FAPA: FLOODING ATTACK PROTECTION ARCHITECTURE IN A CLOUD SYSTEM

by

KAZI ZUNNURHAIN

SUSAN VRBSKY, COMMITTEE CHAIR
RAGIB HASAN
XIAOYAN HONG
MARCUS BROWN
JINGYUAN ZHANG

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2014

ABSTRACT

The rate of acceptance of clouds each year is making cloud computing the leading IT computational technology. While cloud computing can be productive and economical, it is still vulnerable to different types of external threats, one of which is a Denial of Service (DoS) attack. DoS attacks have long been an open security problem of the internet. Most proposed solutions to address DoS attacks require upgrades in routers, modification in the BGP (Border Gateway Protocol), usage of additional control bits in the IP packets, or adjustments to legacy routers in the routing path. It is extremely difficult to manipulate all these criteria, considering that the internet, and potentially a cloud, consists of a very large number of autonomous systems with routers from different vendors deployed over decades. Authentication protocols are typically implemented by some of the leading companies manufacturing DoS prevention routers. However, authentication protocols and embedded digital signatures are very expensive and vulnerable. This is contrary to the benefits of renting a cloud system, which is to save capital expenditure as well as operational expenditure.

Rather than depending on cloud providers, we proposed a model, called FAPA (Flooding Attack Protection Architecture), to detect and filter packets when DoS attacks occur. FAPA can be deployed at different levels of the system, such as at the user's end. FAPA can run locally on top of the client's terminal and is independent of the provider's cloud machine. There is no need to deploy any expensive packet capturing tools nor does it require any embedded digital signature inside the packets. There is no additional charge from the provider's end since the application runs in the customer's end. Moreover, automatic message propagation invokes the cloud server to trace the source or adversary.

In FAPA, detection of denial of service is handled by the periodic analysis of the traffic behavior from the raw packets. It generates an alarm if any DoS attack is detected and removes flooding by filtering. Because FAPA is employed on the client's side, customers have control over traffic trends, which is absent in other DoS prevention approaches. FAPA is comprised of five individual modules, where each module has an assigned task in detecting DoS attacks and removing threats by filtering the spoof packets. A module fetches the traffic packets and does the unwrapping. Another module records the pertinent parameters of network packets.

Implementation of a FAPA prototype and experimental results has demonstrated the feasibility of FAPA. From our initial experiments we observed that in the event of a DoS attack, some of the network parameters change. Hence, in FAPA a separate module is dedicated for storing information about traffic behavior. If FAPA observes any inconsistent traffic behavior, it invokes the filtering modules to remove the compromised network packets. FAPA filtering detects the threat by using previously recorded information. FAPA filtering was implemented for a cluster environment and we ran experiments to determine its effectiveness. The filtering module was then modified to run in a cloud environment and was able to handle a large set of network packets. We investigated the impact of DDoS attacks on co-resident virtual machines and their neighbors. Later we conducted DDoS attacks from a commercially launched public cloud onto private cloud instances to observe the amplification of an attack and checked the efficiency of FAPA in terms of filtering those non legitimate packets. We also measured FAPA performance in terms of false positive and false negative rates. We deployed several commercially used stress testing tools to observe FAPA's performance. Both in the cloud and on the cluster, our

experimental results demonstrated that FAPA was able to detect and filter packets to successfully remove a DoS attack.

Keywords-DDoS, Cloud, Autonomous System (AS), Behavioral Pattern, Compromised, Bandwidth, Throughput, Transfer Packets.

DEDICATION

To my mother, and my late father who suffered a lot from carcinoma malignancy

&

To the pioneers of computer science that paved the way before me.

ACKNOWLEDGMENTS

My appreciation goes to my professor and adviser, Dr. Susan Vrbsky. I am very thankful for Dr. Vrbsky's involvement in my academic and professional life for the past three years. During my time here Dr. Vrbsky helped me become a better computer science researcher and teacher. When I started in UA, Dr. Vrbsky was the one to invite me in her lab and motivated me to join her research group that would evolve into the Cloud and Cluster Computing Group here at the University of Alabama. As I was progressing through the program, Dr. Vrbsky was guiding me to participate in many great conferences around the United States. Even though most of our interaction involved research topics, I also learned much about the aspects of working in the academic environment. Those insights are invaluable and would be difficult to find anywhere else. As my mentor she guided me through each and every step to reach my goal, but I must mention I learned more from her than just research. It was a great opportunity to work under such a motivating person. As I progress through my career, I plan on remaining in contact with the person that most influenced the direction of my life. Thank you, Dr. Vrbsky, for all the inspiration and motivations through these years of research and teaching.

I would like to express my gratitude to Dr. Ragib Hasan for serving as a co-advisor in my PhD committee. His advice and thoughtful ideas opened different paths for my research on cloud computing. He gave me a lot of new ideas regarding the new cloud services and their vulnerabilities. Dr. Ragib Hasan has been a great source of information in last two years of my PhD study. It was a privilege and opportunity for me to have him as my co-advisor.

To Dr. Xiaoyan Hong, I had the opportunity to work with her in her networking courses. We did some research on ad-hoc networking and I came to know a lot about computer networking during this study. She has been a wonderful mentor. Dr. Marcus Brown, thank you

for always providing a positive aspect on any situation we discussed. Dr. Jingyuan Zhang, thank you for your insights and great suggestions for my research.

To Kathy DeGraw and Jamie Thompson, thank you for everything that you do to make this department what it is! We would all surely fall without your support. I am also privileged to be associated with the current cloud and cluster computing group at UA: Gabriel Loewen, Jeffrey Robinson, Dr. Ashfakul Islam, and Todd McCutchen. I really enjoyed working with this group and each and every one of them was very helpful regarding any matter. I want to thank Dr. Michael Galloway and Gabriel Loewen, for all their expertise on clouds. Their thoughts and knowledge helped me a lot while building my private cloud for my experiments. Thanks to Jeffrey Robinson, for all the candies he bought every week for the lab. Also I would like to thank the new member of our lab, Todd McCutchen for the positive thoughts. I will always remember my time in this lab. Our first "Turtle" died, so what, we are going for the "Second Turtle"!

To others in the Ph.D. program, Dr. Ferosh Jacob, Amiangshu Boshu, Debarshi Chatterji, thank you for your discussions about computer science and software engineering topics, and approaches we should make in relaying these topics to students in the classes.

Finally, I would like to thank the financial support provided by the Department of Computer Science at UA. Thank you for the opportunity to pursue my Ph.D. in the area of private cloud computing security issues and strengthen my abilities as a computer science instructor.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

INTRODUCTION

In the field of computation, there have been many approaches for enhancing the parallelism and distribution of resources for the advancement and acceleration of data utilization. Data clusters, distributed database management systems, data grids, and many more mechanisms have been introduced. Cloud computing is currently emerging as a mechanism for high level computation, as well as serving as a storage system for resources. Clouds allow users to pay for whatever resources they use, allowing users to increase or decrease the amount of resources requested as needed. Cloud servers can be used to motivate the initiation of a business and ease its financial burden in terms of capital expenditure and operational expenditure. According to Cisco guard global network, by 2016 cloud data centers will occupy 64% of the total data centers in the market with a capacity more than 6.5 Zetta Bytes [83]. Also, a study by Armando investigated and reported that SaaS clouds would be a $6 Billion market by the end of 2013 [41], but the actual size is now $6.9 Billion according to Asia Pacific Cloud Market Forecast [92]. The forthcoming growth of clouds in terms of both market share and acceptance rate clearly validates the certainty of wide spread approval of clouds in the IT market, but does not guarantee the mitigation of its vulnerabilities. The three major pillars of any technology are: cost, trust and sustainability, and cloud computing is not an exception. A cloud is more economical compared to other distributed services, with one of the appealing aspects of cloud computing being pay-as-you-go. Sustainability can be ensured by maintaining a strong infrastructure with abundant availability of resources. But ensuring trust is the hardest task, because it involves securing cloud services completely (integrity) and providing sufficient

support for uninterrupted services to cloud users (availability) with complete confidentiality of users' data [85], [86].

There are three general layers that can be used to describe the services provided by a cloud system as shown in Figure 1: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). The infrastructure layer consists of all the hardware modules available in the cloud. The platform layer is responsible for hosting different environments for customer specific services. The software layer is for forwarding the customers' requests to the server through web services and browsers. All of these layers are highlighted in Figure 1. Amazon Elastic Compute Cloud, also known as EC2, is a popular example of IaaS. EC2 is a commercial web service where customers can rent their computers, run their desired applications and pay Amazon according to their usage. Google App engine is an example of PaaS and salesforce.com is an example of SaaS [35].

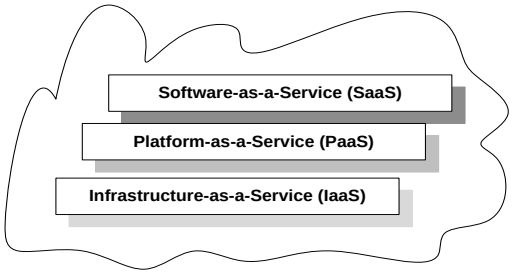


Figure 1: Principle Layers of a Cloud

There are three different types of clouds: public, private and hybrid. In a public cloud, the infrastructure is provided by the CSP (Cloud Service Provider) and cloud users have no visibility or control over the compute servers' locations [35], [37]. In a private cloud, the infrastructure is contained within one organization and not shared among other providers or vendors [38].

Organizations may host their security intensive applications in a private cloud and applications with less security concerns in an economical public cloud. This combination of hosting applications is known as a hybrid cloud [38].

Cloud computing can be viewed as the transformation into reality of a long held dream called "Computing as Utility" although experts describe it as the reinvention of the distributed mainframe model [11]. While cloud computing has received mixed reviews from its customers, it is emerging into the market with much potential. It could be the most promising improvement in IT infrastructure area in recent times, but a great deal of work is still warranted in the domain of security to minimize the threats. Many small or midsized organizations adopted cloud computing to reduce the upfront investment costs, to minimize maintenance work in IT infrastructure, to curtail the labor cost for monitoring, and to enhance on-demand capabilities. Industries are not required to plan for their IT growth in advance with the "pay as you go" feature of a cloud system. Instead, users of a cloud only need to request and pay for services as they are needed. Users of a cloud can range from single users taking advantage of the benefits of a public cloud to multiple users in a startup company utilizing a commercially rented public cloud. For example, a single user can use the cloud to store his/her personal data files or image files. A startup company can utilize the infrastructure provided by a public cloud, such as a Visual Studio framework, to develop a website consisting of multiple types of databases in the environment of the cloud provider. Utilization of a cloud for a range of services has been shown to be very cost-effective [29], [71]. However, there is a risk for not doing an assessment on security when utilizing a cloud. Before describing the main security threats in a cloud system, we will discuss the prime characteristics of the cloud.

Virtualization, elasticity, scalability, load balancing, instant service and pay-as-you-go are the main properties that convert a data center into a cloud. With virtualization the users have the privilege of running their instances with a variety of application options provided by the cloud provider, or the user's chosen applications, without any initial cost. Virtual machines (VMs) are of variable size depending on the customer's applications requirements. Instances are running on top of a monitoring system independent of the operating system in the local machine. The feature of elasticity allows the customer to increase or decrease its requests for resources as needed. The load balancing feature provides the cloud a certain extended capability, such as replacing an overloaded server or faulty system with another server and reallocating the pending tasks of the faulty server to the newly assigned server. Scalability is a feature where continuous monitoring of a customer's memory allocation or CPU utilization sends feedback to the controller to scale up the resources or scale down the facilities of each customer's sessions. Pay-as-you-go is the basic feature of cloud computing, which is letting the cloud users pay for what they have used. A customer is charged depending on the total time of use or amount of data stored. Unfortunately, these services provided by a cloud could be hindered if the system is compromised by some denial of service (DoS) or distributed denial of service (DDoS) attacks.

