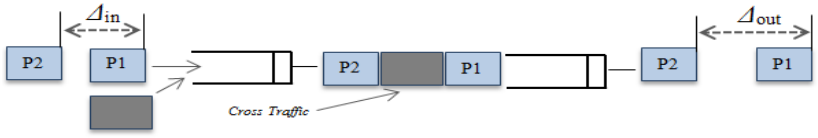
(d), one or more packets are queued before the first packet and the packet-pair experiences compression due to queuing in the busy link [92]. At the end of a typical path, the final dispersion is affected by a combination of these cases through the entire path. This example illustrates that the network can either increase or decrease the dispersion due to cross traffic rate and link capacity.



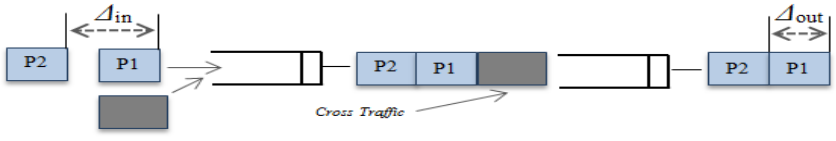
(a) An example network with two routers



(b) Packet-pair from moves from high to low bandwidth $\Delta_{in} < \Delta_{out}$



(c) Cross traffic comes between packet-pair $\Delta_{in} < \Delta_{out}$



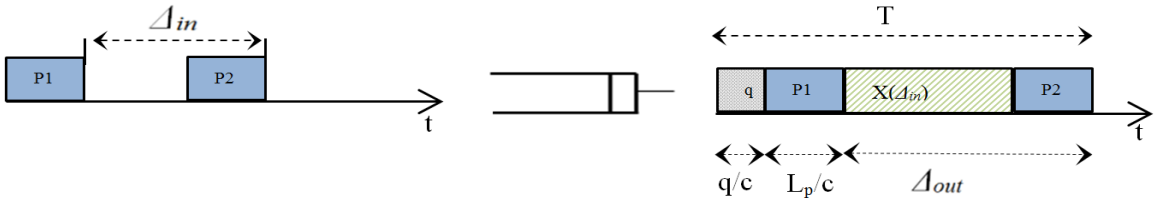
(d) Cross traffic comes before the first packet $\Delta_{in} > \Delta_{out}$

Figure. 6.1. Effect of traffic and network links on packet-pair dispersion

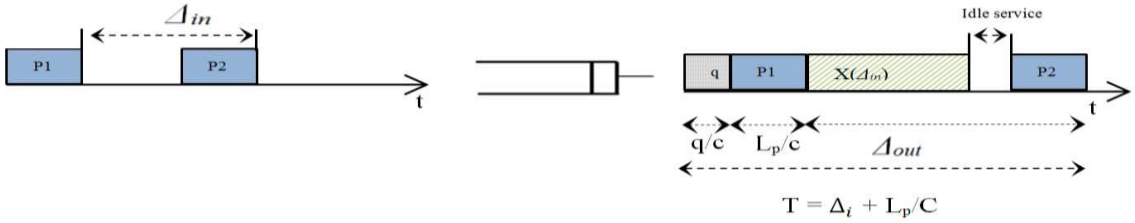
Next we outline the model in [29] that established a relationship between Δ_{in} and Δ_{out} for a packet-pair over a single hop. Let the capacity of this link be C and the length of each of the packets in the pair be L_p. Let t_0 be the time at which the first packet of the pair arrives at the queue, q be the total of cross traffic already in the queue, and T indicates the interval between the arrival of the first packet and the departure of the second packet. Figure. 6.2 shows the packet-pair in a single link for busy and idle server conditions [29]. In Figure. 6.2, $X(t_0, \Delta_{in})$ denotes the total of cross traffic that arrived during $[t_0, t_0+\Delta_{in}]$. In this figure instead of $X(t_0, \Delta_{in})$, $X(\Delta_{in})$ is used as it is independent of t_0 in fluid assumption. By considering a constant rate r, $X(t_0, \Delta_{in})$ generates $r\Delta_{in}$ amount of cross traffic, and Δ_{in} and Δ_{out} are related to each other according to following equation depending on whether the link is being fully utilized or not [86]:

$$\Delta_{out} = \begin{cases} \frac{r}{c}\Delta_{in} + \frac{L_p}{c} & \Delta_{out} \le \Delta^* \\ \Delta_{in} - \frac{q}{c} & otherwise \end{cases} \qquad (6.1)$$

$$where\ \Delta^* = \frac{L_p + q}{C - r}$$



(a) The Server busy when the second packet arrives



(b) The Server idle when the second packet arrives

Figure. 6.2. Packet-pair behavior in a single link with fluid cross-traffic

The assumption of fluid cross traffic with constant rate introduces a considerable error in this model. In contrast, $X(t_0, \Delta_{in})$ is considered as a stochastic process in following, and our model depicts the relationship between Δ_{in} and Δ_{out} more accurately, even in the multi-hop case. However, first consider the rationale as to how the behavior of a packet-pair can be used in the transient queuing analysis with the M/D/1 model in [29] for the single hop case.

The queuing model in [29] assumes Poisson cross traffic. When the first packet pair arrives at the queue at t_0, there are q_1 packets already in the queue and one in service with residual service time d. Assume similar way for the second packet pair that there are q_2 packets already in the queue when the second packet pair enters in the queue. Consider W_1 and W_2 as the waiting time of the first and second packet pair when the service times of the both packet pairs equal to L_P/C [14]:

$$\Delta_{out} - \Delta_{in} = \left(W_2 + \frac{L_p}{C} \right) - \left(W_1 + \frac{L_p}{C} \right) = W_2 - W_1 \tag{6.2}$$

The logic is that the system can be considered as an M/D/1 queue model which starts to evolve at $t = t_0$ with a total of $N_0 = q_1 + L_p/L_c + 1$ in the system, including possibility of service time where the first packet pair arrives at $t = t_0^+$. Then, the waiting time W_2 of the second packet pair entirely corresponds to a packet that arrives at the M/D/1 system at $t = \Delta_{in}$ [29]. In this model, to provide the waiting time distribution of the second packet pair, first $\pi_j(t)$ - the probability of the system holding j packets at time t indicate and then the state vector $\pi(t) =: (\pi_0(t), \pi_1(t), \pi_2(t), \ldots)$ was obtained [29].

The distribution of $W(\Delta_{in})$ for any given value of Δ_{in} is found in Eq. 6.3. This equation corresponds to the waiting time distribution of the second packet pair in the queuing model. Then it is straightforward to obtain the relationship between final dispersion Δ_{out} and initial dispersion Δ_{in} from Eq. 6.2. The waiting time distribution $P\ (W(\Delta_{in}) < kD - v)$ for all $k \in N$ and $v \in (0, D]$, can be determined by considering the queuing position of the packet at $t = \Delta_{in} + D - v$, and the random variable $N_{\Delta in}(t)$ was defined as the number of packets arriving before Δ_{in} that are still in the system at t. The cumulative distribution function (CDF) of the waiting time $W(\Delta_{in})$ $F_{\Delta in}(x) = P\{W(\Delta_{in}) < x\}$ is as follows [14]:

$$F_{\Delta in}(x) = \begin{cases} \sum_{j=0}^{\left\lfloor \frac{x}{D} \right\rfloor - N_0 + K(y+D)} \frac{(\lambda * \Delta_{in})^j}{j!} e^{-(\lambda * \Delta_{in})} & y \le 0 \\ \sum_{j=0}^{\left\lfloor \frac{x}{D} \right\rfloor} Q_{\left\lfloor \frac{x}{D} \right\rfloor - j}(y) \frac{(\lambda * z)}{j!} e^{-(\lambda * z)} & otherwise \end{cases} \tag{6.3}$$

Here, $y := \Delta_{in} - D + (x \bmod D)$, $z := D - (x \bmod D)$, and $Q_m(t) := \sum_{i=0}^{m+1} \pi_i(t)$. λ is the packet arrival rate of cross traffic and $K(t)$ is the number of packets that have left the system by t. To compute $Q_m(t)$ in our model we use above equation; however, the transition matrix P to determine $\pi_i(t)$ was corrected based on the proof results in [88] as follows. Here, function $F_{\Delta in}(x)$ provides a delay model for packet-pair in the single-hop case. The Cumulative Distribution Function (CDF) of the waiting time $W(\Delta in)$, $F_{\Delta in}(x)$, corresponds to the waiting time distribution of the second packet in the pair that we will consider in our model.

$$P = \begin{pmatrix} e^{-\lambda D} & \lambda D e^{-\lambda D} & \ldots & (\frac{\lambda D}{j!}) e^{-\lambda D} & \ldots \\ e^{-\lambda D} & \lambda D e^{-\lambda D} & \ldots & (\frac{\lambda D}{j!})^{j} e^{-\lambda D} & \ldots \\ 0 & e^{-\lambda D} & \ldots & (\frac{\lambda D}{(j-1)!})^{j-1} e^{-\lambda D} & \ldots \\ 0 & 0 & e^{-\lambda D} & \ldots & (\frac{\lambda D}{(j)!})^{j} e^{-\lambda D} \ldots \\ . & \ldots & \ldots & \ldots & \ldots \\ . & \ldots & & & \end{pmatrix} \quad (6.4)$$

6.3 Queuing Model for Multihop Case

In this section, derives the general delay model to determine relationship between dispersion of packet pair at all intermediate links in a multihop path. First, we give our rationale as to how the behavior of the packet pair dispersion can be described and the main challenge in handling packet-pairs with multihop queuing model. Then, we develop a stochastic delay model to describe the mathematical relationship between input and output dispersions of packet-pairs.

$F_{\Delta in}(x)$ provides a delay model to determine the dispersion of the packet-pair for a specific Δ_{in} in a single link or a path with a single tight link. In multihop case, the gap between the packet-pairs entering a link is no longer a known constant value. The path signature in fact represents the net effect due to these variable packet dispersions at all the intermediate links. Therefore, a challenge in deriving a general multihop queuing model is to determine the initial dispersion of the pair entering link i, $\Delta_{in}(i)$. As in multihop model Δ_{out} from the link i is the Δ_{in} for link $i+1$, the single link model $F_{\Delta in}(x)=P\{W(\Delta_{in})<x\}$ can provide the distribution of dispersion only for a specific Δ_{in}. Simply chaining multiple instances of single link dispersions together does not work, as it requires that at the junction between two links, the packet gap be reset to a known fixed Δ_{in}. However, we have a different Δ_{in} as the input for each hop for each packet-pair. To determine a general multihop model, consider M links in an end-to-end path as in Figure. 6.3. Let C_i and λ_i

denote the link capacity and the packet arrival rate of cross traffic respectively of the i^{th} link, $1 \le i < M$. Furthermore, let $\Delta_{in}(i)$ denote the initial dispersion of a packet-pair at the i^{th} link. The Cumulative Distribution Function (CDF) of the waiting time $W(\Delta_{in}(i))$, for i^{th} link, $F_{\Delta in}(x, i)=P\{W(\Delta_{in}(i))<x\}$ is as follows:

$$F_{\Delta_{in}}(x,i)=\begin{cases}\sum_{j=0}^{\left\lfloor \frac{x}{D} \right\rfloor - N_0 + K(y+D)} \frac{(\lambda_i * \Delta_{in}(i))^j}{j!} e^{-(\lambda_i * \Delta_{in}(i))} & y \le 0 \\ \sum_{j=0}^{\left\lfloor \frac{x}{D} \right\rfloor} Q_{\left\lfloor \frac{x}{D} \right\rfloor - j} \frac{(\lambda_i * z)^y}{j!} e^{-(\lambda_i * z)} & otherwise \end{cases} \quad (6.5)$$

Where, $y: = \Delta_{in}(i) - D + (x \bmod D)$, $z: = D - (x \bmod D)$, and $Q_m(t,i) := \sum_{n=0}^{m+1} \pi_n(t,i)$. We can determine $\Delta_{out}(i)$ similar to the single hop case from Eq. 6.2, but with W_2 and W_1 replace by $W_2(i)$ and $W_1(i)$ for the given packet pair.

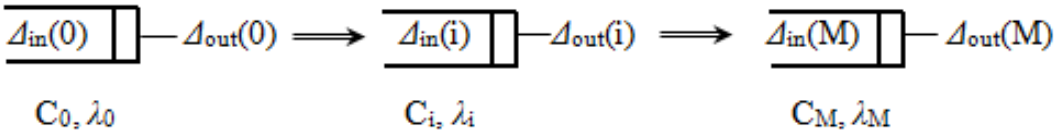


Figure. 6.3. Multihop queuing system model

Let $S_{\Delta in}(r_i, i)$ denote the signature of link i for a given δ_{in}:

$$S_{\Delta_{in}}(r_i, i \mid \delta_{in}) = F_{\delta_{in}}(x,i) \quad (6.6)$$

where $r_i = \delta_{in}(i) + x + W_1(q_1, i)$

$S_{\Delta in}$ is calculated for each given $\Delta_{in}(i) < r_{i-1}$, and r covers the range of values for $\Delta_{out}(i)$ in i^{th} link where x determines the range of distribution values for $W_2(i)$ in $F_{\Delta in}(x, i)$ and we need to obtain the range of values for $\Delta_{out}(i)$ according to Eq. 6.6.

As $\Delta_{out}(i)$, the dispersion at the output of link for a packet-pair is the inter-arrival time $\Delta_{in}(i+1)$ for the $(i+1)^{th}$ link, we can compute the probability of each inter-arrival time $\Delta_{in}(i+1)$at link $i+1$ by using CDF of the $\Delta_{out}(i)$ given by Eq. 6.6. CDF of $\Delta_{out}(i+1)$ can then be generated by multiplying $S_{\Delta in}(r_i, i)$ for a specific Δ_{in} by probability of separation Δ_{in} in the current link. Probability of a specific point can be found from CDF distribution function as we consider in Eq. 6.7, where, $P_{\Delta in}(r_i,i)$ is defined as the Cumulative Distribution Function (CDF) of the $\Delta_{out}(i)$ in the queuing model for a multihop case:

$$P_{\Delta in}(r_i, i \mid \delta_{in} < r_{i-1}) =$$

$$\begin{cases} \int_0^{\infty} S_{\Delta in}(r_i, i) * (P_{\Delta in}(r_i, i-1) - P_{\Delta in}(r_i^-, i-1)) dr & i \geq 1 \\ S_{\Delta in}(r_i, i) & i = 0 \end{cases} \tag{6.7}$$

For example, consider a three hop path, i.e., $M=2$ in Figure. 6.3. First $F_{\Delta in}(x,0)$ and r_0 are computed and $F_{\Delta in}(x,0)$ is assigned to $S_{\Delta in}(r_i, i \mid \delta_{in})$ as well. At this step we have the CDF of $\Delta_{out}(0)$ $F_{\Delta in}(x,0)$, which also corresponds to CDF of $\Delta_{in}(1)$. To find the delay dispersion of next link the delay dispersion for all $\Delta_{in}(1)$ that is in the range of $0<\Delta_{in}(1)< r_0$ will be computed. Finally, we can find the delay distribution of second packet-pair from Eq. 6.7 and then $\Delta_{out}(1)$ from Eq. 6.2.

Therefore, by computing $P_{\Delta in}(r_i,i)$ for link i we can determine the signature of link i, and use this signature to create the next link's signature. In the other words, initially the packets are generated in the first link using the packet-pair delay model of Eq. 6.5 for specific C_i, λ_i, Δ_{in}. Then, as the output packets of one link become the arrival packets for the next link, we obtain the probability of each inter-arrival time Δ_{in} of packet pair by using the previous link's delay

dispersion model. Therefore, in the next link we generate the packet-pair dispersion model $S_{\Delta in}(r_i, i)$ for each inter-arrival time Δ_{in} in the current link and compute the path signature according to $S_{\Delta in}(r_i, i)$ and probability of each inter-arrival time Δ_{in} at link i.

6.3.1 Available Bandwidth Estimation

By establishing a dispersion model for packet pair in a multi hop model we can find the other useful properties of the path. Here, we estimate the available bandwidth based on the proposed model. The available bandwidth of a path is the rate at which additional data can be sent through the path.

For a given value of $\Delta_{in}(0)$, M, q_1 and $N_0 = q_1 + L_p/L_c + 1$ the expected value of $\Delta_{out}(M)$ comes from Eq. 6.2 as follows:

$$E[\Delta_{out}(M)] = \Delta_{in}(0) + E[E[W_M \mid q_1]] - E[W_1(q_1, 0)] \tag{6.8}$$

where,

$$E[W_M \mid q_1] = E[W_2(\Delta_{in}, 0) \mid q_1] = \int_0^{\infty} 1 - (P_{\Delta in}(x, M))dx \tag{6.9}$$

$E[W_M \mid q_1]$ in Eq. 6.8 can be found from distribution model $P_{\Delta in}(x,M)$ as it is shown in Eq. 6.9 and $E[W_1(q_1, 0)]$ can be obtained from the steady state solution of queuing models [88].

According to the result in [29], the relationship between first and the second packet-pair is as follows:

When $\Delta_{in}(0) < 1/\lambda$:

$$E[\Delta_{out}(M) \mid q_1] - \Delta_{in}(0) = \lambda D \Delta_{in}(0) + L_p D / L_c + O(\Delta_{in}(0)) \tag{6.10}$$

When $\Delta_{in}(0) \rightarrow \infty$:

$$E[\Delta_{out}(M) \mid q_1] = \Delta_{in}(0)$$

6.3.2 Path Characterization

As discussed above, one of our contributions is the use of the multihop queuing model to characterize network paths. Now we show how our model can be used to determine link properties such as rate of cross traffic, utilization, and available bandwidth for each link. Initially we assume the path fingerprint is created at each point for a specific initial Δ_{in} . The signature at the end point refers to $F_{\Delta in}(x, M)$. According to Eq. 6.5, λ_M can be obtained as follows since the other variables are known, and we have $F_{\Delta in}(x, i)$ for $i=M$. For $y \leq 0$ we have:

$$F_{\Delta in}(x,i) = \sum_{j=0}^{\lfloor x/D \rfloor} -N0+K(y+D) \frac{(\lambda_i * \Delta_{in}(i))^i}{j!} e^{-(\lambda_i * \Delta_{in}(i))}$$

which simplifies to :

$$F_{\Delta in}(x,i) = e^{-(\lambda_i * \Delta_{in}(i))} * (1 + (\lambda_i * \Delta_{in}(i))^1 + (\lambda_i * \Delta_{in}(i))^2 + ...) \quad (6.11)$$

Since $|(\lambda_i * \Delta_{in}(i))| < 1$, this is approximated by:

$$F_{\Delta in}(x,i) = e^{-(\lambda_i * \Delta_{in}(i))} * \frac{1}{1 - (\lambda_i * \Delta_{in}(i))} \quad (6.12)$$

Hence, the rate of cross traffic λ_M in link M can be obtained from Eq. 6.12. A similar process applies to find λ_M for $y>0$. The utilization and available bandwidth of link M can then be determined from $A = C_i (1-u_i)$ since the capacities of the links are assumed to be known. This procedure can be extended to find other link properties as well by using the corresponding path fingerprint for each link.

6.4 Model Validation

In this section we validate our stochastic model via OPNET® simulations and explore how the signature is shaped by factors such as the number of the hops, link capacity, initial dispersion and cross traffic arrival rate.

We perform simulation for two-hop, four-hop and eight-hop networks, connected via 100 MB/s links. The cross traffic on each node is set at 20% and 60% of link utilization respectively for low and heavy Poisson cross traffic conditions. Non-persistent cross traffic [93] as shown in Figure. 6.4 is used in which packets exit one hop after entering the path. Each packet-pair source generates packets with 120μs initial dispersion, which correspond to the back-to-back transmission of the two 1500 byte packets. We normalized the dispersion at the receiver by the initial dispersion and report Δ_{out}/Δ_{in} as the normalized dispersion.

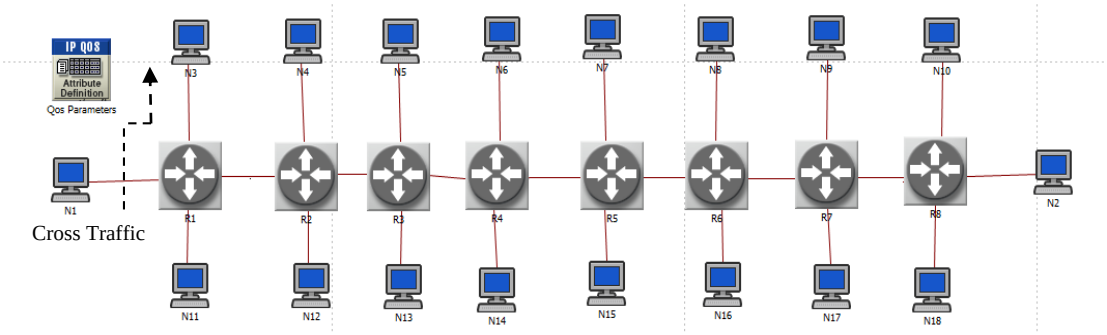


Figure. 6.4. Multi-hop model in OPNET® simulation

Figure. 6.5 and 6.6 shows path signatures based on CDF of packet-pair dispersion obtained with OPNET® simulation and our model for low and heavy cross traffic. The simulation results agree very closely with the stochastic model in Eq. 6.7. In addition, we investigated the effect of number of hops in estimating the path signatures. When the number of hops is increased from 2 to 8 the stochastic model continues to perform quite accurately for heavy cross traffic utilizations. The maximum error of CDF between OPNET® simulation and stochastic model in heavy cross

traffic condition is 0.051 when path signature is generated for eight-hop paths indicating close agreement between stochastic model and simulation results. The other evaluation part for each signature is investigating tight link available bandwidth estimation based on our analysis. The available bandwidth estimation is computed from Eq. 6.12 and shown in Tables. 6.1 and 6.2 for the first scenario. The estimation error shows that our model is able to determine the available bandwidth for the tight link fairly closely; however, the result indicates that the error increases as the number of the hops grow. Moreover, it is clear that having heavy cross traffic affects the accuracy of signatures as well as available bandwidth estimation. Thus the error of estimation for heavy cross traffic is more than at low cross traffic.

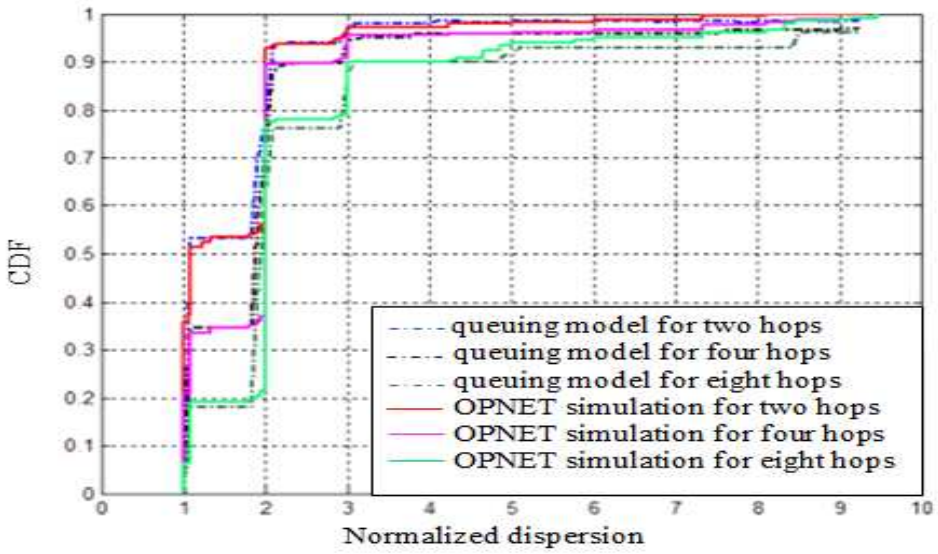


Figure. 6.5. Queuing model and OPNET® simulation results for low cross traffic and back-to-back packet-pair

Table. 6.1. Available bandwidths for low cross traffic

Available Bandwidth	**Actual value (MB/s)**	**Estimated value (MB/s)**	**Estimation error (MB/s)**
Two hop model	80	79.7	0.3
Four hop model	79.8	79	0.8
Eight hop model	80.5	79.1	1.4

Next part of the experiment evaluates our model by injecting packet-pairs with 240µs initial dispersion. In this case we investigate the effect of initial dispersion on path signature and the accuracy of analytical model. As in the previous experiments, we can see in Figure. 6.7 and 6.8 that the model follows OPNET® simulations well in both low and heavy cross traffic conditions. The maximum errors in these results are 0.036 and 0.058 for low and high cross traffic respectively, when dispersion CDF generated for eight-hop model. Estimates for the available bandwidth for this scenario are shown in Tables 6.3 and 6.4.

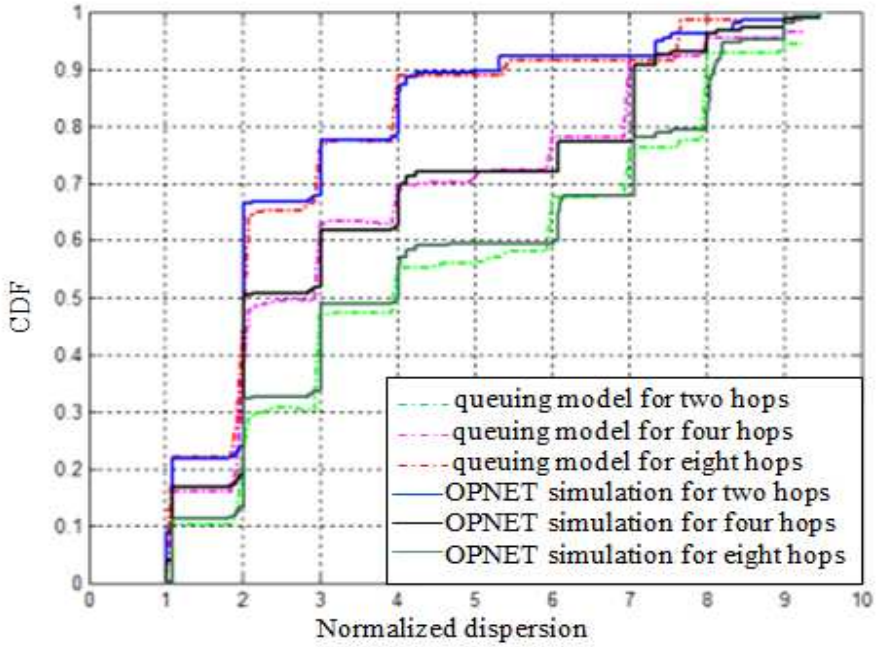


Figure. 6.6. Queuing model and OPNET® simulation results in heavy cross traffic

Table 6.2. Available bandwidth for heavy cross

Available Bandwidth	**Actual value (MB/s)**	**Estimated value (MB/s)**	**Estimation error (MB/s)**
Two hop model	40.1	39.5	0.6
Four hop model	40.3	39	1.3
Eight hop model	39.9	38.1	1.8