There are different types of DoS (Denial of Service) and DDoS (Distributed Denial of Service) attacks. Basically, spoofing is the first link in the chain of DoS. Spoofing is a mechanism where the attacker camouflages his identity behind the IP address of a legitimate user or an unused IP address of a domain that was unused for a long time. The adversary then generates a bulk amount of fake requests and sends them toward the target machine. Before reaching the server, validation

of these packets is essential. But validation takes a long time, and engages server resources. The legitimate requests starve while resources are engaged validating the spoof requests. Eventually DoS takes place, due to the enormous number of spoof packets generated by the non-legitimate users. In a large-scale distributed system, like a grid or a cloud system, if the same types of attacks take place from many network terminals targeting a single server in the distributed system, such phenomenon is known as DDoS.

IP spoofing attacks impose substantial threats for internet services. According to some surveys it has been found that every year there are about 4000 to 7000 DoS attacks taking place in different network places, such as public websites (Yahoo, Facebook), private sectors (Company websites: Aramco), financial institutes and many more [4], [5], [3], [2]. According to a survey report [6], 74.6% of the service providers strongly recommend that the most vital issue for cloud computing is security assurance. In addition, according to Prolexic's Q3 2011 report (one of the first and largest companies offering DDoS mitigation) [62], 24% of all cyber-attacks were SYN flooding. The report also stated 22% of ICMP and 19% of UDP attacks were accomplished by cyber adversaries [63]. TCP-SYN packets can clog the victim's bandwidth easily if the attacker is wise enough to exceed the maximum bandwidth capacity of the network channel. Another report from Prolexic [63], [54] stated that HTTP flooding is responsible for 88.9% of DDoS attacks in Q2 2011. Nowadays, we are intertwined with internet services in each and every way. Our dependency on the internet has made itself the most significant part of our regular life. If this is hindered by an outsider, then not only do we lose money but also our personal information could be compromised.

The main properties of a cloud can, unfortunately, enhance a DoS attack and these properties serve to distinguish a DoS attack in a cloud from traditional DoS attacks as follows. Due to the load balancing feature in a cloud, if a single adversary sends spoof packets to a cloud server, once the server becomes overloaded it will offload its validating tasks to the nearest server. The newly assigned server also eventually becomes overloaded and offloads its task to another server, thus propagating the flooding attack over the entire network. The cloud properties of elasticity and scalability provide further disadvantages during an attack. A server overloaded with enormous validation tasks will scale up to engage more of its resources and computation nodes to validate the spoof packets, continuously exhausting each server with its assigned resources. A dynamic adversary with knowledge of the targeted cloud topology and server connections can even try to compromise the complete system by using a botnet and deploy several handlers to compromise a cloud network.

A cloud's use of virtualization can also be exploited by an adversary. There exist many generic tools (*nmap, hping, wget* etc.) with which a cloud user can estimate the placement of VMs in a cloud with a high probability [64]. Unfortunately, these tools can be used by the adversaries to impair the cloud system by launching various attacks. A generic network testing tool can successfully identify a target virtual machine on the cloud with higher likelihood and instantiate VMs co-resident to the target VM to conduct a variety of attacks [91]. There are many DDoS attack tools like *Agobot, Mstream,* and *Trinoo* [91] that can also be used against a cloud. For example, *Agobot* can be used as a backdoor Trojan and/or network worms by establishing an IRC channel to the remote servers. Then a client running user friendly social networking websites can be easily controlled by a single attacker to conduct highly effective and efficient

DDoS attacks [64], and [91]. Obviously, in these examples of DoS attacks in the pay-as-you-go model of a cloud, users can be charged for services not requested. These observations of DoS vulnerabilities in a cloud environment motivated our work to design a strategy for DoS protection in a cloud system.

While currently there exist many different threats of DoS attacks, there are also different types of countermeasures that promise to protect flooding. The existing countermeasures are not cost effective in a cloud and may require an upgrade in the end systems. In this dissertation, the focus was to design a solution for a cloud system that protects against a DoS attack but has a moderate overhead cost. A model is proposed, called FAPA (Flooding Attack Protection Architecture), which detects flooding and filters packets when DoS attacks occur. FAPA can run locally on top of the client's terminal and is independent of the provider's cloud machine.

In order to design the filtering mechanism for FAPA, traffic analysis was performed to explore traffic behavior when a cloud is compromised. FAPA deploys filtering based on the traffic pattern learned from the network systems. Investigation began by analyzing the number and length of incoming TCP-SYN requests and the SYN-ACK responses from the server with and without flooding. Next, the focus was on the virtualization feature of clouds by considering the impact of flooding on the bandwidth and transfer rates of sibling (co-resident VMs) and neighbors (VMs located in a different physical node) of a compromised VMs. Information from this traffic analysis was subsequently utilized in the FAPA filtering mechanism to detect spoof IP addresses. Experiments on a private cloud utilizing the filtering mechanism were applied to

sample data collected from CAIDA (Cooperative Association for Internet Data Analysis) to measure a large amount of DoS compromised network packets.

The objective of the research is to provide a model to effectively protect against DoS attacks in clouds. The experiments performed determine the effectiveness of the FAPA filtering model in detecting and filtering spoof packets. Also, unique to this research, is the study of the effect of flooding from an external adversary on sibling virtual machines and neighbor nodes, a topic which to the best of our knowledge has not been considered previously.

This dissertation is organized as follows. Ongoing related research in the cloud for DoS/DDoS protection is presented in chapter 2. Different types of DoS prevention mechanisms, followed by a criticism of each prevention mechanism are covered in chapter 3. A demonstration of our proposed solution, which is a reactive approach and depends on the running virtual machines, appears in chapter 4. This is followed by the illustration of our experimental results and analysis of the impact of a flooding attack in chapter 5. We also analyze the impact of a DoS attack on virtual machines. In chapter 6 we describe how the results from our experiments in chapter 5 are incorporated into FAPA. Chapter 6 then presents the experiments with Amazon EC2 botnet deployment onto the private cloud instances with an analysis of FAPA's ability to filter packets in order to protect from a flooding attack. In this chapter we have also included the performance of FAPA under some industrially used stress-testing tools. Chapter 7 presents an analysis of the false positive and false negative rates of FAPA. Future directions are presented in chapter 8. Finally, the conclusions appear in chapter 9.

Chapter 2

RELATED WORK

Clouds are vulnerable to various security issues, privacy [73], [76], data stealing, resource unavailability [75], denial of service attack [66], [67] and many more. Virtualization could also be a victim of these types of threats in cloud. Many approaches have been deployed or are under construction to overcome these threats. In this section we will discuss some of the approaches of cloud security focusing on secure virtualization and intrusion detection. We identify the weaknesses of each approach and discuss the feasibility of the solution for DoS or DDoS protection in a cloud environment.

One of the prominent solutions to provide security services for a cloud environment is CloudSec [87]. Implementation of decentralized task scheduling, collaboration of server resources to defend attacks and utilization of a hypervisor to induce additional monitoring for virtual machines are the major concerns for CloudSec [87]. However, decentralization of tasks can cause additional cost for continuous communication among these collaborating organizations. Also in a heterogeneous cloud environment, implementation of collaboration for resource sharing requires extended Service Level Agreements (SLA) among the organizations for accountability purposes. A unique approach was proposed for maintaining VM security based on using VM safe libraries on a VMware ESX cloud platform [88]. Rather than focusing on filtering attack packets, the concern was for minimizing the semantic gaps between the hypervisor and the host operating system of the virtual machine. Due to the transformation of raw hardware bytes into an OS process, kernel information would require an extensive manipulation of OS global

variables [88]. This kind of approach might risk the complete platform of the cloud environment due to complications. If there exists a bug to penetrate the system, then an attacker will be able to take control of the total system. In addition, the hypervisor was considered completely trustworthy from the beginning. Also to be mentioned, hosting CloudSec inside the hypervisor [88] not only revokes the cloud client's transparency with the cloud provider but generates a potential dispute between them related to a security breach.

In [86], a multilayer overlay approach was introduced to defend against denial of service attacks based on IP addresses. Their principle approach was to define a threshold for every layer of virtual hosts, and communication among them through encryption involving Message Authentication Control (MAC) [86]. But extensive amounts of encryption (source) and decryption (destination) in a cloud environment would add additional charges to the clients, decreasing the economic benefits of a cloud. Also using thresholds instead of traffic flows might cause high false positive rates due to the dynamic nature of cloud environments. In FAPA there is no encryption, reducing the cost of communication among VMs as much as possible. A Graphic Turing test [89] could be a suitable approach to defend DoS attacks for a high traffic web server to distinguish between human interaction and automated attack zombies. Throwing an alphanumeric image challenge might defend against a zombie [89], but a modern private cloud is also vulnerable from inside attacks which involve human interaction to a great extent.

Involvement by a hypervisor is an effective approach for secure virtualization in a cloud. A hypervisor allows multiple operating systems to run concurrently on a cloud. In [81], the hypervisor-based architecture secured the guest OS and the running app from each other.

However, their strategy prevented the creation of large numbers of VMs on a physical node, contradicting one of the prominent features of cloud computing, which is its elastic nature [85], [86]. No protection or recovery plan was presented if an adversary compromises the hypervisor. Also, there was no performance evaluation for the proposed architecture [81]. To combat DDoS attacks, an ISP level solution [30] has managed to keep false positive rates as low as 2.8%. However, in their simulation they did not consider the scalable and elastic nature of the cloud, causing any DoS attack to be quickly amplified.

Virtualization is an important component of cloud computing. A virtual machine (VM) provides a guest operating system upon which the user's software can run. CUDACS [79] considers the security of VMs by utilizing underutilized GPU cores to conduct VM monitoring efficiently. However, the extensive monitoring of guest VMs results in an overhead in memory access. Lares architecture [52] for secure active monitoring uses virtualization that is deployable only on VMs running with Windows XP on Zen.

In 2009 Kai et al. proposed a P2P hierarchy location based reputation system for the protection of cloud servers and ensured data object protection in file access levels [73]. According to their method, deployment of more virtual instances divides a physical node into secure partitions. This approach decreases the possibility of DDoS attacks, as the VMs are isolated and contained from affecting other VMs. However, the authors do not discuss that although a virtual machine owns its individual virtual networks (*vnet*), it shares a common image repository system with other VMs in the same physical node. They share identical CPU resources and their network interfaces have the same bridge for communication between the physical router and the *vnet*.

Hence, if a VM is compromised in one physical node it can affect the neighbor VMs as well [8], [85]. Also, another study suggested a P2P reputation system to grant access in coarse-grained and fine-grained access control of shared resources [73]. This system is valid only for customers with relational database requests to process access level queries, which is provided by some commercial cloud providers (Amazon EC2 and Microsoft Azure). To ensure data consistency they suggested multiple replication mechanisms, but did not disclose to what extent the replication should be implemented [73]. Also, they have not considered the cost of having multiple replicas of data objects distributed geographically.

Another approach was to deploy an intrusion detection mechanism in each of the running virtual instances and record the inbound/outbound traffic into a database using SNORT [74]. According to their study, SNORT will detect intrusion if there is any spike in the sender's traffic. A honeypot deployed in each VM will then send a ping request. If there is no reply to the ping request then the sender will be considered as an adversary and all packets from that IP address will be dropped. This approach does not consider if there was a jitter or time delay due to any faulty router along the path, or if the VM went into an ideal mode after sending the last packet with a FIN bit set on [84], [85], meaning the finished packet has arrived. Without consideration of these possibilities, a legitimate user may be considered as an attacker. In the study, migration was considered from one data center to another if a VM is detected as a threat [74], but they did not consider the cost of migration or provide any experimental results to demonstrate the effectiveness of their approach.

DDoS attacks not only affect physical CPU cycles but also affect virtual cycles, create I/O bottlenecks, etc. [75]. In 2011, Sabahi proposed to provide virtualization level security in cloud environments by deploying constant monitoring of the virtual machines and guest operating systems. The author suggests the cloud user protect cloud infrastructure [75], but a cloud user is not knowledgeable about the hardware infrastructure or the physical location of the running instances. Also, the author suggested that data remanence could become a reason for cloud vulnerability, as the adversary could retrieve some information from the residue of erased data [75]. Data remanence is the residual digital data left after attempts have been taken to erase or remove them. To overcome such a possible threat, data encryption is advised, so that even after the erase an adversary cannot retrieve any information without the encrypt key used for data encryption. Injecting encryption for every data object is not a feasible solution for an economical cloud [85]. Also, the author claimed that resource increment could be a traditional solution to prevent DDoS attacks in clouds [75], without any sufficient experimental evidence, cost analysis or to what extent the resources needed to be added during a DDoS threat.