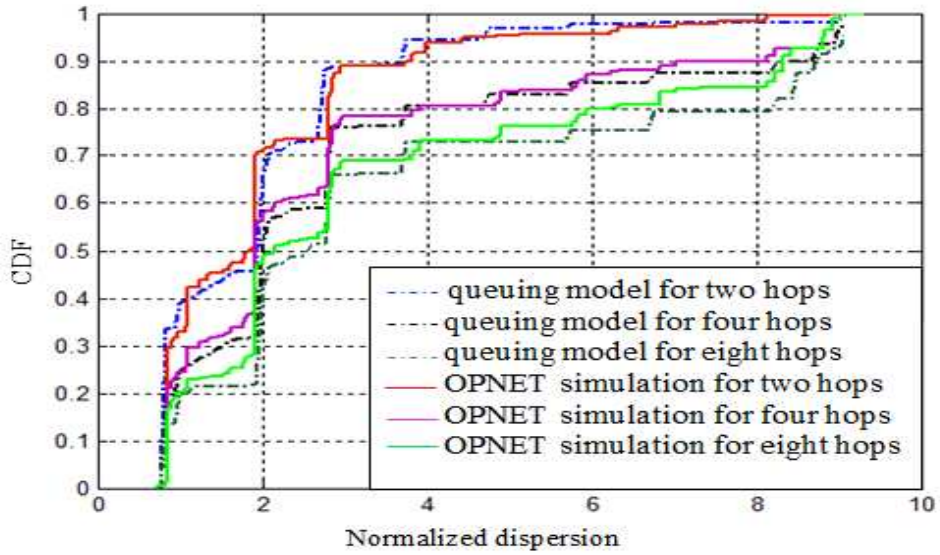


Figure. 6.7. Queuing model and OPNET® simulation results for low cross traffic and 240µs initial dispersion

Table. 6.3. Available bandwidth for low cross traffic

Available Bandwidth	**Actual value (MB/s)**	**Estimated value (MB/s)**	**Estimation error (MB/s)**
Two hop model	80	79.8	0.2
Four hop model	79.8	79.2	0.6
Eight hop model	80.5	79.4	1.1

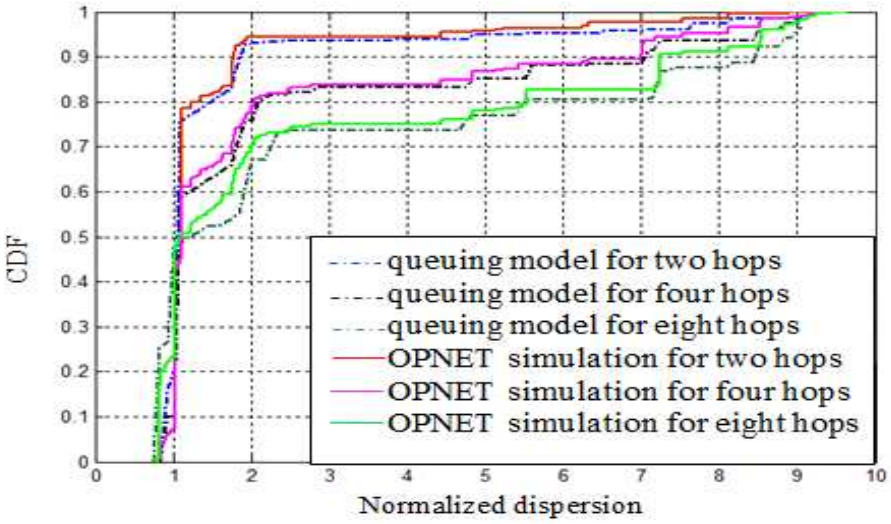


Figure. 6.8. Queuing model and OPNET® simulation results for heavy cross traffic and 240 µs initial dispersion

Table. 6.4. Available bandwidth for heavy cross

Available Bandwidth	**Actual value (MB/s)**	**Estimated value (MB/s)**	**Estimation error (MB/s)**
Two hop model	40.1	39.6	0.5
Four hop model	40.3	39.1	1.2
Eight hop model	39.9	38.3	1.6

Next we evaluate the impact of link capacity values on the shape of path signature. Figure. 6.9 shows the path signatures based on different link capacities and low cross traffic. In this case we perform simulations for paths with 2 and 4 hops. The capacities of links are 100, 80, 60 and 40 MB/s in a sequence. For two-hop case links capacities are set to 100 and 80 MB/s respectively. The path signatures still have reasonable accuracy, with maximum errors of 0.040 and 0.049 for two and four hop cases respectively. The results presented above as well as additional simulations we performed show that the maximum errors are reasonable. In addition, the available bandwidth estimates are shown in Table 6.5. The estimation error is reasonable in this condition as well. Based on the results from different scenarios to estimate the available bandwidth, we can conclude that the heavy cross traffic can affect further in comparison with other parameters such as initial dispersion, different link capacity through the path.

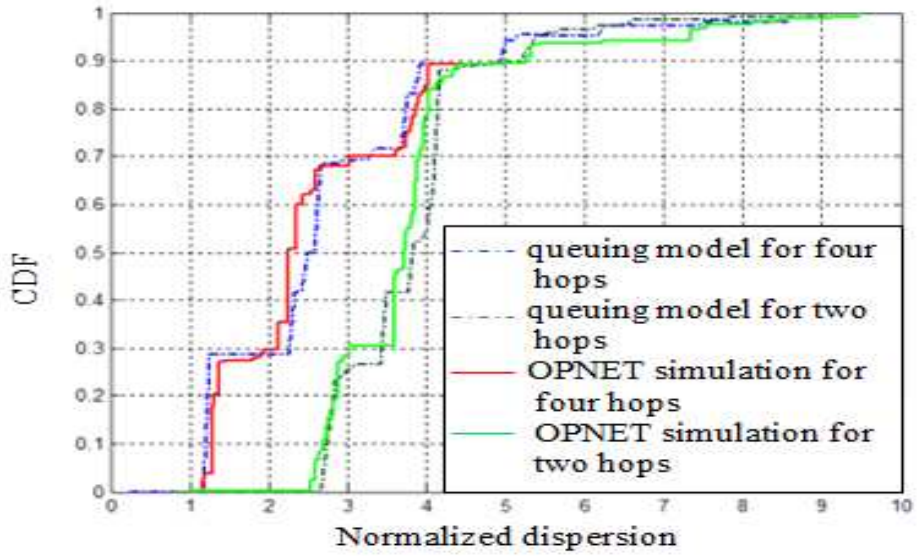


Figure. 6.9 Queuing model and OPNET® simulation results for different link's capacity in low cross traffic

Table. 6.5. Available bandwidth for low cross traffic and different link's capacity

Available Bandwidth	**Actual value (MB/s)**	**Estimated value (MB/s)**	**Estimation error (MB/s)**
Two hop model	32	31.7	0.3
Four hop model	32.3	31.6	0.7

The dispersion CDFs are distinct and persistent for a given path and shape according to link characteristics such as cross traffic, initial dispersion and link capacity, and agrees with the observations in [102]. Therefore, finding an accurate model to generate path signatures for a multi hop model is important to facilitate estimation of other useful properties of the path, such as utilization, bandwidth and cross traffic rate as described in the previous section. We evaluate path characteristics, as described in the previous section, for different link capacities of the path. A path with 4 hops and the links of capacities as 100, 80, 60 and 40 MB/s in sequence. As described, in the first step, the arrival rate of cross traffic is determined from Eq. 6.11, and then the utilization and available bandwidth are computed for each link. The results in Table 6.6 and 6.7 show the estimated utilization and available bandwidth. The worst estimation error for

utilization is not more than 2% and available bandwidth is not more than 1.9 MB/s, which occurs on the last link.

Table. 6.6. Available bandwidth for different links in low crosses traffic

Available Bandwidth	Actual value (MB/s)	Estimated value (MB/s)	Estimation error (MB/s)
Link 1	31.52	31.72	.20
Link 2	47.94	47.22	.28
Link 3	64.08	63.52	.46
Link 4	78.5	80.4	1.9

Table. 6.7. Utilization for different links in low cross traffic

Utilization	Actual value %	Estimated value %	Estimation error %
Link 1	21.2%	20.7%	0.9%
Link 2	20.1%	21.3%	1.2%
Link 3	19.9%	20.6%	1.7%
Link 4	21.5%	19.6%	1.9%

From the overall results we can see that the proposed model can assist to determine the path and link properties in the network; however, we have made the assumption that packets arrive according to a Passion process with rate λ and provide for M/D/1 based queuing model. In the last part of our evaluation, we run experiments with another type of distribution for cross traffic on OPNET® and compare it with the path signature based on our queuing model. This result checks the sensitivity and accuracy of model when cross traffic does not follow a Poisson process. In this case we repeat the first scenario for two and four hops with Pareto arrival process. In general, the Pareto distribution is applied to model self-similar arrival in packet traffic for simulation tools. The results are shown in Figure. 6.10 and 6.11 for low and heavy utilizations. The maximum error between OPNET® simulation with Pareto distribution and stochastic model in low cross traffic condition is 0.09 when path signature generates for four-hop model, and for heavy cross traffic rate maximum error is 0.19. Practically, the shape of signature from our

model follows the OPNET® simulation in this case as well; however, the error increases from 0.03 to 0.09 for low cross traffic rate and from 0.051 to 0.19 for heavy cross traffic. The estimation error in Tables 6.8 and 6.9 shows that our model has 2.9MB/s estimation error in the worst case for heavy cross traffic and four hop model. As the results demonstrate, it is clear that having the other type of arrival cross traffic model cause the reduction in accuracy to provide path properties; however, the estimation error is fairly acceptable in our experiment. Moreover, it is predictable that having more number of hops could affect the accuracy in this condition as well.

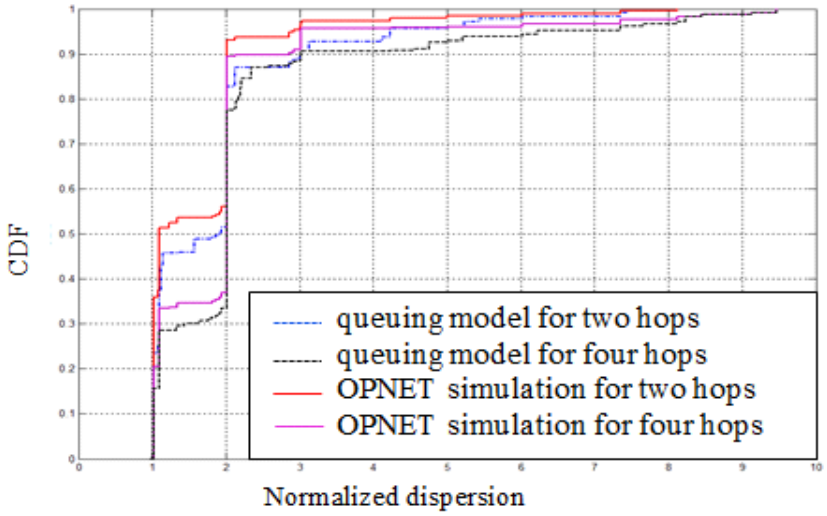


Figure. 6.10. Queuing model and OPNET® simulation results for Pareto distribution and low cross traffic

Table. 6.8. Available bandwidth for low cross traffic and Pareto distribution

Available Bandwidth	**Actual value (MB/s)**	**Estimated value (MB/s)**	**Estimation error (MB/s)**
Two hop model	80.6	79.7	0.9
Four hop model	80.6	79	1.6

In summary, the proposed model can provide the path and link properties in the network based on Poisson model for cross traffic accurately and indicate that the our model can determine appropriate information of the path based on path signature.

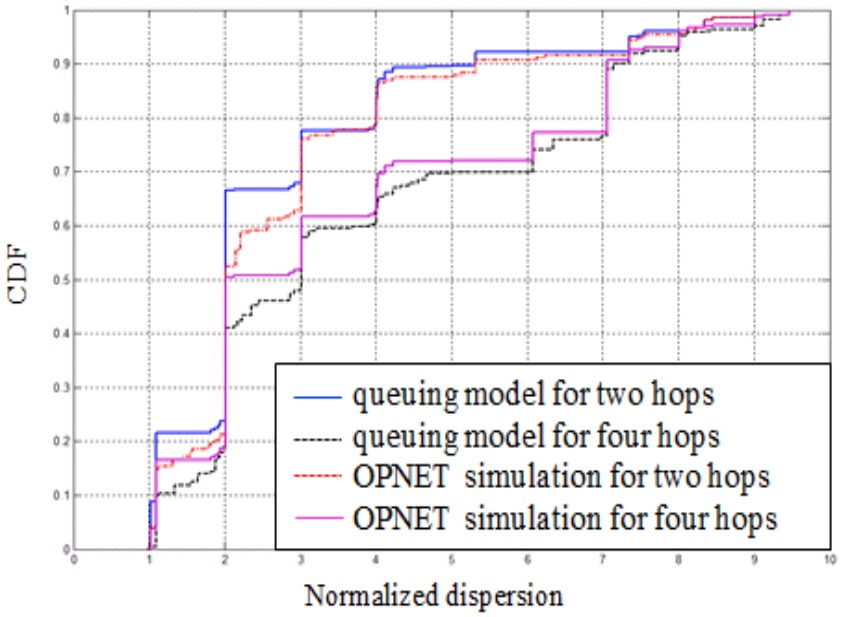


Figure. 6.11. Queuing model and OPNET® simulation results for Pareto distribution and heavy traffic

Table. 6.9. Available bandwidth for heavy cross traffic and Pareto distribution

Available Bandwidth	Actual value (MB/s)	Estimated value (MB/s)	Estimation error (MB/s)
Two hop model	41.3	39.5	1.8
Four hop model	41.9	39	2.9

6.5 Conclusion

In this chapter, we investigated the mathematical relationship between the input and the output packet-pair dispersion to generate accurate path signatures for multi-hop paths without a limitation of single tight links. A mathematical relationship was presented to generate accurate signatures for multihop paths. The analytical model was verified using simulations, and was used to evaluate the impact of factors such as the number of hops, initial dispersion, link capacities and cross traffic, that affect the shape of the signature. Results from the proposed model agree closely with OPNET® simulations. As the path signatures can provide other properties of a path, such as estimate available bandwidth, utilization, bottleneck capacity of the path, monitor and

diagnose network problems, the model can be used to determine useful path properties from the path signatures. Development of the multi hop model for general conditions and a G/G/1 based queuing model remains as future work.

CHAPTER 7

CONCLUSIONS

This dissertation is focused on characterizing and mitigating DDoS attacks by distributed defense mechanisms. We began with efficient attack traffic identification mechanism. An identification scheme is proposed, based on analysis of multiple features of DDoS attacks and normal traffic, to distinguish between attack and normal traffic. Based on the identification model we propose a history-based profiling mechanism to discriminate against malicious traffic with minimal effect on the legitimate traffic. The approach is verified and evaluated using the DARPA dataset as well as by extensive analysis with data sets from Colorado State University and University of Auckland. Then we focused our attention on a filtering mechanism. Look up the IP address in the upstream routers during the attack time is a costly task and creates extreme overhead on routers. However, dropping attack traffic requires such inspection. We approach this dilemma by identifying only those routers that carry a significant amount of attack traffic. Identification is carried out using the proposed history based profile. A novel Compacted Bloom filter is presented and a filter dissemination mechanism is proposed based this mechanism. The identification of the particular routers is done without interfering with the normal operation. Our approach helps reduce the communication and computation costs and storage requirements of the upstream routers required to check for malicious traffic. The compacted Bloom filter is motivated by applications, including DDoS mitigation, that must transmit Bloom filters over the network and/or when endpoint machines have restrictions on memory available for this purpose. The scheme reduces the memory requirement, but at the cost of false positives in response to the queries. A corporative defense mechanism is presented to mitigate DDoS traffic. Our scheme

identifies and places such filters to block attack traffic and allow legitimate traffic as close to the source node as possible, so that network resources are not wasted in propagating the attack. The solution benefits by using a recently developed packet probe technology, such as that from JDSU, typically implemented as Small Firmfactor Probes (SFP) using FPGAs. We evaluate the effectiveness of the proposed mechanisms using extensive simulation in OPNET® with a real network topology. Finally, an analytical model that provides deeper insights to packet-pair path signature in the presence of multiple tight links is presented, thus enabling efficient and accurate network monitoring, problem diagnosis and estimation of link and path parameters such as end-to-end network bandwidths and link capacities.

Learning about traffic features during the attack and normal conditions, and identifying clues which can separate attack flows from normal flows efficiently are some of the main challenges in attack mitigation. Deep Packet Inspection (DPI) [106] is a maturing technology capable of recognizing data relationships and communication patterns. Future work is needed to exploit emerging capabilities of DPI to improve the detection of attack traffic provide new solutions. DPI solution can also assist in detecting malware and normal patterns, and this can be added as a new feature in the identification model to discriminate between malicious or attack traffic and the normal traffic. Moreover, to reduce the monitoring and computation costs at the victim node when creating history-based profiles, packet sampling based methods can be investigated. Packet sampling involves selection of a subset of packets in the stream, such as a random selection or a systematic selection, e.g., count-based with the periodic selection of every k^{th} packet or time-based driven a packet is selected every constant time interval. Further analysis and evaluation is required in order to better understand how packet sampling can meet the accuracy and timing

requirements, and its effect on false alarms. This future work also may address the impact of packet sampling on different features.

The evaluations presented were based on IPv4 traffic traces. In contrast to IPv4, IPv6 has a vastly enlarged address space (128 bits compared with 32 bits in IPv4) that makes it more difficult to address the specific characteristics related to identifying and filtering the attack traffic. As IPv6 traces become available, further evaluation of the proposed scheme may be carried out. IPv6 may present unique opportunities to sub-sample IP header field, e.g., without having to look at the entire address fields. The issue of having the same capabilities to defend against both IPv4 and IPv6 attacks is critical and according to some literatures [107] [108] the current IPv6 DDoS attacks are most likely due to tests performed by malware writers who want to be prepared for the case when large internet service providers switch their subscribers to IPv6.

BIBLIOGRAPHY

[1] C. Gong and K. Sarac, "IP traceback based on packet marking and logging," in *Proc. of IEEE International Conference on Communications*, Seoul, Korea, May 2005.

[2] T. Peng, C. Leckie and K. Ramamohanarao, "Proactively detecting distributed denial of dervice attacks using source IP address monitoring," in *Proc. of 3th International IFIP-TC6 Networking Conference*, Athens, Greece, 2004.

[3] J. B. D. Cabrera, L. Lewis, . X. Z. Qin and W. Lee, "Proactive intrusion detection and distributed denial of service attacks-A case study in security management," *Journal of Network and Systems Management*, vol. 10, no. 2, pp. 225-254, 2002.

[4] CNN.Cyber-attacks batter web heavyweights, "http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html," February 2000. [Online].

[5] T. M. Gil and M. Poletto, "MULTOPS: a data-structure for bandwidth attack detection," in *Proc. of the 10th conference on USENIX Security Symposium*, Washington, D.C, USA, 2001.

[6] "Waikato Applied Network Dynamics Research Group. Auckland university data traces," [Online]. Available: http://wand.cs.waikato.ac.nz/wand/wits/.

[7] [Online]. Available: http://www.darkreading.com/attacks-and-breaches/ddos-attack-hits-400-gbit-s-breaks-record/d/d-id/1113787.

[8] T. Peng, C. Leckie and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Computing Surveys*, vol. 39, no. 1, pp. 1-42, 2007.

[9] J. Cheng, J. Yin, Y. Liu, Z. Cai and C. Wu, "DDoS attack detection using IP address feature interaction," in *Proc. 1st Int. Conference Intelligent Networking and Collaborative Systems*, Barcelona, Spain, 2009.

[10] C. Cheng, H. T. Kung and K. Tan, "Use of spectral analysis in defense against DoS attacks," in *Proc. of IEEE GLOBECOM 2002*, Taipei, Taiwan, 2002.

[11] A. Lakhina, M. Crovella and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proc. of the 2004 conference on Applications, technologies, architectures, and protocols*

for computer communications, Portland, Oregon, USA, 2004.

[12] C. Manikopoulos and S. Papavassiliou, "Network intrusion and fault detection: a statistical anomaly approach," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 76-82, 2002.

[13] S. Noh, G. Jung, K. Choi and C. Lee, "Compiling network traffic into rules using soft computing methods for the detection of flooding attacks," *Journal of Applied Soft Computing*, vol. 8, no. 3, pp. 1200-1210, June, 2008.

[14] S. Abdelsayed, D. Glimsholt, C. Leckie, S. Ryan and S. Shami, "An efficient filter for denial-of-service bandwidth attacks," in *Proc. of the 46th IEEE Global Telecommunications Conference (GLOBECOM'03)*, San Francisco, CA, USA, 2003.

[15] J. Li, J. Mirkovic, W. Wang, P. Reiher and L. Zhang, "SAVE: Source address validity enforcement protocol," in *Proc. of IEEE INFOCOM 2002*, New York, USA, 2002.

[16] J. Mirkovic and P. Reiher, "D-WARD: A source-end defense against flooding denial-of-service attacks," *IEEE Trans. on Dependable and SecureComputing*, vol. 2, no. 3, pp. 216-232, July 2005.

[17] H. Wang, D. Zhang and K. Shin, "Detecting SYN flooding attacks," in *Proc of IEEE INFOCOM 2002*, New York, USA, 2002.

[18] T. Peng, C. Leckie and K. Ramamohanarao, "Detecting distributed denial of service attacks using source IP address monitoring," in *Proc. of the 3rd International IFIP-TC6 Networking Conference Networking 2004*, Athens, Greece, 2004.

[19] "DARPA Intrusion Detection Dataset 1998," [Online]. Available: http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1998data.html.

[20] "Colorado State University dataset," [Online]. Available: https://www.predict.org/Default.aspx?tabid=104.

[21] "The CAIDA "DDoS Attack 2007" Dataset," [Online]. Available: http://www.caida.org/data/passive/ddos-20070804_dataset.xml.

[22] R. Mahajan, S. M. Bellovin, S. Floyd and J. Ioannidi, "Controlling high bandwidth aggregates in the network," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 62-73, 2002.

[23] C. Papadopoulos, R. Lindell, J. Mehringer and A. Huss, "COSSACK: Coordinated Suppression of Simultaneous Attacks," in *Proc. of Discex III*, Washington, DC, USA, April 2003.

[24] Z. Shu and P. Dasgupta, "Denying Denial-of-Service Attacks: A Router Based Solution," in *Proc. of International Conference on Internet Computing*, 2003.

[25] J. Francois, I. Aib and R. Boutaba, "Firecol, a collaborative protection network for the detection of flooding ddos attacks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, p. 1828–1841, 2012.

[26] Z. Aghaei Foroushani, "TDFA: traceback-based defense against DDoS flooding attacks," in *Proc of 28th International Conference on Advanced Information Networking and Applications(AINA) IEEE 2014*, Victoria, BC , 2014.

[27] R. Chen and J.-M. Park, "Attack Diagnosis: Throttling Distributed Denial-of-Service Attacks Close to the Attack Sources," in *IEEE Int'l Conference on Computer Communications and Networks (ICCCN'05)*, Oct. 2005.

[28] H. Beitollahi and G. Deconinck, "Analyzing Well:Known Countermeasures against Distributed Denial of Service Attacks," *Computer Communications*, vol. 35, no. 11, pp. 1312-1332, June 2012.

[29] K.-J. Park, H. Lim and C.-H. Choi, "Stochastic analysis of packet-pair probing for network bandwidth estimation," *Computer Networks:The International Journal of Computer and Telecommunications Networking*, vol. 50, no. 12, pp. 1901-1915, Aug. 2006.

[30] [Online]. Available: http://www.jdsu.com/en-us/Test-and-Measurement/Products/a-z-product-list/Pages/packetportal.aspx.

[31] B. Kumar Dey, D. Manjunath and S. Chakraborty, "Estimating network link characteristics using packet-pair dispersion: A discrete-time queueing theoretic analysis," *The International Journal of Computer and Telecommunications Networking*, vol. 55, no. 5, pp. 1052-106, 2011.

[32] Y. Chen, K. Hwang and W.-S. Ku , "Collaborative detection of DDoS attacks over multiple network domains," *IEEE Trans. on parallel and distributed*, vol. 18, no. 12, pp. 1649 -1662, 2007.

[33] Y. Chen, A. Kumar and J. Xu, "A new design of Bloom filter for packet inspection

speedup," in *IEEE GLOBECOM*, 2007.

[34] D. Žagar, K. Grgić and R.-D. Snježana , "Security aspects in IPv6 networks – implementation and testing," *Computers & Electrical Engineering,* vol. 33, no. 5-6, p. 425–437, November 2007.

[35] J. Jung, B. Krishnamurthy and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites," in *Proc. of 11th World Wide Web conference*, Honolulu, Hawaii, USA, 2002.

[36] "RioRey, Inc. 2009-2012, RioRey Taxonomy of DDoS Attacks, RioReyTaxonomyRev2.32012, 2012.," [Online]. Available: [online]http://www.riorey.com/xresources/2012/RioRe.

[37] S. Dietrich, N. Long and D. Dittrich, "Analyzing distributed denial of service tools: The Shaft case," in *Proc. of 14th USENIX conference on System administration*, New Orleans, Louisiana, USA, 2000.

[38] T. Peng, C. Leckie and K. Ramamohanarao, "Protection from distributed denial of service attack using history-based IP filtering," in *Proc. of IEEE International Conference on Communications*, Anchorage, Alaska, 2003.

[39] K. Lee, J. Kim, K. H. Kwon, Y. Han and S. Kim, "DDoS attack detection method using cluster analysis," *Expert Systems with Applications,* vol. 34, no. 3, p. 1659–1665, April 2008.

[40] M. Mitzenmacher, "Compressed Bloom filters," in *Proc. of the 20th annual ACM symposium on Principles of distributed computing*, Newport, Rhode Island, 2001.

[41] M. Mitzenmacher, "Compressed Bloom filters," *IEEE/ACM Transactions on Networking,* vol. 10, no. 5, p. 604–612, October 2002 .

[42] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM,* vol. 13, no. 7, pp. 422-426, July 1970.

[43] K. Xie, Y. Min , D. Zhang, J. Wen and G. Xie, "A Scalable Bloom Filter for Membership Queries," in *Proc. of IEEE Conference on Global Telecommunications*, Washington, DC, Nov. 2007 .

[44] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: a survey," *Internet Mathematics,* vol. 1, no. 4, p. 485–509, 2004.

[45] R. P. Laufer, P. B. Velloso, D. d. O. Cunha, I. Moraes, M. Bicudo and O. Duarte, "A new IP traceback system against denial-of-service attacks," in *Proc. of Twelfth international conference on telecommunications*, Capetown, South Africa, 2005.

[46] R. Laufer, P. Velloso, D. d. O. Cunha, I. Moraes, M. Bicudo, M. Moreira and O. Duarte, "Towards stateless single-packet IP traceback," in *Proc. of the 32nd IEEE Conference on Local Computer Networks (LCN 2007)*, Dublin, Ireland, 2007.