In another study, cloud security was ensured by increasing virtual machine security [76]. In their work, VMs with obsolete packages were marked with ICS (Image Creation Station) and the starting of a VM with obsolete packages was prevented by XGE (Xen Grid Engine) [59]. With the assistance of these tools, the running/idle virtual instances will be under constant surveillance to upload new patches [76]. But these updates of new packages require multiple databases to be maintained with consistency. There is a database named as Package DB for the installed version of the software patches in the VM, another database called Repo Cache DB to keep track of new versions of all software or applications running in the VM, and a meta DB to record all the meta

information, time stamp and repository names of all the virtual images [76]. Maintenance of all these DBs with consistency does not sound feasible. Other than these databases, there are also cache memories to keep the most recent query results (Result Cache) to speed up the update process, controllers to keep track of all the updates and a report generator for each update process [76]. All these databases and controllers deployed for a VM security patch update could increase the overhead cost in terms of message communication among the controllers.

HTTP DoS (HDoS) and XML DoS (XDoS) attacks are comparatively easy to accomplish but twice as hard for the security experts to protect against [77]. Chonka et al. (2011) in their study claimed to provide protection against HDoS and XDoS attacks using a trace back mechanism they named Cloud Trace Back (CTB) [77]. Their study was based upon the Deterministic Packet Marking (DPM) [12] algorithm. In their study they considered some prior assumptions, one of which was that the CTB module cannot be compromised by any attackers [77]. According to their implementation, CTB is a communication medium between the web servers and external workstations. While DPM may be able to provide protection against external adversaries through marking the flag bits of 16-bit IP headers, the authors do not consider if an insider is compromised and becomes a threat for the cloud system. The authors propose to provide protection evidence against HDoS and XDoS in a commercial cloud environment like Amazon EC2 [77]. In the experiment section they considered the recreation of HDoS and XDoS attacks that affected the Iran government websites as coordinated by the Iranian opposition party in 2009. They also discussed Twitter posting through IFrames on automated browser refreshing, which eventually crashed the web pages [77]. They rebooted the web servers in a crash event and CTB was responsible for reconstructing the HTTP packets. It was not clear how the

reconstruction took place; neither had they mentioned the number of virtual machines deployed for the experiments. For data recovery, their approach was to deploy multiple data replications [77], but considering the cost of replication, storage and time, it is not clear how this approach can be a feasible solution for a large distributed system like a cloud.

Cloud systems are not only vulnerable to extortionists but to competitors as well [76], [54], and [48]. Competitors are trying to exhaust the services of popular cloud providers so that business can be transferred to the next available service providers. In 2011 Flavio et al. proposed a continuous monitoring system for cloud environments, which they named Advanced Cloud Protection System (ACPS), and claim to provide protection against local intrusion and provide complete transparency to the cloud users [79]. They considered data and resource attacks against cloud providers (CP) and service providers (SP), and also considered data attacks against service users (SU), but did not consider the resource vulnerability in service users [79]. According to our work [85], we illustrate how the victim as well as the siblings and neighbor instances of the victim instance are also vulnerable to resource misuse during the event of a DoS attack. They neutralized any kind of timing attack but did not make it clear how this was possible [79]. They evaluated ACPS's performance by comparing the execution time of a VM in KVM [65], Eucalyptus, OpenEC [37] and ACPS [57]. However, ACPS could not outperform any of them in any metric considered (CPU, I/O or Workload) [79]. They did not disclose the number of I/O operations, amount of workloads or CPU utilization in their experiments, nor the execution time units (minutes, second or milliseconds) [79]. They also consider an attack within a guest VM, which is assumed to be conducted from a malicious application (user end) to kernel space (CPU resources) [79], but such kind of attack can impact the attackers' (SU) accounts itself.

Khorshed *et al.* [61] have tried to classify DoS attacks in cloud computing using different rule-based learning. They also claimed to identify the attacks, including a remedy from the client's perspective. The principal idea behind their experiments was to check the CPU performance and network usages of the physical machine, but usually a cloud user is unaware of the physical machine on which his/her applications are running. It is unclear how the proposed remedy removed the ongoing attacks after the detection. The strategy involved running a statistical analysis among a few classification Machine Learning algorithms, to observe which algorithm performs the best detecting the DoS attacks. It was also unclear whether in order to protect from DoS attacks, the cloud customers need to run *Weka* [44] applications in their VMs or not.

Concerning security issues in the cloud infrastructure, several intrusion detection systems (IDS) have been proposed. Roschke *et al.* [68] proposed architecture to detect DDoS attacks in a cloud system. They identified IDS management issues with respect to both host IDS (intrusion will be detected from the providers end) and network IDS (intrusion will be detected in the routers). However, their study lacks the information about how the DDoS attacks were generated and detected in the cloud infrastructure. Several architectures were proposed [50], [68] and [74] for detecting intrusion in cloud computing. To detect DDoS attacks a behavioral analysis was performed, including knowledge-based analysis. The authors did not consider the virtualization issue in each physical node, which is one of the most essential characteristics of a cloud system. A client always runs his/her applications in the virtual instances provided by the cloud infrastructure.

Bedi *et al.* [64] proposed a game theoretic defense mechanism for securing the cloud infrastructure against co-resident DoS attacks. An inside attacker compromises a network channel shared by multiple VMs, co-residing in the same physical node of the cloud system. However, this defense mechanism does not provide protection against external adversaries. Fasheng *et al* [70] in their Source Based Filtering (SBF) mechanism were motivated by a reliable feature of reappearance of legitimate IP addresses in a webserver by the traffic analysis in a small stub network and the stable nature of hop counts of each client from the server. Their assumption was that a spoofed IP address residing in the same address space of the handlers (clients from which the attack packets were generated towards the server) is very unlikely. Hence, maintaining an IP address space in the server memory for the likely IP addresses (regular clients) will ease the filtering, when the incoming packet's source address is absent in the address table. In a cloud system, with millions of packets passing among several domains with different sizes and the random nature of users, maintaining an address space is not a feasible solution. Also, a situation like a co-resident adversary is not handled in their work.

To protect from DDoS in grid and cluster environments, similar to SBF [70], a mechanism was proposed by Prasad and Kodada *et al.* [71]. In the modification of their mechanism, rather than maintaining an IP address table, both the IP address and Mac address were stored at the beginning of cluster node booting. Each cluster node will send their IP and Mac addresses to the cluster head to update the IP table, costing more memory space for storing the table. Also, their algorithm compares each packet's IP address and Mac address three times, first only the IP, then only the Mac and finally both IP and Mac addresses with the stored addresses in the cluster head.

This kind of comparison checking in a cloud system hampers the efficiency and delays the client's requests to a great extent.

Rahmani *et al.* [72] proposed a scheme for DDoS flooding attack detection, based on F-divergence. This scheme identifies a DDoS based on the number of packets and number of connections in the network. For a regular webserver, an F-divergence scheme might be a good solution for DDoS detection but for a cloud infrastructure it will not be a good solution for several reasons. This is because, firstly, a substantial change in the number of IP packets could be reasoned from the legitimate client himself or could be a bandwidth consuming application, which was newly added in the client's virtual instance. Secondly, a cloud infrastructure has to be elastic and scalable, so a sudden increment in the total number of connections could be just an upgrade in resources by the cloud providers. Finally, the F-divergence strategy did not consider the virtualization feature from the cloud service providers. F-divergence accomplished detection of DDoS attacks only in the network layer and not in the application layer. According to Prolexic's [63], nowadays, 76% of the DDoS attacks are committed in the application layer.

In [43] T. Kim, Y. Choi *et.al* proposed an abnormal behavior approach to detect traffic anomalies for mobile cloud structures to identify possible DoS attacks. They used a machine-learning tool *Weka* [44] to detect abnormal behavior of traffic patterns and used *NetMon*, for monitoring the VM instances running in the hypervisor *Xen* 4.0.0 with the operating system *CentOS 5.6*. Once abnormal behavior is detected, traffic is passed to the analyzer, which is an integral part of the *NetMon*, and an investigation is conducted with the Random Forest (RF) machine-learning algorithm [45]. This type of analyzer is based on total traffic volume (not on

specific types of packets) and CPU usage. For mobile communication, this approach could be helpful in terms of UDP packets, as UDP is the voice carrier packets in the network. But for a user browsing the net, conducting some kind of computation locally in his application hosted in the cloud, or storing some information on the cloud, requires the analysis of TCP, HTTP, and IPv4 and IPv6 packets.

In a white paper of Cisco, they claimed to invent complete protection against DDoS attacks [55]. They claimed to solve two of the most basic DDoS attacks for any data center. Those are [55]: (1) Bandwidth attack and (2) Application attack. In a bandwidth attack, the attacker tries to occupy the resources allocated for servers deployed in the network, both upstream and downstream. In an application attack the adversary ties up the expected behavior of protocols with the system resources and prevents them from processing transactions or requests. Examples are HTTP half-open and HTTP error attacks [55].

In [55] the limitations of different types of attacks are identified. Firewall [7] protections are rudimentary because they are too far downstream on the path to provide sufficient protection for the access link extending from the provider to the edge router. IDS (Intrusion Detection System) can only detect an anomaly, but not prevent one. Routers can filter but not efficiently, as most of the time packet filtering causes dropping of the legitimate user's packets once her prefix is compromised. *Black holing* is another approach where if DoS is detected then all the packets travelling from the suspicious node will be diverted towards a black hole node (unused server) located far away from the data center in order to protect the legitimate sub network. Unfortunately, legitimate packets are also filtered along with malicious traffic, and victims lose

all their traffic; hence, the attacker wins. After considering all the above cases of current counter-measures, CISCO invented a DDoS defense system consisting of 4 properties: (1) detection of the attack, (2) diversion of spoof traffic to a Cisco appliance for treatment, (3) analysis and filtering of the bad traffic flow from the good traffic flow and (4) forwarding of the good traffic [55]. Cisco Guard is deployed at the distribution layer in the data center for protection of lower speed links downstream and the servers. Also, the Cisco detector is comprised of five distinct phases to conduct an early detection of anomaly and controlling spoof traffic rate.

There are several drawbacks as suggested by Cisco. If spoof traffic is diverted, then in a dynamic distributed system such as a cloud, all the legitimate services could starve for an unknown kind of DoS due to the analysis time. Throughput will decrease and will affect the cloud provider's quality of service. The second drawback is if Cisco Guard has to be deployed in corner autonomous systems (AS), which are network border controllers, then an outbound anomaly may be handled. However, it does not consider if the attacker is an insider and using localized communication links to hinder some localized application running in local hosts. In such case, those packets will remain undetected. The third argument considers that routers are dynamic in nature and filtering in both static and dynamic methods can prevent lots of DDoS attacks and also maintain the throughput. However, a cloud provider will charge the customers an additional cost for deploying such a dynamic DDoS mitigation. Cloud customers will be forced to pay a lot of money even if their VMs are not under attack. The application owners will not be able to know if the filtering was generated due to malfunctioning from the application's end or provider's end.

Huan Li *et.al* proposed a DoS prevention mechanism for clouds by predicting a new form of DoS attack [53]. As application owners have very little or no controls over the running application in remote hosts, the attackers have lots of opportunity to flood the system. Usually, attackers target the uplink capacity as it is smaller than the aggregated capacity, and attackers send spoof packets to other hosts in a remote router. Also, hosts can learn the network topology by trace routing because of several reasons: (1) for an application running in remote routers, requested packets will occupy less bandwidth than the host router, thus, the attacker can make an assumption about the hop-count [2]; (2) an attacker can send a continuous sequence of packets from a number of sources toward a presumed sink, then analyzing the traffic rate the adversary can make an assumption about the number of switches between the source and the sink.