[47] L. Fan, P. Cao , J. Almeida and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, p. 281–293, June 2000.

[48] F. Cuenca-Acuna, C. Peery, R. Martin and T. Nguyen, "PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities," in *Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing*, June 2003 .

[49] S. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *Proc. of the IEEE INFOCOM 2002 Conference*, New York, NY, USA, 2002.

[50] T. Hodes, . S. Czerwinski, B. Zhao and A. Josep, "An architecture for secure wide-area service discovery," *Wireless Networks*, vol. 8, no. 2/3, p. 213–230, March-May 2002.

[51] L. H. M. K. Costa, S. Fdida and O. C. M. B. Duarte, "Incremental service deployment using the hop-by-hop multicast routing protocol," *IEEE/ ACM Transactions on Networking*, vol. 14, no. 3, p. 543–556, June 2006 .

[52] A. C. Snoeren, C. Partridge, L. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. Kent and W. Strayer, "Single-packet IP traceback," *IEEE/ ACM Transactions on Networking*, vol. 10, no. 6, p. 721–734, December 2002 .

[53] R. P. Laufer, P. B. Velloso and O. C. M. B. Duart, "A generalized Bloom filter to secure distributed network applications," *Computer Networks*, vol. 55, no. 8, pp. 1804-1819, 2011.

[54] D. Guo, Y. Liu, X. Li and P. Yang, "False negative problem of counting Bloom filter," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, p. 651–664, May 2010 .

[55] F. Bonomi, M. Mitzenmacher, R. Panigr, S. Singh and G. Varghese, "An improved construction for counting bloom filters," in *Proc. of the 14th conference on Annual*

European Symposium, 2006.

[56] G. Antichi, D. Ficara, S. Giordano, G. Procissi and F. Vitucc, "Counting Bloom filters for pattern matching and anti-evasion at the wire speed," *IEEE Network*, vol. 23, no. 1, pp. 30 -35, 2009.

[57] S. Cohen and Y. Matias, "Spectral Bloom filters," in *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data*, San Diego, California, USA, 2003.

[58] H.-S. Hu, H.-W. Zhao and F. Mi, "L-priorities bloom filter: A new member of the bloom filter family," *International Journal of Automation and Computing*, vol. 9, no. 2, pp. 171-176, April 2012.

[59] A. Yaar, A. Perrig and D. Song, "Pi: A Path Identification Mechanism to Defend against DDoS Attacks," in *Proc. of the 2003 IEEE Symposium on Security and Privacy*, Pittsburgh, PA, USA, May 2003.

[60] H. Wang, C. Jin and K. Shin, "Defense against spoofed IP traffic using hop-count filtering," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, p. 40–53, 2007.

[61] N. Ahmad and A. Yaacob, "End to End Ipsec Support across Ipv4/Ipv6 Translation Gateway," in *2010 Second International Conference on Network Applications Protocols and Services (NETAPPS)*, Kedah, Sept. 2010.

[62] J. Mirkovic, G. Prier and P. L. Reiher, "Attacking DDoS at the Source," in *Proc. of the 10th IEEE International Conference on Network Protocols*, Los Angeles, CA, USA, November 2002.

[63] J. Mirkovic and P. Reihe, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communications Review*, vol. 34, no. 2, pp. 39-53, April 2004.

[64] J. Mirkovic, P. Reiher and M. Robinson, "Forming alliance for DDoS Defense," in *Proc. of New Security Paradigms Workshop, Centro Stefano Francini*, Ascona, Switzerland, 2003.

[65] A. John and T. Sivakumar, "DDoS: Survey of Traceback Methods," *International Journal of Recent Trends in Engineering ACEEE (Association of Computer Electronics & Electrical Engineers)*, vol. 1, no. 2, May 2009.

[66] S. Zargar, J. Joshi and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys & Tutorials*,

vol. 15, no. 4, pp. 2046 - 2069, March 2013.

[67] M. Sung and J. Xu, "IP traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 9, pp. 861-872, September 2003.

[68] A. Yaar, A. Perrig and D. Song, "SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks," in *Proc. of IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 2004.

[69] B. Prabadevi and N. Jeyanthi, "Distributed Denial of service attacks and its effects on Cloud environment- a survey," in *The 2014 International Symposium on Networks, Computers and Communications*, Hammamet , 2014.

[70] R. Thomas, B. Mark, T. Johnson and J. Croall, "NetBouncer: Client-legitimacy-based High-performance DDoS Filtering," in *Proc. of DISCEX III*, 2003.

[71] O. Osanaiye, "Short Paper: IP spoofing detection for preventing DDoS attack in Cloud Computing," in *Proc. of 2015 18th International Conference on Intelligence in Next Generation Networks (ICIN)*, Paris, Feb. 2015.

[72] A. Keromytis, V. Misra and D. Rubenstein, "SOS: an architecture for mitigating DDoS attacks," *Journal on Selected Areas in Communications*, vol. 22, no. 1, p. 176–188, 2004.

[73] F. Lee and S. Shieh, "Defending against spoofed DDoS attacks with path fingerprint," *Elsevier Journal of Computers & Security*, vol. 24, no. 7, p. 571–586, 2005.

[74] M. Jog, M. Natu and S. Shelke, "Distributed Capabilities-based DDoS Defense," in *Proc. of 2015 International Conference on Pervasive Computing (ICPC)*, Pune, Jan. 2015 .

[75] D. Yau, J. Lui, F. Liang and Y. Yam, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Transactions on Networking*, vol. 13, no. 1, p. 29–42, 2005.

[76] A. AlSa'deh and C. Meinel, "SEcure Neighbor Discovery: Review, Challenges, Perspectives and Recommendations," in *Security & Privacy*, 2012.

[77] A. Sadeghian and M. Zamani, "Detecting and Preventing DDoS Attacks in Botnets by the Help of Self Triggered Black Holes," in *Proc. of 2014 Asia-Pacific Conference on Computer Aided System Engineering (APCASE)*, 2014.

[78] H. Wang, D. Zhang and K. G. Shin, "Change-Point Monitoring for the Detection of DoS Attacks," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, no. 4, pp. 193-208, 2004.

[79] A. Sardana, R. Joshi and T. Kim, "Deciding Optimal Entropic Thresholds to Calibrate the Detection Mechanism for Variable Rate DDoS Attacks in ISP Domain," in *Proc. of the 2008 International Conference on Information Security and Assurance (ISA 2008)*, Washington, DC, USA, 2008.

[80] B. Gupta, M. Misra and R. Josh, "FVBA: A combined statistical approach for low rate degrading and high bandwidth disruptive DDoS attack detection in ISP domain," in *Proc. 16th IEEE International Conference on Networks (ICON)*, New Delhi, India, Dec. 2008 .

[81] J. Françcois, A. E. Atawy, E. A. Shaer and R. Boutaba, "A collaborative approach for proactive detection of distributed denial of service attacks," in *Proc. of IEEE MonAM*, Toulouse, France, 2007.

[82] N. Mosharraf, A. Jayasumana and I. Ray, "A Responsive Defense Mechanism against DDoS Attacks," in *Proc.of 7th International Symposium on Foundations and Practice of Security (FPS 2014)*, Montreal, Canada, November 2014.

[83] [Online]. Available: http://www.jdsu.com/ProductLiterature/PacketPortal-difference-between-SFProbe-and-SFP.pdf.

[84] G. Franx, "The transient M/D/c queueing system," [Online]. Available: URL: http://www.cs.vu.nl/~franx/. (2002).

[85] V. Paxson, "End-to-end internet packet dynamics," *IEEE/ACM Transactions on Networking (TON),* vol. 7, no. 3, pp. 277-292, June 1999.

[86] X. Liu, K. Ravindran, B. Liu and D. Loguinov, "Single-hop probing asymptotics in available bandwidth estimation: sample-path analysis," in *Proc. of the 4th ACM SIGCOMM conference on Internet measurement*, October 2004.

[87] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil and L. Cottrell, "Pathchirp: efficient available bandwidth estimation for network paths," in *Proc. of Passive and Active Measurement Workshop (PAM)*, 2003.

[88] H. Tijm, Stochastic Models—an algorithmic approach, John Wiley & Sons Inc, Second ed., 2000.

[89] L. Mingfu, W. Yueh-Lin and C. Chia-Rong, "Available bandwidth estimation for the network paths with multiple tight links and bursty traffic," *Journal of Network and Computer Applications,* vol. 36, no. 1, pp. 353-367, January 2013.

[90] A. Downey, "Using pathchar to estimate Internet link characteristics," in *Proc of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, May 1999.

[91] R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Performance Evaluation,* Vols. 27-28, pp. 297-318 , Oct. 1996.

[92] Z. Hu, D. Zhang, A. Zhu, Z. Chen and Z. Zhou, "SLDRT: A measurement technique for available bandwidth on multi-hop path with bursty cross traffic," *Computer Networks*, vol. 56, no. 14, pp. 3247-60, September. 2012.

[93] R. Kapoor, L. Chen, L. Lao, M. Gerla and M. Sanadidi, "CapProbe: a simple and accurate capacity estimation technique," in *Proc. of Applications, Technologies Architectures and Protocols for Computer Communications*, Aug. 2004.

[94] R. Lubben, M. Fidler and J. Liebeherr, "Stochastic bandwidth estimation in networks with random service," *IEEE/ACM Transactions on Networking,* vol. 22, no. 2, pp. 484-497, 2014.

[95] J.-C. Bolot, "End-to-end packet delay and loss behavior in the internet," in *Proc. of Communications Architectures, Protocols and Applications*, September 1993.

[96] C. Dovrolis, P. Ramanathan and D. Moore, "What do packet dispersion techniques measure?," in *Proc. of INFOCOM 2001*, 2001.

[97] N. Hu and P. Steenkiste, "Evaluation and characterization of available bandwidth probing techniques," *IEEE Journal of Selected Areas in Communication,* vol. 21, no. 6, p. 879–894, Aug. 2003.

[98] V. Jacobson, "Congestion avoidance and control," in *Proc. of Communications architectures and protocols Symposium*, August 1988.

[99] S. Keshav, "A control-theoretic approach to flow control," in *Proc. of Communications Architecture & Protocols*, Zurich, Switzerland, September 1991.

[100] B. Melander, M. Bjorkman and P. Gunningberg, "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM) 2000*, San Francisco, CA, 2000.

[101] J. Strauss, D. Katabi and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proc. of ACM Internet Measurement Conference (IMC)*, 2003.

[102] R. Sinha, C. Papadopoulos and J. Heidemann, "Fingerprinting Internet paths using packet pair dispersion," Technical Report 06-876, USC, Computer Science Dept, 2005.

[103] R. Prasad, C. Dovrolis, M. Murray and K. Claffy, "Bandwidth estimation: metrics, measurement techniques, and tools," *IEEE Network: The Magazine of Global Internetworking*, vol. 17, no. 6, pp. 27 - 35 , November 2003.

[104] K. Lai and M. Baker, "Measuring bandwidth," in *Proc. of IEEE INFOCOM*, 1999.

[105] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 4, pp. 537-549, August 2003.

[106] S. Dharmapurikar, P. Krishnamurthy, T. Sproull and J. Lockwood, "Deep Packet Inspection Using Parallel Bloom Filters," in *Proc. 11th Symp. High Performance Interconnects (HOTI'03)*, Stanford, California,, 2003.

[107] J. Arkko, T. Aura, J. Kempf , V.-M. Mäntylä, P. Nikander and M. Roe, "Securing IPv6 neighbor and router discovery," in *Proc. of the 1st ACM workshop on Wireless security*, Atlanta, GA, USA, September 2002.

[108] S. Szigeti and P. Risztics, "Will IPv6 Bring Better Security?," in *Proc. of 30th Euromicro Conference*, 2004.

[109] J. Arkko and P. Nikander, "Limitations of IPsec policy mechanisms," in *Proc. of the 11th international conference on Security Protocols*, 2005.

[110] M. Huang, J. Liu and Y. Zhou, "An Improved SEND Protocol against DoS Attacks in Mobile IPv6 Environment," in *Proc. of 2009 IEEE International Conference on Network Infrastructure and Digital Content*, 2009.

[111] M. Barati, A. Abdullah, N. Udzir, R. Mahmod and N. Mustapha, " Distributed Denial of Service detection using hybrid machine learning technique," in *Proc. of the 2014 International Symposium on Biometrics and Security Technologies (ISBAST'14)*, Kuala Lumpur, Malaysia, August 2015.