Several countermeasures can be adapted, such as providing full bisectional bandwidth, but it will take years to deploy this kind of a network and replace all the legacy routers from the current network [51]. Another approach could be capping each server with limited bandwidth, but for a dynamic system like a cloud, it will affect the throughput, which is not good from the business aspect of a cloud. Additional charges could be considered for bandwidth consumption, but again it is not economical, and will affect the prime goal of a cloud, which is to make it attractive to customers with less cost.

The author's suggestion in [55] was to deploy a monitoring agent. The host will send UDP help packets when an application detects a limited amount of bandwidth for a long time. In such a situation, an assumption is that some of the UDP packets will pass through the router and ask for help from the monitoring agent. The agent will search for an active standby server in other

routers to reassign the application in another sub-network. But for such a mechanism, continuous monitoring will require a large amount of message passing and cost additional bandwidth. Also, if the application is running in a remote host server, then some delay could also generate unnecessary help packet generation from the customer's end.

An award winning open source implementation is the Virtual Computing Laboratory (VCL) [46]. It provides services for wide area access to solutions based on virtualized resources. For security purposes the NC state Secure Open Systems Initiative has done the watermarking of the images and data to ensure verifiable integrity. They claim to have an excellent experience with VCL and the security solution has been holding up well. Watermarking is a good solution for telecommunication where vast numbers of instances on mobile applications are non-productive. With respect to productive instances, such as a cloud system for preventing Denial of Services, watermarking is not feasible due to the heterogeneous nature of the cloud. Also, the deployment of watermarking is very costly.

As web service (WS) technology is the most used technology in the field of Service-Oriented Architecture (SOA) in clouds, the WS-security system should be rigid enough to mitigate the security attacks from different adversaries. The message-based Simple Object Access Protocol (SOAP) provides the specifications for accessing services on the web [47]. SOAP is an XML-based messaging framework used to exchange encoded information (e.g. web service request and response) over a variety of protocols (e.g. HTTP, SMTP, MIME) [35]. As of now, the most common attack using SOAP messages is the Denial of Service attack [63], [70], and [71].

There are some prevention strategies against Denial of Service attacks. In Amazon Web Services [47] the customer has to provide a Self-Signed Certificate in the beginning. A randomly generated RSA key is sent to the AWS. Alternatively, a publicly defined certificate can be sent to the AWS with the signature. Here the AWS provides command line tools with fundamental functionality for searching the virtual machine images (AMI- Amazon Machine Images), running AMIs, monitoring the images and finally terminating some of the AMIs. These SOAP messages can be modified by the developers. The SOAP Header contains two elements [47]. One is the Binary Security Token, which contains the certificate. Second is the Time Stamp, which will contain the information about the creation and expiration of this SOAP message and which is also limited with a validation of five minutes [47]. If the SOAP message is transferred through an unsecured layer, the SOAP Body (inside the SOAP Envelope) as well as the Time Stamp (inside the SOAP Header) needs to be signed. If the transport layer is secured then only the Time Stamp needs to be signed. Since the channel is protected by means of SSL/TLS by default [54], this is largely an ineffective attack vector. Also as the EC2 Web services allow access via simple HTTP, a passive attack would be sufficient to get possession of such a request. As the public certificates are distributed among the hosts in the cloud, an inside host with ill intensions can deploy his public key as a legitimate key. He can then request a certificate from the Certificate Authority (CA). CA is responsible for issuing digital certificates. Digital certificate certifies the ownership of a public key, which allows the recipient parties to trust the embedded public key to initiate the communication between sender and receiver [30], [51]. With that certificate, the compromised host communicates with a legitimate host. SOAP messages are vulnerable to replay attacks, man in the middle attacks and masquerading attacks.

While there have been some strategies to address DoS in clouds, as described above, none of them provides a comprehensive solution for addressing DoS in clouds. The goal of this research is to propose a solution for heterogeneous cloud systems that does not increase the cost nor decrease the Quality of Service for the users. The solution will be efficient and avoid starvation. It will not result in an increase in the network packets, as only the most relevant network packets will be compared. Instead of checking the performance of the physical machine on which an application is running, we propose performance and usage measurement checking on the virtual networks, which interfaces the running VM with the physical network device installed in the client's terminal. Through analysis, FAPA will detect the malicious behavior.

In the next chapter we identify and analyze different types of DoS attacks in order to identify good solutions for cloud computing.

Chapter 3

FLOODING AND CLOUD COMPUTING

There have been many different types of DoS attacks that have been identified in general (non-cloud) environments as well as countermeasures for these attacks. In Section 3.1 we identify some of the different types of DoS attacks and in Section 3.2 we present the countermeasures to these attacks that have been proposed. In Section 3.3 we discuss the merits and disadvantages of these countermeasures in a cloud environment, and determine their potential relevance to our proposed solution.

3.1 DIFFERENT TYPES OF FLOODING

Different types of DoS attacks are discussed below and we note that all these attacks involve client requests to servers. All of these attacks can be found in general non-cloud environments. Each of the DoS types is a threat for the cloud because the cloud is a heterogeneous system where a fleet of servers are engaged in processing clients' requests. An attack targeted for any cloud server will propagate throughout the fleet and engage much of the server processing time in the cloud.

3.1.1 TCP-SYN Flooding

A TCP-SYN is a form of DoS attack in which an attacker sends a series of SYN (Synchronize) requests to the target in an attempt to consume enough resources to make the system unresponsive to legitimate traffic. Figure 2 illustrates a general TCP connection.

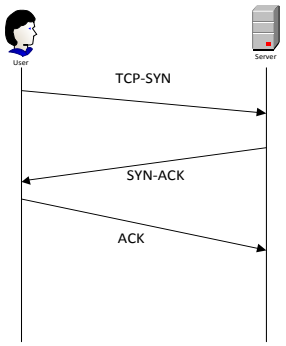


Figure 2: Normal TCP connection

In the general case, a client makes a request to start a TCP (Transmission Control Protocol) connection with the server, and then the client and the server exchange a series of messages:

1. The client sends a TCP-SYN request with a seq (sequence) number to the server.
2. The server receives the request and sends back the seq+1 and ACK to the client.
3. Next the client sends back an ACK (Acknowledgement) to let the server know that he received the packet.

Then the requested port is allocated between the client and the server, and data passing starts. Figure 3 illustrates a TCP attack scenario.

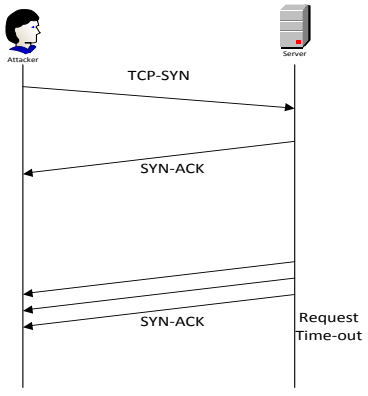


Figure 3: TCP-SYN Flooding Attack

If the source address is spoofed then the server will be waiting for the ACK reply from the requester after sending the SYN-ACK packet. The requester will not reply with an ACK packet. This stage will make a half-open connection where the server has appointed a memory location by using its finite data structure for the client and waiting for the client to acquire the space by sending an ACK to the server [6], [10]. It will not last long as all the messages passing will have some expiration time limit. In a cloud system, the servers are engaged in processing the client's requests in a continuous manner.

If such type of fake TCP requests arrive in the cloud server due to a masquerader, then handling such requests with expired time is not efficient. A lot of legitimate packets will starve and throughput will be hampered. One might think of curtailing the waiting time in the server end, but then the cloud servers might lose some legitimate traffic with longer delay. Thus, an attacker sending a bulk amount of spoof requests would engage the cloud server resources and cause denial of services in the cloud.

3.1.2 DDoS Reflector Attack

In the DDoS reflector attack, the attacker does not send traffic directly towards the victim; rather he sends packets with the victim's IP address as the source towards some intermediate servers. These intermediate servers cannot detect the originator and routes all the response packets towards the victim. So with proper coordination, attackers can cause flooding to strike the victim without even sending a single packet to the victim directly. Sometimes these intermediate servers amplify attacks; Figure 4 demonstrates the reflector attack. For example, in DNS server-based reflector attacks, attackers send short DNS lookup requests (such as 50 bytes) [10],

whereas the replies can be over a thousand bytes long, thus attacking the victim with 20 times the amplification.

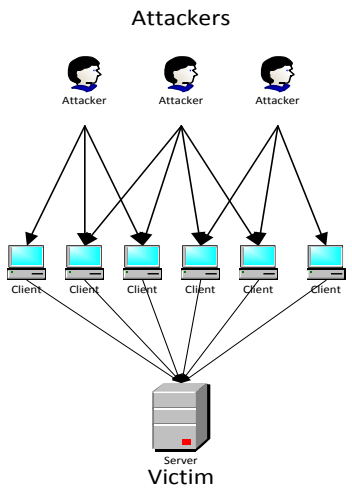


Figure 4: Reflector Attack

Internet Control Message Protocol (ICMP) echo request attacks, also called Smurf attacks, can be considered one form of reflector-based attack. In ICMP the adversary, or group of adversaries, sends echo requests to the broadcast addresses of misconfigured networks, thereby manipulating many legitimate hosts into sending echo reply packets to the target host. A hybrid cloud with the collaboration of different vendors is vulnerable to replay attacks. Before the identification of the actual originator, the intermediate cloud servers are held responsible for replay attacks.

3.1.3 TCP Hijacking Attack

A TCP hijacking attack is accomplished by injecting malicious data inside the TCP connection. The connection could also be hijacked if the IP address is known, as well as the TCP port

number, and exact guessing of the sequence numbers issued [1], [6]. Though becoming obsolete nowadays, for instance if there is a telnet session, then the attacker could run some UNIX commands which would delete all files of the current user. There are some additional similar commands which could be very disruptive, like TCP RST (closing the established TCP connection).

A cloud is vulnerable to TCP hijacking. Hijacking is usually accomplished by an inside attacker because it is not easy for outside attackers to retrieve the port numbers, unless all the ports are open in the victim's end. It could be the carelessness of a legitimate user or it could be intentional. A cloud user, not knowing the required port number of her application, could keep all the TCP ports open in the server end while running an application. If the user forgets to close the ports she opened earlier, an adversary waiting for such kind of mistake can easily strike the cloud server with a flooding attack. Also, a user with evil intention could delete some session logs intentionally by using some UNIX command. Later the cloud will not be able to retrieve the session logs.

3.1.4 Port Scanning Attack

If an attacker's machine probes a port on the victim's machine, the attacker not only sends a probe packet with its own source IP address, but also sends many fake probe packets with other (often random) source IP addresses [1], [4], [10]. Thus, even when the victim realizes that it is being probed, it cannot figure out exactly which host is probing it, thus flooding the victim.

3.1.5 UDP Flooding

A UDP (User Datagram Protocol) flooding attack can be initiated by sending a large quantity of UDP packets, to random ports on a remote host. In response to these requests, a distant host (in general a server) will check for an application listening at that port. Since no applications are listening at the random ports, the host will respond to the large quantity of packets by sending many ICMP destination unreachable packets, eventually deploying a lot of resources in response to unnecessary ICMP echo requests. The system tries to verify the spoofed IP address with its ACL (Access Control List) or in its IP look-up table, and finds nothing. At this point, the attack has already committed its intentional task of occupying the resources of the cloud system.

A cloud is vulnerable to all of these DoS attacks. A camouflaged attack propagated from a user may compromise the cloud servers. Traditional countermeasures require modification in all autonomous systems, which is not a feasible solution for the cloud system. In section 3.2 we discuss some of the prominent countermeasures with respect to network topology. The distributed network of a cloud is vulnerable to such type of denial of service attacks due to some existing disadvantages in the countermeasures, and we discuss these flaws in Section 3.3.

3.2 DIFFERENT TYPES OF COUNTERMEASURES

Defense against DoS is an ongoing concern. In [6] a direct quotation from William Cheswick asserts: "*I pulled a chapter from my book, Firewalls and Internet Security [7], at the last minute because there was no way for an administrator of the system under attack to effectively defend the system.*"

There are several strategies for the prevention of IP spoofing and some of them are as follows: (1) Ingress/Egress filtering, (2) Inter Domain Packet Filtering (*IDPF*), (3) Packet PASSPORT, (4) Spoofing Prevention Method (*SPM*), (5) Stacking Path Identification (*StackPi*), (6) Route Based Filtering (*RBF*), and (7) Hope Count Filtering (*HCF*).