APPENDIX A

SOURCE CODES

A.1 The code below creates Bloom filter from corresponding list of IP address

```
% Read list of IP address in History

fin=fopen('C:\Data\data\history\3part\tcp_udp_icmp_history1326_3partip.txt');

C=textscan(fin,'%s\n');

size_C = size(C{1})

for i=1:1:size_C

% call 4 different HashFunction() for each IP address
%
% input,C{:}(i) is IP address

D=HashFunction(char(C{1}(i)));

set_bit= mod(D,28640)+1;

A(set_bit,1)=1;

D=HashFunction2(char(C{1}(i)));

set_bit= mod(D,28640)+1;

A(set_bit,1)=1;

D=HashFunction3(char(C{1}(i)));

set_bit= mod(D,28640)+1;

A(set_bit,1)=1;

D=HashFunction4(char(C{1}(i)));

set_bit= mod(D,28640)+1;

A(set_bit,1)=1;
```

```
D=HashFunction5(char(C{1}(i)));

set_bit= mod(D,28640)+1;

A(set_bit,1)=1;

end

filename = 'U:\Documents\Bloomfilter_1326_3partip.txt';

fid = fopen(filename, 'w');

for j=1:28640

   fprintf(fid, '%d\n', A(j,1));

end

fclose(fin);

function hash=string2hash(str,type)

% This function generates a hash value from a text string
%
% hash=string2hash(str,type);
%
% inputs,
%   str : The text string, or array with text strings.
% outputs,
%   hash : The hash value, integer value between 0 and 2^32-1
%   type : Type of has 'djb2' (default) or 'sdbm'
%
% example,
%
%  hash=string2hash('hello world');
%  disp(hash);
%
% From string to double array

str=double(str);

if(nargin<2), type='djb2'; end

switch(type)

    case 'djb2'
```

```
        hash = 5381*ones(size(str,1),1);
        for i=1:size(str,2),
            hash = mod(hash * 90 + str(:,i), 2^64-1);
        end
    case 'sdbm'
        hash = zeros(size(str,1),1);
        for i=1:size(str,2),
            hash = mod(hash * 65599 + str(:,i), 2^32-1);
        end
    otherwise
        error('string_hash:inputs','unknown type');
end
```

A.2 The code below check if corresponding IP address is a member of created Bloom filter or not

```
% input, B and T: created Bloom filter and incoming traffic
%
% output, Match_ip: list of IP address match with Bloom filter

Match_ip= cell(1,1);

n=1;

% Read created Bloom filter

fin=fopen('U:\Documents\Bloomfilter_1225_3partip.txt');

B=textscan(fin,'%d\n');

size_B = size(B{1});

% Read incoming traffic that contains list of IP address
```

```
fin1=fopen('C:\Data\data\history\3part\history_2firstweek_3partip.txt');

T=textscan(fin1,'%s\n');

size_T = size(T{1})

% check if corresponding IP address is a member of created Bloom

for i=1:1:size_T

D1=HashFunction(char(T{1}(i)));

set_bit1= mod(D1,28495)+1;

D2=HashFunction2(char(T{1}(i)));

set_bit2= mod(D2,28495)+1;

D3=HashFunction3(char(T{1}(i)));

set_bit3= mod(D3,28495)+1;

D4=HashFunction4(char(T{1}(i)));

set_bit4= mod(D4,28495)+1;

D5=HashFunction5(char(T{1}(i)));

set_bit5= mod(D5,28495)+1;

   if(B{1}(set_bit1)==1 && B{1}(set_bit2)==1 && B{1}(set_bit3)==1 &&
B{1}(set_bit4)==1 && B{1}(set_bit5)==1)

          Match_ip{1}(n)= T{1}(i);

          n=n+1;
   end

end

fclose(fin);

filename = 'C:\Data\data\attack_match\3part\326\Match_ip_326_3partip.txt';

fid2 = fopen(filename, 'w');
```

```
for j=1:n-1
   fprintf(fid2,'%s\n', char(Match_ip{1}(j)));
end
fclose(fin);
fclose(fin1);
function hash=string2hash(str,type)
% This function generates a hash value from a text string
%
% hash=string2hash(str,type);
%
% inputs,
%   str : The text string, or array with text strings.
% outputs,
%   hash : The hash value, integer value between 0 and 2^32-1
%   type : Type of has 'djb2' (default) or 'sdbm'
%
str=double(str);
if(nargin<2), type='djb2'; end
switch(type)
   case 'djb2'
      hash = 5381*ones(size(str,1),1);
      for i=1:size(str,2),
         hash = mod(hash * 33 + str(:,i), 2^32-1);
      end
   case 'sdbm'
      hash = zeros(size(str,1),1);
      for i=1:size(str,2),
```

```
            hash = mod(hash * 65599 + str(:,i), 2^64-1);
        end
    otherwise
        error('string_hash:inputs','unknown type');
end
```

A.3 The code below check TCP connections and extract those IPs establish three way handshake connection

```
% file names
incFName='filterdata20010318-020000-0.txt';
ougFName='filterdata20010318-020000-1.txt';
m=1;
% settings
timeout=60; % 1min

% load data
fprintf('Loading data... ');
tic
incF=fopen(incFName);
ougF=fopen(ougFName);
incData=textscan(incF,'%f %s %s %d %d');
ougData=textscan(ougF,'%f %s %s %d %d');
fclose(incF);
fclose(ougF);
toc
```

```
fprintf('Building history... ');
tic
History={}; % {remote-IP, time-stamp}
filename = 'History20010318_020000_0.txt';
fid = fopen(filename, 'w');
histInd=1;
% incoming trace first timestamp
incT1=incData{1}(1);
% outgoing trace first timestamp
ougT1=ougData{1}(1);
% start time % time block indices
stT=min(incT1,ougT1);
% time block indices for incoming
incTBlocks=floor((incData{1}(:)-stT)/60)+1;
% time block indices for incoming
ougTBlocks=floor((ougData{1}(:)-stT)/60)+1;
% last time block index
lastTBlock=max([incTBlocks;ougTBlocks]);
count=0;
for tblock=1:lastTBlock
% for tblock=1
    incBlockInds=find(incTBlocks==tblock | incTBlocks==tblock+1);
    ougBlockInds=find(ougTBlocks==tblock);
```

```
% timestamp info
incTSs=incData{1}(incBlockInds);
ougTSs=ougData{1}(ougBlockInds);
% ip info
incIPs=removePort_v01(incData{2}(incBlockInds));
ougIPs=removePort_v01(ougData{3}(ougBlockInds));
% sequence# info
incSEQs=incData{4}(incBlockInds);
ougSEQs=ougData{4}(ougBlockInds);
% ack# info
incACKs=incData{5}(incBlockInds);
ougACKs=ougData{5}(ougBlockInds);
for pktInd=1:length(ougBlockInds)
count=count+1;
 if count>300
   return
 end
   % scan for each outgoing packet
   candInd=find(strcmp(incIPs,ougIPs(pktInd)) & ...
       incTSs>ougTSs(pktInd) & incSEQs==ougACKs(pktInd) & ...
       incACKs==ougSEQs(pktInd)+1,1,'first');
   if ~isempty(candInd)
       History{histInd}={ougIPs(pktInd),ougTSs(pktInd)};
```

```
        histInd=histInd+1;
      end

   end

   [nrows,ncols]= size(History);

   for i=m:1:ncols

       fprintf(fid, '%s %f\n',  char(History{i}{1}),  History{i}{2});

       m=ncols+1;
   end

end

 fclose(fid);

toc
```

A.4 The code below checks DARPA dataset and create signature of extracted feature for TCP, UDP and ICMP packets

```
% TCP traffic

fin=fopen('N:\DARPA1998\TrainingDataset\week5\monday\TCP_Week5_Monday.txt');

C=textscan(fin,'%f %s %f %s %f %f %f %f %f');

size_C = size(C{1});

%get occurrence of port and size

PortS=unique(C{3});

P1= hist(C{3}, PortS);

PortS(1:end,2)= P1(1:end);

[m1, n1]=size(PortS);

filename =
'N:\DARPA1998\TrainingDataset\week5\monday\TCP_PortS_Week5_Monday.txt';
```

```
fid1 = fopen(filename, 'w');
for j=1:1:m1
    fprintf(fid1,'%d ', PortS(j,1:2));
    fprintf(fid1,'\n');
end
fclose(fid1);
%port destination
PortD=unique(C{5});
P2= hist(C{5}, PortD);
PortD(1:end,2)= P2(1:end);
[m1, n1]=size(PortD);
filename =
'N:\DARPA1998\TrainingDataset\week5\monday\TCP_PortD_Week5_Monday.txt';
fid1 = fopen(filename, 'w');
for j=1:1:m1
    fprintf(fid1,'%d ', PortD(j,1:2));
    fprintf(fid1,'\n');
end
fclose(fid1);
%size of packet
PacketS=unique(C{9});
P3= hist(C{9}, PacketS);
PacketS(1:end,2)= P3(1:end);
```

```
[m1, n1]=size(PacketS);

filename =
'N:\DARPA1998\TrainingDataset\week5\monday\TCP_PacketS_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1

   fprintf(fid1,'%d ', PacketS(j,1:2));

   fprintf(fid1,'\n');
end

fclose(fid1);


%get unique source and destination pair

for i=1: size_C

   IP(i,1)= C{2}(i);

   IP(i,2)= C{4}(i);
end

u = sortrows(IP);

[u{end+1,:}] = deal('');

j = ~all(cellfun(@strcmp, u(1:end-1,:), u(2:end,:)),2);

u=u(j,:);

[m1, n1]=size(u);

filename = 'N:\DARPA1998\TrainingDataset\week5\monday\TCP_IP_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1
   fprintf(fid1,'%s %s', u{j,1:2});

   fprintf(fid1,'\n');

end
```

```
fclose(fid1);
% compute frequency of each ip address
 C2=unique(C{2});
 size_C2 = size(C2);
 for k=1:1:size_C2
   C2{k,2}=0;
 end
 for i=1:1:size_C2
     for j=1:1:size_C
         if (strcmp(C2{i},C{2}(j))==1)
             C2{i,2}= C2{i,2}+1;
         end
     end
 end
%UDP traffic
fin=fopen('N:\DARPA1998\TrainingDataset\week5\monday\UDP_Week5_Monday.txt');
C2=textscan(fin,'%f %f %s %f %s %f %f');
size_C = size(C2{1});
%get occurance of port and size
PortSU=unique(C2{4});
P4= hist(C2{4}, PortSU);
PortSU(1:end,2)= P4(1:end);
[m1, n1]=size(PortSU);
```

```
filename =
'N:\DARPA1998\TrainingDataset\week5\monday\UDP_PortS_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1

   fprintf(fid1,'%d ', PortSU(j,1:2));

   fprintf(fid1,'\n');

end

fclose(fid1);

%port destination

PortDU=unique(C2{6});

P5= hist(C2{6}, PortDU);

PortDU(1:end,2)= P5(1:end);

[m1, n1]=size(PortDU);

filename =
'N:\DARPA1998\TrainingDataset\week5\monday\UDP_PortD_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1

   fprintf(fid1,'%d ', PortDU(j,1:2));

   fprintf(fid1,'\n');
end

fclose(fid1);

%size of packet

PacketSU=unique(C2{7});

P6= hist(C2{7}, PacketSU);
```

```
PacketSU(1:end,2)= P6(1:end);

[m1, n1]=size(PacketSU);

filename =
'N:\DARPA1998\TrainingDataset\week5\monday\UDP_PacketSU_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1

   fprintf(fid1,'%d ', PacketSU(j,1:2));

   fprintf(fid1,'\n');
end
fclose(fid1);

%get unique source and destination pair

for i=1: size_C

   IP2(i,1)= C2{3}(i);

   IP2(i,2)= C2{5}(i);

End

u2 = sortrows(IP2);

[u2{end+1,:}] = deal('');

j = ~all(cellfun(@strcmp, u2(1:end-1,:), u2(2:end,:)),2);

u2=u2(j,:);

[m1, n1]=size(u2);

filename =
'N:\DARPA1998\TrainingDataset\week5\monday\UDP_IP_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1

   fprintf(fid1,'%s %s', u2{j,1:2});
```

```
    fprintf(fid1,'\n');

end

fclose(fid1);

%ICMP traffic

fin=fopen('N:\DARPA1998\TrainingDataset\week5\monday\ICMP_Week5_Monday.txt');

C3=textscan(fin,'%f %f %f %s %s %f');

size_C = size(C3{1});

PacketSI=unique(C3{6});

P7= hist(C3{6}, PacketSI);

PacketSI(1:end,2)= P7(1:end);

[m1, n1]=size(PacketSI);

filename =
'N:\DARPA1998\TrainingDataset\week5\monday\ICMP_PacketSI_Week5_Monday.txt';
fid1 = fopen(filename, 'w');

for j=1:1:m1

    fprintf(fid1,'%d ', PacketSI(j,1:2));

    fprintf(fid1,'\n');

end

fclose(fid1);

%get unique source and destination pair

for i=1: size_C

    IP3(i,1)= C3{4}(i);

    IP3(i,2)= C3{5}(i);

end
```

```
u3 = sortrows(IP3);

[u3{end+1,:}] = deal('');

j = ~all(cellfun(@strcmp, u3(1:end-1,:), u3(2:end,:)),2);

u3=u3(j,:);

[m1, n1]=size(u3);

filename = 'N:\DARPA1998\TrainingDataset\week5\monday\ICMP_IP_Week5_Monday.txt';

fid1 = fopen(filename, 'w');

for j=1:1:m1

    fprintf(fid1,'%s %s', u3{j,1:2});

    fprintf(fid1,'\n');

end

fclose(fid1);

%port frequency computation for udp traffic

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\portSU\Combine_UDP_port_all
.txt');

C2=textscan(fin,'%f %f');

size_C2 = size(C2{1});

for i=1: size_C2(1)

    P_U{i,1}= C2{1}(i);

    P_U{i,2}= C2{2}(i);

end

sum_port=sum([P_U{:,2:2}]);

for i=1: size_C2(1)

    P_U{i,3}= (P_U{i,2}/sum_port);
```

```
end

% read IP address frequency for udp traffic

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\IPU\Combine_UDP_all.txt');

C1=textscan(fin,'%s %s %f');

size_C1 = size(C1{1});

for i=1: size_C1(1)

   C_IP(i,1)= C1{1}(i);

   C_IP(i,2)= C1{2}(i);

   C_IP{i,3}= C1{3}(i);
end

sum_IP_oc=sum([C_IP{:,3:3}]);

for i=1: size_C1(1)

C_IP{i,4}= C_IP{i,3}/sum_IP_oc;

end

% read packet size frequency

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\packetU_size\Combine_U
DP_PacketSU_all.txt');

C3=textscan(fin,'%f %f');

size_C3 = size(C3{1});

for i=1: size_C3(1)

   P_US{i,1}= C3{1}(i);

   P_US{i,2}= C3{2}(i);
end

sum_psize=sum([P_US{:,2:2}]);
```

```
for i=1: size_C3(1)

   P_US{i,3}= (P_US{i,2}/sum_psize);
end

% read IP address frequency for icmp traffic

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\IPI\Combine_IP_ICMP_all.txt');

C1=textscan(fin,'%s %s %f');

size_C1 = size(C1{1});

for i=1: size_C1(1)

   C_IP(i,1)= C1{1}(i);

   C_IP(i,2)= C1{2}(i);

   C_IP{i,3}= C1{3}(i);
end

sum_IP_oc=sum([C_IP{:,3:3}]);

for i=1: size_C1(1)

C_IP{i,4}= C_IP{i,3}/sum_IP_oc;

end

% packet size frequency for icmp traffic

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\packetI_size\Combine_ICMP_al
l_Week.txt');

C3=textscan(fin,'%f %f');

size_C3 = size(C3{1});

for i=1: size_C3(1)

   P_US{i,1}= C3{1}(i);

   P_US{i,2}= C3{2}(i);
end
```

```
sum_psize=sum([P_US{:,2:2}]);

for i=1: size_C3(1)

   P_US{i,3}= (P_US{i,2}/sum_psize);
end

% read frequency of each IP address and compute occurrence rate of it for TCP

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\IP\Combine_IP_all.txt');

C1=textscan(fin,'%s %s %f');

size_C1 = size(C1{1});

for i=1: size_C1(1)

   C_IP(i,1)= C1{1}(i);

   C_IP(i,2)= C1{2}(i);

   C_IP{i,3}= C1{3}(i);
end

sum_IP_oc=sum([C_IP{:,3:3}]);

for i=1: size_C1(1)

C_IP{i,4}= C_IP{i,3}/sum_IP_oc;

end

% read frequency of each port and compute occurrence rate of it for TCP

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\portS\Combine_port_all.txt');

C2=textscan(fin,'%s %f');

size_C2 = size(C2{1});

for i=1: size_C2(1)

   P_U{i,1}= C2{1}(i);

   P_U{i,2}= C2{2}(i);
end
```

```
sum_port=sum([P_U{:,2:2}]);

for i=1: size_C2(1)

   P_U{i,3}= (P_U{i,2}/sum_port);
end

% read frequency of each packet size and compute occurrence rate o fit for TCP

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\packet_size\Combine_all.txt');

C3=textscan(fin,'%f %f');

size_C3 = size(C3{1});

for i=1: size_C3(1)

   P_US{i,1}= C3{1}(i);

   P_US{i,2}= C3{2}(i);

end

sum_psize=sum([P_US{:,2:2}]);

for i=1: size_C3(1)

   P_US{i,3}= (P_US{i,2}/sum_psize);

end
```

A.5 The code below applies score mechanism to create history for TCP traffic

```
% read source/destination IP address, size of packet and port number for each
day incoming traffic

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\TCP_connection\Combine_TCP
_all_history.txt');

C=textscan(fin,'%s %s %s %s %f');

size_C = size(C{1});

for i=1: size_C
```

```
    Source_IP(i,1)= C{1}(i);
    PortS(i,1)= C{2}(i);
    Dest_IP(i,1)= C{3}(i);
    PortD(i,1)= C{4}(i);
    SizeU(i,1)= C{5}(i);
end
% read occurrence rate for each corresponding IP address
fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\IP\Combine_IP_all.txt');
C1=textscan(fin,'%s %s %f');
size_C1 = size(C1{1});
for i=1: size_C1(1)
    C_IP(i,1)= C1{1}(i);
    C_IP(i,2)= C1{2}(i);
    C_IP{i,3}= C1{3}(i);
end
sum_IP_oc=sum([C_IP{:,3:3}]);
for i=1: size_C1(1)
C_IP{i,4}= C_IP{i,3}/sum_IP_oc;
end
% read occurrence rate for each corresponding port number

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\portS\Combine_port_all.txt');
C2=textscan(fin,'%s %f');
size_C2 = size(C2{1});
```

```
for i=1: size_C2(1)
    P_U{i,1}= C2{1}(i);
    P_U{i,2}= C2{2}(i);
end
sum_port=sum([P_U{:,2:2}]);
for i=1: size_C2(1)
    P_U{i,3}= (P_U{i,2}/sum_port);
end
% read occurrence rate for each corresponding size of packet
fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\packet_size\Combine_all.txt');
C3=textscan(fin,'%f %f');
size_C3 = size(C3{1});
for i=1: size_C3(1)
    P_US{i,1}= C3{1}(i);
    P_US{i,2}= C3{2}(i);
end
sum_psize=sum([P_US{:,2:2}]);
for i=1: size_C3(1)
    P_US{i,3}= (P_US{i,2}/sum_psize);
end
% apply score mechanism for each corresponding IP address according to ip address C_IP{},
size of packet SizeU{}, and port number frequencies P_U{}
for k=1:1:size_C
```

```
    j=1;
    while ~(strcmp(Source_IP(k,1),C_IP(j,1)) &&
strcmp(Dest_IP(k,1),C_IP(j,2)))
        j=j+1;
    end
     if (C_IP{j,3}> 500)
         Score{k,1}=4*C_IP{j,4};
     else
         if (C_IP{j,3}> 100)
             Score{k,1}=3*C_IP{j,4};
         else
             if (C_IP{j,3}> 20)
                 Score{k,1}=2*C_IP{j,4};
             else
                 Score{k,1}=1*C_IP{j,4};
             end
         end
     end
    m=1;
    while ~strcmp(PortS(k,1),P_U{m,1})
        m=m+1;
    end
        if P_U{m,3}>.70
          Score{k,2}=4*P_U{m,3};
```

```
else
        if P_U{m,3}> .50
            Score{k,2}=3*P_U{m,3};
        else
            if P_U{m,3}> .30
                Score{k,2}=2*P_U{m,3};
            else
                Score{k,2}=1*P_U{m,3};
            end
        end
    end
n=1;
while ~(SizeU(k,1)==P_US{n,1})
    n=n+1;
end
    if P_US{n,3}> .70
        Score{k,3}=4*P_US{n,3};
    else
        if P_US{n,3}> .50
            Score{k,3}=3*P_US{n,3};
        else
            if P_US{n,3}> .30
                Score{k,3}=2*P_US{n,3};
```

```
            else
                Score{k,3}=1*P_US{n,3};

            end
        end
    end
end
%compute overall score
for i=1:1: size_C(1)
   Score{i,4}= sum([Score{i,1:3}]);
end
t=0;
for i=1:1:size_C(1)
   if Score{i,4}> .35
      t=t+1;
      history{1}(t)= Source_IP(i,1);
      history{2}(t)= Dest_IP(i,1);
   end
end
filename = 'N:\DARPA1998\TrainingDataset\DARPA98analyze\TCP_History_169.txt';
fid1 = fopen(filename, 'w');
for i=1:1:t
  fprintf(fid1,'%s ',history{1}{i});
```

```
   fprintf(fid1,'%s ',history{2}{i});

   fprintf(fid1,'\n');

end

fclose(fid1);
```

C.6 The code below checks if the attack can pass the created history

```
%read created history C{}, and attack traffic C2{}
%udp packets

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\Udp_history\UDP_History_size.txt');

C=textscan(fin,'%s %s');

size_C = size(C{1});

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\Testing_Trace\attack_ip.txt');

C2=textscan(fin,'%s %s');

size_C2 = size(C2{1});

k=0;

for i=1:1:size_C(1)

flag=0;

j=0;

   while (j<size_C2(1)) && (flag==0)

      j=j+1;

      if strcmp(C{1}(i),C2{1}(j)) && strcmp(C{2}(i),C2{2}(j))

         k=k+1;

         C3{1}(k)=C{1}(i);
```

```
        C3{2}(k)=C{2}(i);
        flag=1;
    end
  end
end

%read created history C3{}, and attack traffic C2{}
%tcp packets

filename =
'N:\DARPA1998\TrainingDataset\DARPA98analyze\Testing_Trace\Icmp\pass_history_trace_size.txt';

fid1 = fopen(filename, 'w');

for j=1:1:k
  fprintf(fid1,'%s ',C3{1}{j});
  fprintf(fid1,'%s ',C3{2}{j});
  fprintf(fid1,'\n');
end

fclose(fid1);

fin=fopen('TCP_all_week_f24.txt');

C=textscan(fin,'%s %s');

size_C = size(C{1});

fin=fopen('TCP_History_168.txt');

C2=textscan(fin,'%s %s');

size_C2 = size(C2{1});

k=0;

for i=1:1:size_C(1)
```

```
flag=0;

j=0;

   while (j<size_C2(1)) && (flag==0)

      j=j+1;

      if strcmp(C{1}(i),C2{1}(j)) && strcmp(C{2}(i),C2{2}(j))

        k=k+1;

        C3{1}(k)=C{1}(i);

        C3{2}(k)=C{2}(i);

        flag=1;

      end

   end

end

filename = 'pass_Tcp_history_trace.txt';

fid1 = fopen(filename, 'w');

for j=1:1:k

  fprintf(fid1,'%s ',C3{1}{j});

  fprintf(fid1,'%s ',C3{2}{j});

  fprintf(fid1,'\n');

end
fclose(fid1);

%read created history C{}, and attack traffic C2{}
%icmp packets

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\icmp_history\icmp_Histor
y_size.txt');
```

```
C=textscan(fin,'%s %s');

size_C = size(C{1});

fin=fopen('N:\DARPA1998\TrainingDataset\DARPA98analyze\Testing_Trace\attack_ip.txt');

C2=textscan(fin,'%s %s');

size_C2 = size(C2{1});

k=0;

for i=1:1:size_C(1)

flag=0;

j=0;

   while (j<size_C2(1)) && (flag==0)

      j=j+1;

      if strcmp(C{1}(i),C2{1}(j)) && strcmp(C{2}(i),C2{2}(j))

         k=k+1;

         C3{1}(k)=C{1}(i);

         C3{2}(k)=C{2}(i);

         flag=1;

    end

   end

end
```

A.7 The code below checks normal detection rate for CSU traffic including parallel programming

```
parpool(8)
```

```
% reading incoming traffic file a{}

load('C:\Users\negar\Documents\Data\History\3oct_history\02_27\ipslist.mat')

% reading history file H{}

fid=fopen('C:\Users\negar\Documents\Data\History\3oct_history\02_27\History_3
oct-IP-data-2015-02-20_26.txt');

H = textscan(fid, '%s');

fclose(fid);

% search for IP and collect IP address match with created history

result=0;

size_H = size(H{1});

size_C = size(c);

count=0;

C=zeros(size_H);

parfor j=1:1:size_H

  ixn = find(strcmp(a,H{1}(j)));

  if ~isempty(ixn)

  C(j)=c(ixn);

  count=count+1;

  end

end

result=sum(C(:));

% write the result

filename =
'C:\Users\negar\Documents\Data\History\3oct_history\02_27\Match_History_27_1Copy.txt';
```

```
fid3 = fopen(filename, 'w');

fprintf(fid3,'%f\n', result);

fclose(fid3);

filename =
'C:\Users\negar\Documents\Data\History\3oct_history\02_27\Match_Uniq_IP_27_1Copy.txt';

fid4 = fopen(filename, 'w');

fprintf(fid4,'%f\n', count);

fclose(fid4);
```

A.8 The code below creates unique list of IP address from history and incoming traffic for CSU traffic. Then history accuracy check according to that

```
% reading incoming traffic file a{}

FID = fopen('C:\Users\negar\Documents\Data\History\3oct_history\3oct-IP-data-
2015-02-28.0800.txt', 'r');

if FID == -1, error('Cannot open file'), end

Data = textscan(FID, '%s', 'delimiter', '\n', 'whitespace', '');

CStr = Data{1};

fclose(FID);

size(CStr)

% remove the repeated ips

[a,~,c]=unique(CStr);

clear CStr Data FID

c=accumarray(c,1);

% find match IP between created history H{} and incoming traffic AD{}
```

```
fid=fopen('C:\Users\negar\Documents\Data\History\Match_History\02_28\all-IP-2015-02-28.txt');

AD = textscan(fid, '%s');

size(AD{1})

fid2=fopen('C:\Users\negar\Documents\Data\History\History_all-IP-data-2015-02-25_27.txt');

H = textscan(fid2, '%s');

fclose(fid2);

size(H{1})

Result=intersect(AD{1},H{1});

filename = 'C:\Users\negar\Documents\Data\History\3days_history\Match_Uniq_IP_02-28.txt';

fid3 = fopen(filename, 'w');

fprintf(fid3,'%s\n', Result{:});

fclose(fid3);
```