3.2.1 Ingress/Egress Filtering

Ingress filtering [8] imposes some rules on the border routers of each domain to prevent against any sort of spoof packets and protects the system from flooding. Some of these rules are highlighted here.

If a packet has a source address which is not residing within the legitimately announced prefixes being used by the routers inside the network for packet passing among the hosts, it will be filtered by the administrator. Ingress does not allow UDP packets destined for system diagnostic ports from outside the administrative domain. Border routers do not allow directed broadcast packets. If an adversary has framed a broadcast address and sends it as a layer-2 packet, then the network interface card (NIC) of the router will not be able to distinguish between the framed packet and an original MAC address. Allowing this fake broadcast packet in large numbers will engage the resources of all the autonomous systems in the network and cause DoS in the system.

Another type of spoofing involves forging a legitimate source address and sending a large amount of TCP-SYN packets toward the target system. The target will be exhausted by replying with all the SYN-ACK packets and waiting for the reply, also the legitimate host which was used

as a spoof address will be flooded with an enormous amount of ACK packets coming from the target server. Thus, both the host and the target server will be the victims of flooding. Ingress prevents the system from different types of spoofed packets trying to enter the network. Egress on the other hand, provides protection against the inside hosts trying to send some spoof packets outside of the domain. Egress deploys the filtering mechanism at the border router of the stub network so outgoing packets are filtered if malicious packets are identified.

3.2.2 Inter Domain Packet Filtering (IDPF)

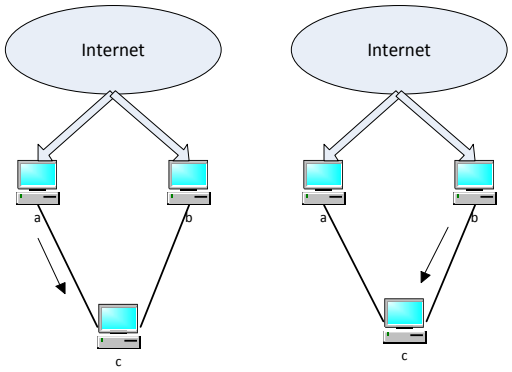
There is anecdotal evidence that modern network systems are very vulnerable to spoof attacks, and adversaries are more active than ever to commit a DoS attack due to rivalry or entertainment. An attacker can masquerade its IP address behind the prefix (legitimate IP address) of an authorized host, but to explore the original routing path is very difficult and almost impossible. Based on that advantage, Park and Lee [11] proposed the *route based filtering* scheme. The assumption was that every pair of nodes has exactly a single path to route packets between them. So a path *p(s, d)* with source *s* and destination *d*, appearing in a router that is not in *p(s, d)* should be filtered. Construction of a specific route-based filtering requires the global routing decisions of all the other nodes in the network. There are approximately 125,000 network domains or autonomous systems (AS) worldwide, each of which is a logical collection of networks with individual administrative control [4]. Due to the distinctive nature of policies imposed by different vendors of AS, it will be hard to reconcile routing decisions amongst all the logical nodes (routers). Also, each AS cannot acquire the complete knowledge of probable paths announced by all the ASes. Due to this open challenge in a *route based packet filter* [11], IDPF

was proposed to make decisions based on the message exchange between two neighboring ASes and based on local BGP (Border Gateway protocol) updates.

IDPF considers the total network to be an undirected connected graph, G(V, E). All the edges are the connections between two nodes, and all the nodes: u, v ϵ V, corresponds to an AS. If a packet u is destined for v, then e(u,v) ϵ E, where E contains all the edges of the graph G. Nodes exchange BGP route updates, which could be announcements or withdrawals. Announcements will carry the legitimate list of prefixes to allow the packet routes via these prefixes to reach the destination address. All these BGP (Border Gateway Protocol) updates are executed following two policies: (1) Import and (2) Export policies.

In import, a packet is in route from u to v then v ϵN(u), where N(u) is the set of neighbors of u. Among these neighbors the path could be one of many possibilities, but the best path will be chosen by bestR (v, d), where d is the final destination and v is the chosen intermediate node. Finally function importR (v, u) is invoked where (v, d) ϵ bestR in G.

For export, when a packet is leaving the node it will select the best route from the candidate (v, d), depending on the announcements from the source AS, and proceed towards destination d. These candidates depend on the type of nodes and the policies imposed by other vendors. All the route selections take place locally in the BGP protocol. More simply, the topological path (a loop free path between two nodes) is a feasible path in BGP as long as the packets don't violate the export policies of commercial AS's as demonstrated in Figure 5(a).



(a) Route preference (b) Alternative to Primary

Figure 5: Multi-homed AS feature

The IDPF autonomous system has the privilege to use one provider as the preferred or primary, and treating the rest as back-up providers. In Figure 5, suppose c is the destination prefix and a, b are the possible paths for packets coming from the internet [4]. So if a, the primary node is down or overloaded, then packets should route through node b as shown in the Figure 5.

3.2.3 Packet PASSPORT Filtering

Filtering at the routers can prevent DoS, but finding the actual origin is difficult. An attacker can masquerade arbitrary packet contents and hide its identity. A number of strategies were proposed for this purpose, such as ingress filtering [9], path marking scheme [1], [11], and path based source address validation [4], [12]. These methods are not economical for global deployment. Some of the methods could be deployed partially, but in a network system of 125,000 ASes, partial is not a small number. The Packet Passport method was introduced for efficiency as well as security.

In a passport packet topology, either the routers or the hosts could be compromised. So using the same BGP protocol as IDPF, the methods need to have some restrictions and maintain some principles. The border routers maintain separate tables with a reachable address prefix [9]. The principles are:

(1) The domain routers should be impenetrable so that no attacker can spoof the source when the packet is launching.
(2) All the domains will verify the passport packets without the involvement of other domains.
(3) Valid passports cannot be replayed. This is because if an attacker gets a hold of a valid passport then he can try to flood the system by making a lot of copies of the same passport and sending them towards the domain to commit a replay attack [9].

Thus, the system prevents replay attacks as well as denial of service attacks [9].

The strategy is as follows. The source will sign the packets with a digital signature and the intermediate routers will validate the packet by using the source's public key. Because the signature convention is very expensive to deploy, the passport system uses the light weight MAC (Message Authentication Control), like HMAC or UMAC [9], making it more economical. All the passport enabled routers have a list of keys. A packet arrives in one of the domains, places a request for passport checking, and then validation is done by cross checking the key. If the key is found then the packet is legitimate, or else it will be filtered. The key is stored in the shim layer or the option header of the IP packet [9]. Domains that request passport checking will examine the IP header of the arrived packet since the key is stored in the 16-bit IP header of the IP packet. In Figure 6, the assumption is all the domains in the network are passport enabled.

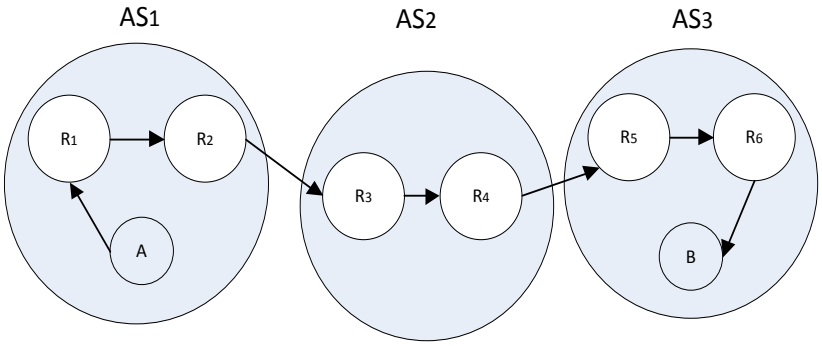


Figure 6: Packet Passport Routing

The secret key shared between AS_i and AS_j is $K(AS_i, AS_j)$. Suppose host A sends a packet to host B. The border router R_2 stamps the outgoing passport which has three MACs. Each MAC is computed using key $K(AS_1, AS_j)$, j=2, 3. When the packet arrives at AS_2, the border router R_3 uses the secret key $K(AS_1, AS_2)$ to verify the packet is coming from AS_1. Similarly, router R_5 will use the key $K (AS_2, AS_3)$ to verify the origin before the packet reaches the destination host B.

This method increases scalability but compromises security. For example, a rogue or compromised host can fake the host identities. If the routers distribute the resources inside the domains based on the number of IP addresses queued up for bandwidth, no one will actually fake their own identity. But as the passport packet enabled routers manage the number of packets based on a predefined threshold, an attack could exceed the threshold limit and legitimate packets can also be filtered along with the malicious packets.

3.2.4 StackPi

In StackPi all the outgoing packets are marked, and this marking holds the contribution of all the routers. The StackPi key is stored in the identification field of the IP which is 16-bits long [1], [10] and allows the key to be 16-bits long. So the possible number of routing paths cannot be more than 2^{16}. However, since some paths could be longer than that, precautions need to be considered by the user who will implement the StackPi approach for building his network topology [1], [10].

After the receipt of the packet, the ID field is shifted left *n* times and XORed with a pre-calculated n-bits in the least significant bits, which were freed after the shifting of the ID field. The shifting will take place in all the routers until the packet reaches the destination and every router will append their marking in the LSB bits [1]. Before reaching the destination, the ID field could be shifted more than 16 times and the last router might not even be able to mark it completely, which makes the marking incomplete. So the additional marking will camouflage the odd bits [10]. The marking is actually the hash of the router's IP address which should be n-bits in length. Problems will arise if two similar routers try to do marking before reaching the same intermediate router. The target router will be confused by the marking and will not be able to distinguish between the last hop routers. To avoid such kind of collision the concept of the last hop IP address is implemented. StackPi marking will not only contain the marking of the receiver router but also keep the mark of the last hop router or the sending router [10]. Thus, StackPi avoids collision. In Figure 7, every router is marked with its ID and in the next router the last marking is checked before forwarding the packet.

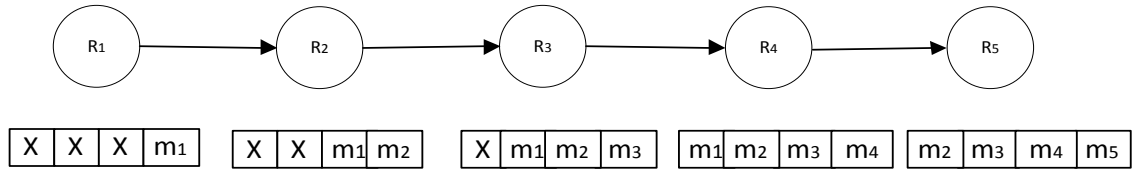


Figure 7: Pushing last hop in every router

If there are some legacy routers, which do not allow this kind of marking in the IP packet headers, then another precaution was considered, though it added some overhead cost in the StackPi marking. This is known as the next hop router information. All the routers will have two types of marking, incoming and outgoing. If the StackPi router forwards packets toward another StackPi router then it will push the marking for the next hop in the ID field so that the next router checks the incoming packet with its tag in it. In Figure 8 the router R3 had no operation for pushing the marker for the next hop, because the next hop is not a StackPi router. As a result, router R5 will check the stack and understand the incoming packet was coming from a legacy router which is not StackPi enabled. In this way all the StackPi routers leave their marks in the IP header and detection of the compromised packets by the routers is feasible. If a packet is compromised, the adversary cannot change the marking of the routers and the attacker can be traced back easily. Also, if non-marked packets are flooding the system then it can be assumed that the attack was generated from a legacy router.