A.9 The code below creates and test Compacted Bloom Filter

```
% number of elements

n=100 ;

x=15;

% size of Bloom filter

m=n*x ;

Fill_ratio=0;

Fill_ratio_new=0;

Correct_counter=0;

counter_1bit_change=0;
```

```
counter_2bit_change=0;

counter_3bit_change=0;

counter_4bit_change=0;


counter_5bit_change=0;

counter_6bit_change=0;
% Create Bloom Filter A{} from created history of IP address C{}

A(m,1)=0;

set_bit=0;

fin=fopen('C:\Users\negar\Documents\MATLAB\tcp_udp_icmp_history1901_3partip_100.txt');

C=textscan(fin,'%s\n');

size_C = size(C{1});

for i=1:1:size_C

D=HashFunction(char(C{1}(i)));

set_bit= mod(D,m)+1;

A(set_bit,1)=1;

D=HashFunction2(char(C{1}(i)));

set_bit= mod(D,m)+1;

A(set_bit,1)=1;

D=HashFunction3(char(C{1}(i)));

set_bit= mod(D,m)+1;

A(set_bit,1)=1;

D=HashFunction4(char(C{1}(i)));

set_bit= mod(D,m)+1;
```

```
A(set_bit,1)=1;
D=HashFunction5(char(C{1}(i)));
set_bit= mod(D,m)+1;
A(set_bit,1)=1;
D=HashFunction6(char(C{1}(i)));
set_bit= mod(D,m)+1;
A(set_bit,1)=1;
D=HashFunction7(char(C{1}(i)));
set_bit= mod(D,m)+1;
A(set_bit,1)=1;
D=HashFunction8(char(C{1}(i)));
set_bit= mod(D,m)+1;
A(set_bit,1)=1;
D=HashFunction9(char(C{1}(i)));
set_bit= mod(D,m)+1;
A(set_bit,1)=1;
D=HashFunction10(char(C{1}(i)));
set_bit= mod(D,m)+1;
A(set_bit,1)=1;
disp(hash)
dec2bin(hash)
end
fclose(fin);
```

```
for i=1:m
  if A(i,1)==1
      Fill_ratio=Fill_ratio+1;
   end
end

% End Create Bloom Filter
% Create Compressed Bloom Filter
CBF(n,1)=0;
% New Bloom Filter B{}
B(m,1)=0;
False_counter=0;
keep_j=0;
for i=1:n
   j=i;
   q=1;
   counter_bit_set=0;
   while q <= x
     if A(j,1)==1
       counter_bit_set=1+counter_bit_set;
       keep_j=q;
     end
      j=j+n;
      q=q+1;
   end
```

```
if counter_bit_set==x

    CBF(i,1)=9;

    Correct_counter=Correct_counter+1;

elseif counter_bit_set > floor(x/2)

    CBF(i,1)=8;

    p=i;
    for k=1:x

    B(p,1)=1;

    p=p+n;

    end

elseif (counter_bit_set <= floor(x/2))

    if (counter_bit_set == floor(x/2)-5 || counter_bit_set == floor(x/2)-4 || counter_bit_set ==
floor(x/2)-3 || counter_bit_set == floor(x/2)-2 ||counter_bit_set == floor(x/2)-1 || counter_bit_set
== floor(x/2))

    False_counter=False_counter+1;

    %Count Number of bit change to 0

      if (counter_bit_set == floor(x/2)-5)

            counter_1bit_change=1+ counter_1bit_change;

      end

      if (counter_bit_set == floor(x/2)-4)

            counter_2bit_change=1+ counter_2bit_change;

      end

      if (counter_bit_set == floor(x/2)-3)

         counter_3bit_change=1+ counter_3bit_change;
```

```
    end
    if (counter_bit_set == floor(x/2)-2)
        counter_4bit_change=1+ counter_4bit_change;
    end
    if (counter_bit_set == floor(x/2)-1)
        counter_5bit_change=1+ counter_5bit_change;
    end
    if (counter_bit_set == floor(x/2))
        counter_6bit_change=1+ counter_6bit_change;
    end
end
%count number of bit change to 1
if (counter_bit_set == 1)
Correct_counter=Correct_counter+1;
end
if (counter_bit_set == 0)
  p=i;
  Correct_counter=Correct_counter+1;
  for k=1:x
     B(p,1)=0;
     p=p+n;
  end
  else
    CBF(i,1)=keep_j;
```

```
            p=i;
            for k=1:x
               if k== keep_j
                B(p,1)=1;
               else
                B(p,1)=0;
               end
            p=p+n;
            end
         end
    end
end
for i=1:m
    if B(i,1)==1
        Fill_ratio_new=Fill_ratio_new+1;
    end
end
% compute false positive rate of compacted Bloom filter
Fill_ratio_new_percent=Fill_ratio_new/m;
Fill_ratio_percent=Fill_ratio/m;
False_poitive_new= Fill_ratio_new_percent^2;
False_poitive= Fill_ratio_percent^2;
filename = 'C:\Users\negar\Documents\MATLAB\CompressBloomfilter.txt';
```

```
fid = fopen(filename, 'w');

for j=1:m

   fprintf(fid, '%d\n', B(j,1));

end
```

A.10 The code below is Linux scripts to read and extract corresponding features from CSU, DARPA CAIDA and university of Auckland dataset

```
// Read Argus file and Extract incoming traffic to CSU (129.82.*.*)
ra -r 2015-02-01.0800.hVwjYxK.lander4.argus.gz  -w - | ra dst net 129.82.00.00/* -s +1dur +tcprtt +bytes +spkts +dpkts > /raid6/negar/all-data-csu-feb15/all-data-2015-02-01.0800.txt

// Extract Source IP, Destination, Port#, Size of packet for each day traffic
cat  /raid6/negar/all-data-csu-feb/all-data-2015-02-01.0800.txt | cut -d' ' -f27 | cut -f1,2,3,4 -d'.' | sort | sed 's/[!Flgs]//'g |sed 's/[!Proto]//'g| sed '/^$/d;s/[[:blank:]]//g' > /raid6/negar/all-data-csu-feb/27.txt

cat 24.txt 25.txt 26.txt 27.txt | sed 's/[!-]//'g | sed 's/[!<->]//'g | sed 's/[!?]//'g |sed '/^$/d;s/[[:blank:]]//g'|sort > all-IP-data-2015-02-01.0800.txt

// Combine all traffic based on window size. For example, combine 2 weeks incoming traffic
cat all-IP-2014-01-30.txt all-IP-2014-01-18.txt all-IP-2014-01-18.txt all-IP-2014-01-19.txt all-IP-2014-01-20.txt all-IP-2014-01-21.txt all-IP-2014-01-22.txt all-IP-2014-01-23.txt all-IP-2014-01-24.txt all-IP-2014-01-25.txt all-IP-2014-01-26.txt all-IP-2014-01-27.txt all-IP-2014-01-28.txt all-IP-2014-01-29.txt > all_data-2014-01-18_31.txt

// Count number of occurrence and create signature for each extracted features
 awk '{for(w=1;w<=NF;w++) print $w}' comb_all-IP-data-2015-02-04.txt | sort | uniq -c | sort -nr > freq_all-IP-data-2015-02-04.txt

// Extract one type of traffic for example UDP traffic
ra -r 2015-02-01.0800.hVwjYxK.lander4.argus.gz - udp -w - | ra dst net 129.82.00.00/240 -s +1dur +tcprtt +bytes +spkts +dpkts > /raid1/negar/all-data-csu-feb/udp-data-2015-02-01.0800.txt

all-data-csu-feb]$ cat  /raid1/negar/all-data-csu-feb/udp-data-2015-02-01.0800.txt | cut -d' ' -f26 | cut -f5 -d'.' | sort | sed 's/[!Flgs]//'g |sed 's/[!Proto]//'g| sed '/^$/d;s/[[:blank:]]//g' > /raid1/negar/all-data-csu-feb/26.txt

// Reading CAIDA traffic and extract features such as Source IP, Destination, Port#, Size of packet as well as TCP SYN and ACK flags
```

tcpdump -vvnnS -r equinix-chicago.dirA.20110607-235000.UTC.anon.pcap 'tcp[13]=18' -tt | sed 's/F.*q//'| sed 's/a.*k//'| sed 's/(.*h//' | sed 's/I.*P//'| sed 's/w.*h//' | sed 's/I.*P//' | sed 's/>//' | sed 's/://' | sed 's/)//' | cut -d' ' -f1,4,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | sed "s/:.*, / /g" | sed 's/,//'| sed 's/,//' | sed 's/ //g' > dirA.20110607-235000_tcp18.txt

// Reading University of Auckland dataset
./tracesplit -f "tcp[13]=16" legacyatm:/mnt/c20311/wits/20010318-020000-0.gz pcapfile:/mnt/c20311/tmp2/convert20010318-020000-0.pcap.gz

// Extract time, Source IP, Destination, Port#, Size of packet for each day as well as TCP SYN and ACK flags
tcpdump -vvnnS -r convert20010314-020000-0.pcap.gz 'tcp[13]=16' -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > /mnt/c20311/data/data20010314-020000-0.txt

./tracesplit -f "tcp[13]=18" legacyatm:/mnt/c20311/wits/20010318-020000-1.gz pcapfile:/mnt/c20311/tmp2/convert20010318-020000-1.pcap.gz

tcpdump -vvnnS -r convert20010314-020000-1.pcap.gz 'tcp[13]=18' -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > /mnt/c20311/data/data20010314-020000-1.txt

// Extract one type of traffic for example UDP traffic from University of Auckland dataset
tcpdump -vvnnS -r udp20010315-020000-0.pcap.gz -tt | sed 's/ t.*!//'| sed 's/ I.*,//'| sed 's/ I.*)//' | sed 's/>>>.*//' | sed 's/WAR.*//' | sed 's/984622873.970670.*//' | sed 's/packetexc.*//'| sed 's/Trn.*//' | sed 's/Data:.*//' | sed 's/OpCode.*//' | sed 's/Rcode.*//' | sed 's/NmFlag.*//' | sed 's/Query.*//' | sed 's/Answer.*//' | sed 's/Author.*//' | sed 's/Address.*//' | sed 's/Addit.*//' | sed 's/uns.*//'| sed 's/[[].*//' | sed 's/Root.*//'| sed 's/[>].*//' | sed '/^$/d;s/[[:blank:]]//g' | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f2 | cut -d'.' -f1,2,3,4 > /mnt/c20311/data/udp20010315-020000-0.txt

// Extract single feature as follows
Time
tcpdump -vvnnS -r convert.pcap.gz 'tcp[13]=16' -c 1200000 -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f1 | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > /mnt/c20311/data/time20010315-020000-0.txt

Source IP
tcpdump -vvnnS -r ddostrace.to-victim.20070804_134936.pcap 'tcp[13]=16' -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed

's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f3 | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > tcp_attack20070804_134936

Destination IP

tcpdump -vvnnS -r convert20010314-020000-0.pcap.gz 'tcp[13]=16' -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f4 | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > /mnt/c20311/data/dest20010314-020000-0.txt

Sequence

tcpdump -vvnnS -r convert.pcap.gz 'tcp[13]=16' -c 30 -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f5 | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' | cut -d':' -f1 > /mnt/c20311/data/dest20010312-020000-0.txt

Acknowledge

tcpdump -vvnnS -r convert.pcap.gz 'tcp[13]=16' -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f6 | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' | cut -d':' -f1 > /mnt/c20311/data/ack20010312-020000-0.txt

Length

tcpdump -vvnnS -r convert20010314-020000-0.pcap.gz 'tcp[13]=16' -tt | sed 's/ t.*!//' | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' |sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11,13 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f7 | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > /mnt/c20311/data/length20010314-020000-0.txt

icmp

./tracesplit -f "icmp" legacyatm:/mnt/c20311/wits/20010329-020000-0.gz pcapfile:/mnt/c20311/udp/icmp20010329-010000-0.pcap.gz

tcpdump -vvnnS -r icmp20010329-010000-0.pcap.gz -tt | sed 's/ t.*!//'| sed 's/ I.*,//'| sed 's/ I.*)//' | cut -d' ' -f1,5,7 | awk 'ORS=NR%2?" ":"\n"' | cut -d' ' -f3 | sed 's/://' > /mnt/c20311/data/icmp/icmp20010329-010000-0.txt

// Combine all traffic based on window size. For example, combine 2 weeks incoming traffic

cat filtercmp20010409-010000-0.txt filtercmp20010327-010000-0.txt filtercmp20010328-010000-0.txt icmp20010329-010000-0.txt filtercmp20010330-010000-0.txt filtercmp20010331-010000-0.txt filtercmp20010401-010000-0.txt filtercmp20010402-010000-0.txt filtercmp20010403-010000-0.txt filtercmp20010404-010000-0.txt filtercmp20010405-010000-0.txt filtercmp20010406-010000-0.txt filtercmp20010407-010000-0.txt filtercmp20010408-010000-0.txt | sort | uniq > icmp_history_2709.txt

// Create list of unique IP addresses

cat tcp_history_2103.txt /mnt/c20311/data/udp/udp_history_2103.txt /mnt/c20311/data/icmp/icmp_history_2103.txt | sort | uniq > tcp_udp_icmp_history_2103.txt

// Find list of IP addresses that are matched between two files. For instance between IP address history and corresponding day traffic
grep -Fxv -f history2608_09.txt /mnt/c20311/data/tcp_sip_2nextweek/tcp_udp_icmp_409.txt

// Reading DARPA dataset and extract time, Source IP, Destination, Port#, Size of packet for each day as well as TCP SYN and ACK flags
tcpdump -vvnnS -r tcpdump.pcap.gz 'tcp[13]=16' -tt -c 1000 | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' | sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | sed "s/:.*, / /g" | sed 's/,//' | sed 's/,//' > /mnt/c20305Negar/DARPA1998/TrainingDataset/week1/monday/tcp16.txt

tcpdump -vvnnS -r outside.tcpdump 'tcp[13]=18' -tt | sed 's/F.*q//' | sed 's/a.*k//' | sed 's/(.*)//' | sed 's/I.*)//' | sed 's/w.*h//' | sed 's/I.*P//' | sed 's/>//' | sed 's/://' | cut -d' ' -f1,5,7,9,11 | awk 'ORS=NR%2?" ":"\n"' | sed "s/:.*, / /g" | sed 's/,//'| sed 's/,//' > /mnt/c20305Negar/DARPA1998/TrainingDataset/week1/monday/tcp18.txt