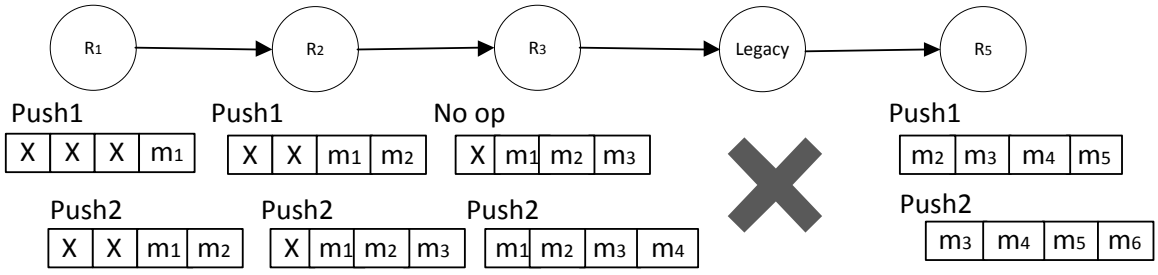


Figure 8: Write-ahead implementation

Mathematically, the ratio between general packets and marked packets is analyzed in this way: $(u_i/a_i+u_i) < t_i$ where the t_i= number of packets allowed with StackPi *i*; u_i= number of packets from user with mark *i* and a_i = number of attack packets with mark *i* [10].

3.2.5 Spoofing Prevention Method (SPM)

The Spoofing Prevention Method prevents against spoofing with the help of a key. SPM deploys marking on all the outgoing packets and authenticating all the incoming packets by checking the embedded key inside the identification field of the IP headers [3]. Authentication and verification require an IP lookup each time. Major responsibilities lie within the Autonomous Systems (AS). All the keys are placed by the AS in the source, whereas the authenticity is checked by the destination AS [3]. The key is placed in the ID field which is only 16 bits long and limits the length of the key by 16-bits. But because there are two keys maintained by every AS pair, the total length of a key pair is 32-bits [3]. A packet starting at a node will be appended with a symmetric key by the source AS. Before entering the destination AS, nodes will check the key and compare it with the source key. If a match is found then the packet will be considered as legitimate or else it will be filtered.

An AS has several tasks [3]:

- Choosing the key from the AS-OUT table.
- Distributing the key towards all other routers inside the AS.
- Announcing the keys from AS-OUT to all other AS servers in SPM.
- Making the AS-IN table from their announcements.
- Updating the AS-IN table to its routers.

Keys are chosen randomly [3], but every key needs to be replaced after a while so that attackers cannot make any timing attacks. At the time of key replacement, the router holds both the new and the old keys. A newly arrived packet is authenticated based on either of the keys (old/new), then after authentication the old one is removed [3]. Another approach for key replacement can be using a PRNG (Pseudo Random Number Generator) in each AS source [13]. SPM routers can tag the packets with keys or this process could be dynamic. Three types of traffic will be generated:

1) SPM certified spoofed traffic,
2) SPM certified non-spoofed traffic and
3) All other traffic. Spoofed traffic is filtered here [3].

Even after all this precaution, the SPM enabled system is not completely secured. Most of the routers in the network needed to be SPM enabled according to topology. If a host under a non-SPM AS sends a large number of packets to a host under SPM topology with randomly generated keys, then all the traffic packets except one will be filtered. The one not passed will hold the legitimate key. This key can be used by a compromised host and copied to every other host in the domain to send a packet to a target host within the domain, as well as the neighbor domain. Hence, the key owner will be the victim for this attack.

3.2.6 Route Based Filtering (RBF)

Unlike the previous approaches discussed, Route Based Filtering (RBF) does not apply any kind of marking strategy on the outgoing packets. It does not implement the authentication strategy on the incoming packets. The RBF mechanism prevents against several DoS attacks, such as TCP

SYN, ICMP flooding. Each router maintains a routing table based on the packet passing while forwarding non-spoofed packets [11].

After a packet arrives from a neighbor router, it checks the packet's source address and destination address. If the packet claims the IP address of an infeasible remote router as the source, then a router filters that packet because the assumption is all packets route through the shortest paths. An attack generated from a nearby host, masquerading its IP address behind the prefix of a distant host, will be caught in the border router between the source and the destination address because of the suspicious distance [11]. Considering Figure 9 below, assume a host in node 7 is trying to attack a host in node 4 and carries the IP address of a host from node 2.

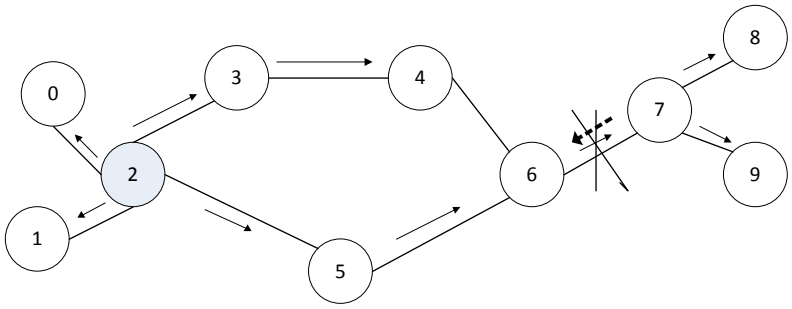


Figure 9: Route-Based Filtering

While routing the packet towards node 4, the peer of node 7 which is node 6, will not allow the packet to pass through node 6. This is because a packet with source address of node 2 is not supposed to come from node 7 when there is a more feasible routing path in the network. So the packet is assumed as a spoofed packet and filtered eventually in node 6.

According to network topology, all the possible routing paths are available inside the routing table of each node. Now let us assume the attacker is in node 7 and hiding its identity behind the IP address of node 8, and sending packets to the same host in node 4. Then this packet will not be filtered in node 6, because packets could route through node 6 coming from node 8 and destined for a host in node 4. Node 6 thinks the sender is node 8, when the actual sender is node 7. Hence, these types of packets are allowed and are able to flood the system in node 4.

3.2.7 Hop Counting Filtering (HCF)

An attacker might obtain the IP address of the source but cannot falsify the number of hops that a packet will take while routing towards its destination address. This vital information can be used to prevent spoofing [2]. If a mapping could be done between the hop counts and their IP addresses, then the routers are able to do victim filtering on their own. The hop counts need to be shared among all the routers along the routing path. To do so, the information can be embedded as TTL inside the IP header as infrastructure information, which would be retrieved by each autonomous system when a packet arrives [2].

In HCF the authentication of packets takes place with the number of pre-calculated hops for each source IP address. An outgoing packet is embedded with a hop count number in the TTL field. TTL actually is a field of 8-bits in length and signifies the life of a packet as in *Time To Live*. As a packet leaves a router, its TTL value is decremented by 1. When it reaches the final address (destination), the TTL value will be the remaining value after subtracting the total number of

routers from the initial TTL. A lookup from the mapping table will clarify the number of hops a packet has traversed before reaching the destination.

The challenge is that the TTL value varies based on the type of operating system [2]. It could vary from 30, 32, 60, 64 to 128 or 255. When the remainder is calculated, the hop count is assumed based on the closest value as in the OS. As an example, if the value is 120 then its initial value is more likely to be 128. Subtracting from the initial TTL, the remainder is the number of hops the packet should visit.

3.3 MERITS AND DEMERITS OF COUNTERMEASURES

In this section we will discuss the advantages and drawbacks of some DoS prevention strategies mentioned in Section 3.2 and analyze why these strategies are feasible or not feasible solutions in a cloud environment.

3.3.1 Inter-Domain Packet Filtering

Most of the filtering processes using the border gateway protocol were faced with the obstacle of path selection locally only within the domains. There was a belief that filters based on feasible routes are less efficient than filters based on best routes. The IDPF strategy overcame this obstacle by embedding the information implicitly in all the BGP updates.

Here are the advantages for which the IDPF is more efficient in preventing spoofing and DDoS attacks than the RBF [4]:

1. In IDPF the feasible paths are restricted to a small set of paths between the source and destination so that filtering becomes more preventive and efficient against IP spoofing.
2. An AS always has alternative providers. If the primary provider is unavailable due to traffic load or service down time, then an AS announces the alternative as the primary path to all the incoming traffic to change their route to the alternative path towards the AS [4]. After a while, if the original primary path is fixed, then AS again announces the change of path to the original one for all the incoming packets. This mechanism makes IDPF more efficient in maintaining the throughput [4], [6], [13].
3. All IDPF enabled ASes are capable of protecting each other in terms of sending traffic. In the general case the customer ASes are always safe from the IDPF enabled AS server.
4. In the worst case scenario, if there is only one IDPF enabled AS in the network, it will take a long time to generate a DDoS attack based on the IDPF perimeter targeting the individual hosts and services. This is because of the encryption used inside the AS by IDPF.
5. With vertex covered IDPF ASes, it is statistically proven that attackers are unable to launch a successful attack in the internet in more than 80% of the ASes in the network [4].
6. IDPF also made the trace route more efficient: a compromised AS can localize the origin of the attack within 28 ASes [4].
7. Usually IDPF is implemented in collaboration with ingress filtering, but it is proven that even without ingress filtering, attackers cannot launch within more than 60% of the ASes [4]. Also, the origin of the attack can be localized within 87 ASes.
8. The number of impenetrable ASes has risen from 5% to 11% [4].

Beside these advantages, IDPF also possesses some drawbacks and pitfalls:

1. As IDPF allows only the upstream neighbors for sending M(s, d) packets towards the destination, an IDPF AS may not be able to catch all the spoofing traffic forwarded by the neighbors [4]. *In a cloud, the servers have to protect against inside attackers too, but IDPF cannot provide protection from neighbors. This is because IDPF is focused on prevention based on the border routers rather than domain routers.*
2. As mentioned earlier, two types of filtering are used for prevention against spoofing [4]: (1) Proactive and (2) Reactive. Proactive means providing the security before passing the spoof packet through routers and reactive is a cure after the DoS is detected. *In a cloud, proactive protections are almost always infeasible due to the random nature of on demand service requests by the cloud users. Instead, a solution should work in reactive manner; filtering starts after spoofing is detected.*

3.3.2 Passport Packet Filtering

In passport topology, hosts only send packets without any signature. The border routers of that domain receive the packets and stamp an identification field in the IP header, known as the passport [9]. Then in all the transit routers the packet is checked and matched with the available key and stamped to verify the source of the packets. If there is a match then the packet is allowed to pass or else it will be dropped. As the key is known only by the source and transit routers, even if the border routers are compromised, they cannot forge the passport of other domains [9]. Hence, a spoofing attack outside the domain is checked.

Key distribution internally has some advantages:

1. No internal host can inject any malicious code as the private keys are stored inside the domain server and can be easily traced back to the attacker host.
2. Adjacent routers inside a domain do not need to have a passport; rather they can exchange packets via the public keys.
3. Also a DoS attack within the domain is checked.
4. A Passport system uses rapid keying without excessive distribution of messages.

The benefits of Passport Packet filtering have incompatibility issues in a cloud environment:

1. *In a cloud, a heterogeneous system may contain a lot of legacy routers which are not Passport enabled. If the spoofing is generated from those routers, then detection will take a long time, and require the involvement of a neutral third party to end disputes. Also, the internal key distribution for each customer will require more computation time.*
2. *A Passport system is cryptographically not forgeable. Each packet can be validated independently without the help of the rest of the network. But involvement of cryptography is very costly for a distributed architecture, such as a cloud, due to the implementation of encryption and decryption in every domain. Also, detection of intrusion requires cryptanalysis of network packets in border routers, which will add more delay*.

3.3.3 Spoofing Prevention Method

In SPM, the edge routers are less overloaded than the domain routers, so packets are authenticated in the edge routers. To maintain the tagging of network-in and network-out tables, a regular IP-lookup table is required to verify the IP addresses of the source destination pairs [3]. Besides maintaining the regular IP table SPM possesses some beneficial factors.

Advantages of SPM are:

1. As the cost of tagging is less expensive than an authentication process, SPM uses tagging because tagging was done with a piggy-backing [39] system.
2. There is no need for extra message passing for authentication, as only the ID needs to be stamped inside the IP header of the packet. On the other hand, authentication also needs additional look-ups [3].

The advantages of SPM are incompatible for a cloud environment to some extent. Those issues have been highlighted below:

1. While damage is reduced under the ingress/egress club defense, the hypothesis is that benefits are proportional to the number of participants [3]. *With a small number of SPM enabled ASes, the benefits won't be large enough to draw the customer's attention. In terms of cloud customers, marketing is one of the most vital issues for business perspective.*
2. In an asymmetric system where the sizes of the domains are not identical in terms of the number of routers and number of packets, the benefit is greater. *In private clouds, most of the domains are identical; hence, SPM will not be a suitable option.*
3. If SPM and ingress/egress are combined and used in an asymmetric domain, then the benefits are more exponential than SPM only [3]. *Tagging the network in and out tables with network prefixes via piggy backing will not be a cost effective solution in clouds. Also, implementation of hybrid DoS prevention techniques (SPM + Ingress filtering [13]) in a heterogeneous system is not a good option due to the large number of owners and providers.*

3.3.4 StackPi Packet Filtering

In StackPi there are two phases for hosts [10]: (1) The Learning Phase and (2) The Attack Phase. In the learning phase the victim will learn the differences between compromised packets and non-compromised packets [10]. Here the assumption is the system is secured and no attack has been detected. But in the attack phase, the victim is no longer capable of distinguishing between legitimate packets and compromised packets [10], [37], [43]. Thus, the host will rely on StackPi filtering to accept or drop decisions.

Several advantages and possible disadvantage in cloud environments are mentioned below for using StackPi:

1. With StackPi 50% of the servers' capacity will be engaged in serving legitimate users, compared to deploying a random strategy where 0.6% of the server's capacity serves legitimate users [10]. *Utilizing half of the server's capacity for legitimate users is not an attractive option due to the high throughput tendency in cloud systems. Also, significant improvement over random filtering is not an option for clouds due to SLA.*
2. When an adversary attacks the system, they consider the StackPi enabled routers rather than the legacy routers. StackPi marking of the source address is obtained by the attacker before launching the attack. According to StackPi, the probability of an attacker spoofing an address with the exact StackPi marking of the source address is $1/2^{16}$, which is very low [10]. *This is a good achievement from StackPi and could be an attractive option for cloud SLAs due to its low probability of spoofing, if it can be implemented.*
3. Even if StackPi marking is compromised, the low order bits still hold the information about the previously routed legitimate paths [10]. A packet can follow the embedded path that was

contributed by legitimate hosts. *This could be an intelligent and dynamic feature for a cloud, if the compromised servers could retrieve the route from the spoof packets for trace back.*

All the packets generated from a network usually have the same StackPi mark and are usually routed along same path [10]. A careful attacker can easily generate some timing attack based on close observation and predict the markings to retrieve the routing path. Implementation of StackPi features in a cloud will be very challenging.

3.3.5 Route Based Filtering

RBF [11] is a productive approach for filtering but as it considers destination reachability, not source reachability, attacks are mostly detected at the destination. Basically in RBF the packets are verified and validated based on the targeted destination [11] and keys are created based accordingly. Prior to an attack, precautions or preventions against DDoS are not easy.

RBF has some benefits but is not that beneficial for DoS protection in cloud environments. The reasons are explained below:

1. With statistical analysis it is proven that only 12% of all the AS systems can be used by attackers to launch a DDoS attack [11] [8]. *But in a huge distributed system such as a cloud, which consists of thousands of network domains and hybrid routers, 12% protection is not satisfactory. In addition, only destination-based spoofing is protected, not a botnet attack, replay attack or eavesdropping*.
2. Compared to probabilistic packet marking, route-based filtering is proactive, because a single spoofed IP packet suffices to reveal the attacker's AS location within a confined number of

sites [11]. *Predetermined packet filtering could jeopardize the scalability feature of a cloud. Bandwidth hungry application packets might be filtered in routers without checking the legitimacy.*

3. RBF with dynamic packet filtering can curtail spoof IP flows from reaching the destination, even with a few number of AS sites [8], [6], [11]. *But implementation of the hybrid approach (RBF and DPF) in clouds is not cost effective*.

3.3.6 Hop Count Filtering

For HCF [2] it is necessary to examine hop-count distributions at various locations to ensure hop-counts are not concentrated around a single value. If 90% of clients' IP addresses are 10 hops away from each other, then it is difficult to distinguish between spoof and non-spoof packets using HCF alone [2]. Statistically, with a Gaussian distribution based on a hop count, the wider the girth, the more effective HCF will be [6], [18], [22].

HCF has some benefits:

1. Attackers may proceed with already compromised IP addresses but the disadvantages for attackers are:
    a. List of spoofed addresses are greatly reduced, so tracing will be easy.
    b. Attackers have to modify the attacking tools every time.
2. HCF can identify 90% of the single source spoof address [2].
3. During the addition of IP2HC entries, attackers attempt to alter HCF values. This issue can be prevented by using TCP topology knowledge [2], [8]. The host with the spoof address is not supposed to receive the SYN/ACK packet for TCP establishment.

4. HCF is preventive against bandwidth attack [2].

The disadvantages in a cloud of using HCF are:

1. If attackers can learn HCF then their packets can dodge hop-count inspections. Though they will need more time and resources [2], [8] and [19], an attacker needs to build an IP2HC mapping table in the host and use trace route to count hops [2]. *In cloud systems, maintaining an IP2HC mapping table in each of the routers in the network could be a burden to prevent denial of service attacks.*
2. TTL values vary for different OS types as mentioned in earlier section. Thus, they require implementation of flexible TTL values for a cloud environment. *This kind of implementation is very difficult due to variation in cloud users' applications dependency on OS. As an example, a user's VM is now running on Windows but it could run on Linux or the MAC later for other applications. This flexible implementation can be very complicated.*

All the edge networks deployed either in an isolated or collaborative manner cannot offer complete protection against IP spoofing or DDoS. Though the performance of Passport, HCF and the StackPi are significant, they prevent against peer-to-peer spoof traffic and do not prevent against DoS or DDoS, which is the most common security issue in distributed systems. The cloud is vulnerable to DoS or DDoS attacks, which can hinder continuous on-demand services of cloud.

We propose a unique approach to prevent DoS or DDoS from the user's perspective. Our solution will work in a reactive manner to different types of DoS. Filtering of malicious packets

occurs after spoofing is detected and will be based on the bit content and behavior analysis of the packets. Some of the effective ideas of HCF are included in our filtering module, such as hop counts. Our approach will protect against insider attackers to provide protection from neighbors. Customers will have control and providers will not be active participants in our model. This will avoid the time cost of contacting third parties. Also, our approach is scalable, more economical and less complicated in terms of initial setup and monitoring.

In the next chapter we describe our proposed model. In a later section we will illustrate the implementation of our prototype from a stub network deployed in our lab, followed by the performance results and an analysis.

Chapter 4

FLOODING ATTACK PROTECTION ARCHITECTURE

4.1 MODEL DETAILS

In modern cloud systems most of the security is provided through a CA (Certificate Authority). Separate certificates are maintained, along with digital signatures to authenticate each session running in the cloud with VMs (Virtual Machines). The authentication process is very expensive and does not prevent against inside offenders, in other words, a trusted user. Also, certificates need to be forwarded by the provider each time a session starts, increasing the message overhead in the system. Vendor specific corporate policies among the ASes (Autonomous Systems) hinder the countermeasures due to a variety of routers in a cloud network. Total agreement on every AS from all the different vendors is very hard to achieve.

To overcome such security threats we propose a model, called FAPA (Flooding Attack Protection Architecture), containing different components that collaborate to protect a cloud system from DoS attacks. FAPA was designed by considering machine learning flows and considering the lower level networking needs required to analyze individual packets. FAPA is user centric and deployed in the user's terminal or virtual machine; it is not proactive but reactive. To be more specific, FAPA is deployed inside each of the VMs hosted by the Node Controllers of the cloud system. It captures raw packets and through analysis detects the behavioral pattern of traffic packets with respect to individual users in each session. These patterns invoke the server to propagate an announcement to the user and filter the packets if an

intrusion is detected. We note that packets are filtered if a DoS attack is detected, rather than deploying a third party for traffic analysis in case of a dispute. Deployment of FAPA within the user's running VM keeps a healthy transparency between the cloud provider and clients in terms of reliability of the user's hosted application.

In FAPA, separate modules have individual functionalities. As shown in Figure 10a, the modules in FAPA are as follows: (1) Traffic Unpacking, (2) Feature Selection, (3) Comparison Checking, (4) Validity Checking, (5) Profile Generator and (6) Hypervisor. We now provide a general overview of how FAPA works.

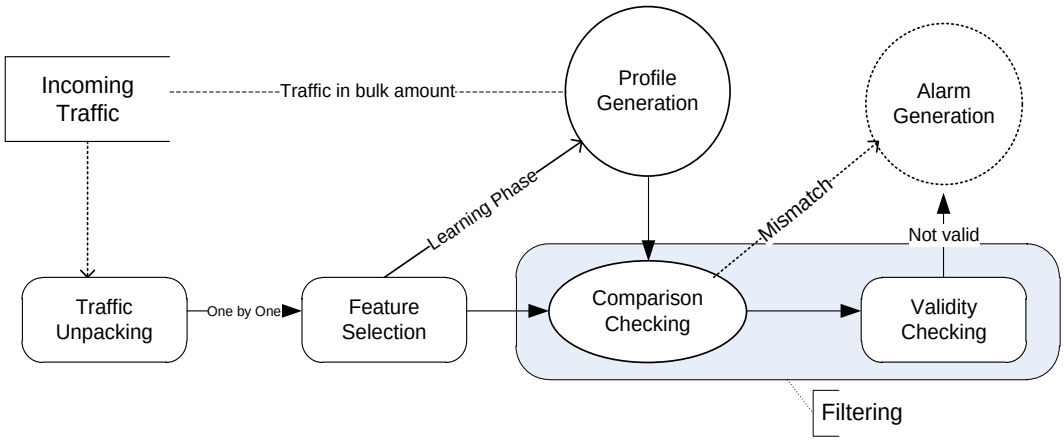


Figure 10a: FAPA Model

FAPA begins by fetching each incoming network packet from a domain in a packet-by-packet manner. A packet could be TCP, UDP, IPv4 or IPv6. The packet is unwrapped in the Traffic Unpacking module. In the Feature Selection module, pertinent header information such as, header length, window scale, payload size, source and destination ports, source and destination IP address, and some flag bits are recorded and used for profile generation. These profiles

identify the uniqueness of every running instance in the cloud. If a DoS or spoofing takes place, then some of these unique features change.

FAPA also utilizes a second capturing style, in which the total throughput of certain packet types is recorded. (A handler is defined to select packets either one by one or in a loop fashion for a bulk amount.) The Comparison Checking module monitors the throughput pattern of each packet type and compares it with the previously recorded packet parameters. Any change in traffic behavior or altering of certain bits in the packets will generate an alarm in the Alarm Generation module. A suspicious node's IP address will be traced back and incoming packets from that node will be filtered by the Validity Checking module. The filtering process consists of calculating the spoof traffic, miscellaneous traffic and original traffic. If a legitimate user is compromised then an actual attacker will be traced back and packets coming from the actuator (attacker) will be filtered. Also the alarm is propagated to alert the rest of the nodes in the cloud to be aware of the attacker and the spoof packets. Regular services in cloud will not be hindered.

FAPA reduces the involvement of a third party in case of a dispute and reduces various costs related to the setup, maintenance, upgrades, and monitoring, etc. FAPA will run in the cloud user's end and users will be enabled with continuous traffic monitoring. A brief discussion about the function and implementation of each module is provided in the following section.

4.2 FAPA PROTOTYPE IMPLEMENTATION

In this section we discuss in detail how each module works and its significance in preventing flooding.

4.2.1 Traffic Unpacking

A cloud network can consist of several network devices. In the Traffic Unpacking module the device list is pre-calculated before starting packet capturing. As mentioned earlier, users have privileges to monitor the current traffic situation locally, so users are enabled with device selection to check on the traffic flow.

Once the network device is chosen, different types of packets are captured and dumped in different library files, such as *libpcap* for raw packet capturing in Linux systems or *winpcap* for a Windows system. TCP-SYN, IPv4, UDP are the major types of packets captured by the module. TCP and IP packets are the most useful types containing the source port, destination port, address of the originator and specific kinds of flag bits, such as SYN, ACK, FIN, ECE etc. These flag bits, as well as the size of the header and payloads of each packet, contain important information. The intention is to generate the behavioral pattern during the occurrence of denial of service or flooding attacks. As an example, if the FIN bit of a TCP packet is set, then the sender will send no more data because that was the last packet in that session. If the SYN bit is set, it is the first request packet for a TCP connection. Also for UDP, HTTP header bits and fields assist in confirming the content of the header with Boolean values. The UDP source and destination ports, length fields, and checksum are major fields for traffic analysis.

Packets are captured in two ways:

1. Packet by packet
2. Multiple packets in each capturing

A handler is defined to select packets either one by one or in a loop fashion for a bulk amount. For a detailed analysis of a certain type of packet, we proceed with the first method, and for the analysis of the packet flow through the working VMs, the second approach is more convenient.

4.2.2 Feature Selection

This module is responsible for fetching packet specific features for behavior analysis. Pertinent header information, such as, header length, window scale, payload size, source and destination ports, source and destination IP addresses, and some flag bits are unwrapped, recorded and used for profile generation after packets are captured. Here the retrieved packet's information is analyzed based on the session time of each user. An application running in the cloud with extensive power of behavior recognition will require no additional devices to guard the system from DoS threats.

4.2.3 Comparison checking

The Comparison Checking module will verify the packet types, their sizes and all infrastructure details with previously recorded information. This module preserves the access rights, encryption, and date/time information for all the users based on every session in a working day. Checking is done before allowing the network packets to be sent to the server. The request is then forwarded to the Validity Checking module.

4.2.4 Profile Generator

After comparison checking, all the individual traffic patterns are stored in the Profile Generator and are sorted by different packet types. In FAPA, all the nodes running a task, such as SQL queries, have a certain amount of buffer size, payload, header length, and window scale. These measurements are kept in the profile generator for a certain period of time before they are refreshed for new types. Within the specified packet interval any packet or multiple packets arriving from the Feature Selection module will be compared in the Comparison checking module with the help of the Profile generator.

4.2.5 Validity checking

For validation we do not use a complex authentication protocol or encryption method. Neither do we use any kind of CA agent for generating encrypted certificates for each session. Rather we use the behaviors detected in an earlier module. If any change in pattern is detected, then packets arriving from those nodes are filtered. This filtering is committed based on the type and amount of different packets.

4.2.6 Hypervisor

Instead of filtering packets within each virtual machine, an alternative approach is to utilize some features from the FAPA model within a hypervisor level. A hypervisor allows multiple operating systems to run concurrently on a cloud, which is a major feature for a heterogeneous system. A hypervisor exists at the lowest level of the hardware. Since the hypervisor will be in the kernel of the cloud system, it will be difficult for adversaries to intrude into the hypervisor [6], [14], and [81]. There are many open source cloud frameworks, such as Eucalyptus, Hadoop, Open-nebula, and VMware's Cloud Foundry. Eucalyptus has the hypervisor KVM and VMware has the ESX

hypervisor. Currently, the only task a hypervisor performs in a typical cloud is the scheduling of resources based on the requests from the customers. Some additional tasks could be appointed to the hypervisor to prevent DoS/DDoS.

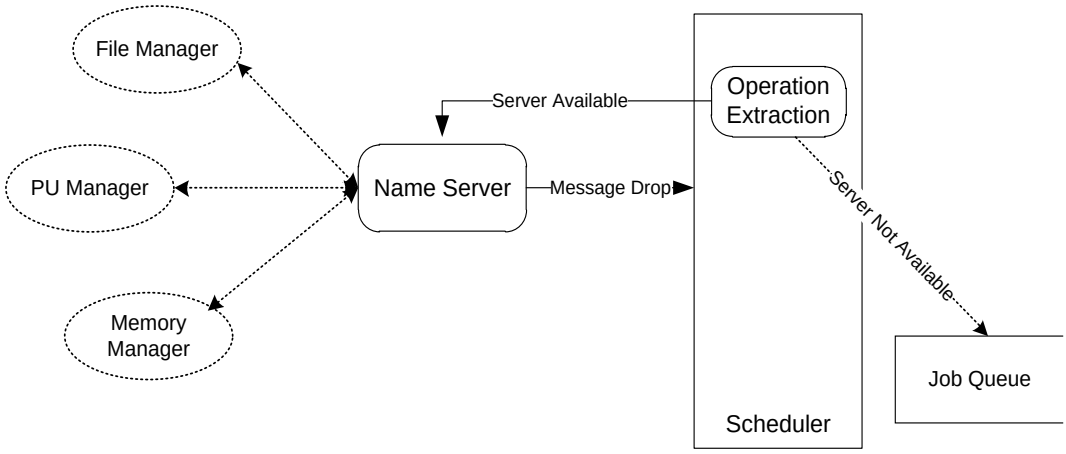


Figure 10b: Hypervisor

We consider different perspectives of involving a hypervisors in a cloud. First, the traffic pattern in a cloud is not unique for all of its users. In other words, clients might have different patterns of usage or different times of day when they access the cloud. Therefore, there should be a module that has the self-learning capability to recognize a valid traffic pattern in order to authenticate all the users. Second, resource allocation should be flexible for each user during a given time span. For example, a user may need more CPU time or RAM during the day, while the same user needs less CPU or RAM at night time depending on the running OS and applications. In this regard, the system also has to be scalable and dynamic for a cloud. Lastly and importantly, scheduling should utilize a separate module which establishes the initial connection between a user and the cloud system through an intermediary system. This is needed

because direct communication could be misused due to the lack of knowledge of a new user or a trivial mistake. The hypervisor should undertake all of the following three responsibilities:

1. Self-Learning;
2. Scalable and dynamic,
3. Scheduling the resources based on the customer's requests.

For the first characteristic of self-learning, there should be information about an application running in the system and the number of requests on a daily basis from a legitimate customer for that specific time period of the day. If there is any discrepancy in the regular service pattern or if bulk amounts of requests start to arrive for the system to process, an alarm could be generated to the user. This kind of property requires a machine learning strategy implementation in the hypervisor.

For the second characteristic, the hypervisor should be scalable and highly dynamic. A cloud is deployed with a fleet of servers. If any server is close to running out of resources or any DoS is detected, then all the running instances of the server could be reassigned to another server with a sufficient amount of resources required by the transferring instances.

As specified by the third characteristic, the hypervisor commits the scheduling task between different servers for balancing the resources among users. Similar to the scheduler in an operating system, which resides in the kernel of the OS, the hypervisor schedules tasks with a lookup table where user specific resource requirements are mapped. This property could be utilized in terms of DoS prevention.

A Job Queue is maintained in each hypervisor, so we can utilize that queue to keep the suspected packets rather than filtering directly based on source prefixes. This is because in between spoof packets, some legitimate packets could also be trying to access the resource. The scheduler will then proceed with the non-spoofed packets. After detection of the actual originator (genuine client) the jobs from the queue will be processed rather than dropped. Accountability is maintained by not transferring the control to the cloud providers. Rather the cloud customers can monitor traffic patterns of their running instances.

The FAPA model provides a framework for DoS detection that is applicable on different system component, from individual virtual machines, to the hypervisor. In this dissertation we will focus on the FAPA model for individual virtual machines deployed in a private cloud. We have implemented all the different modules of FAPA, the Traffic Unpacking, Feature Selection, Comparison Checking, Profile Generator and Validity Checking modules. We have performed experiments to obtain data to discover the behavioral patterns with and without flooding in virtual machines. The results from our experiments are described in the next chapter.

Chapter 5

EXPERMENTAL RESULTS

In our study, we performed experiments on two different types of platforms, a cluster and a cloud. Initially, we wanted to explore the impact of flooding on typical traffic patterns. We used a cluster to study the traffic behavior under DoS attacks, because the features of a cloud, such as hosting virtual instances, were not needed to obtain results from this experiment. However, in order to compare the impact of DoS on sibling and neighbor VM instances, we ran the experiments on a private cloud. Results from both these types of experiments are incorporated into the FAPA filtering strategy and we subsequently tested the effectiveness of the filtering strategy on our private cloud.

5.1 ENVIRONMENTAL SETUP

Experiments are conducted on an Ubuntu12.04 platform with the enterprise server edition. The server has 10 nodes, all with an Ubuntu11.10 client. The server was connected through CISCO Linksys E900. All the nodes were connected to the server via Linksys Ethernet gigabit switches, LKS-SG5P, in a star configuration. Each node is composed of an Intel D201GLY2 mainboard with a 1.2GHz Celeron CPU, 1Gb 533 MHz RAM and 80Gb SATA 3 hard drive. Shared memory is implemented through NFS mounting of the head node's home directory. MySQL Server (version 5.1.54) was installed on each node. The coding was done in java; jdk-1.7.0 and jre-1.7.0 were installed in the server as well as the nodes. We used *JNetPcap* external jar files

which are not 64-bit compatible (enterprise server 64-bit). The 32-bit version of *JavaFX 2.1.0 SDK* (64-bit) with *JavaSE Development* was used, which comes with *JavaFX 2.2 SDK*. Also, to implement flooding, we used the *gcc* compiler and python v-2.0 compiler *scapy*.

At the beginning of the experiment, network devices were listed. To show the list of devices in a console, we used a java program in *Eclipse Juno 32-bit* in a Windows operating system. This Windows terminal was connected with the server on a temporary basis via the CISCO Linksys E900. We used the *Windows 7* operating system with a 3.16 GHz Intel(R) Core(TM)2 Duo E8500 machine, and installed memory was 4GB. Figure 11 illustrates the network devices as displayed with the Eclipse Juno console:

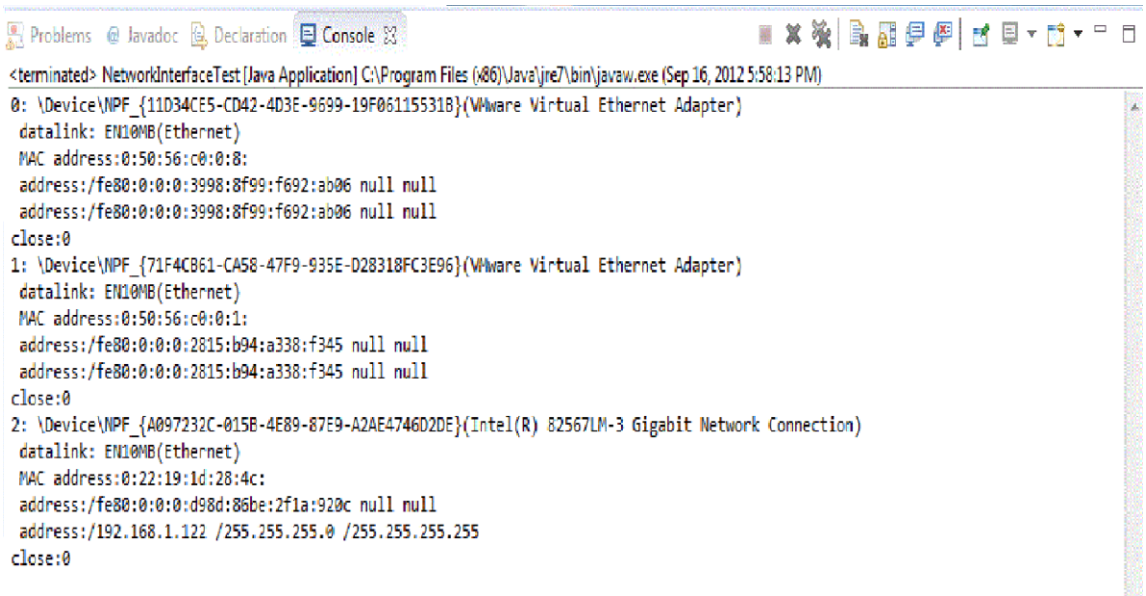


Figure 11: List of Network Devices

To get the device information, we imported the *Jpcap* external jar files. The Jpcap library doesn't work stand alone, so we had to install the *Winpcap* library and store the files in a System32

directory and put the *jpcap* executable file in a java library directory. The connection setup is provided in Figure 12. For our experiments, we assume an attacker has occupied the IP prefix of a node inside the cluster.

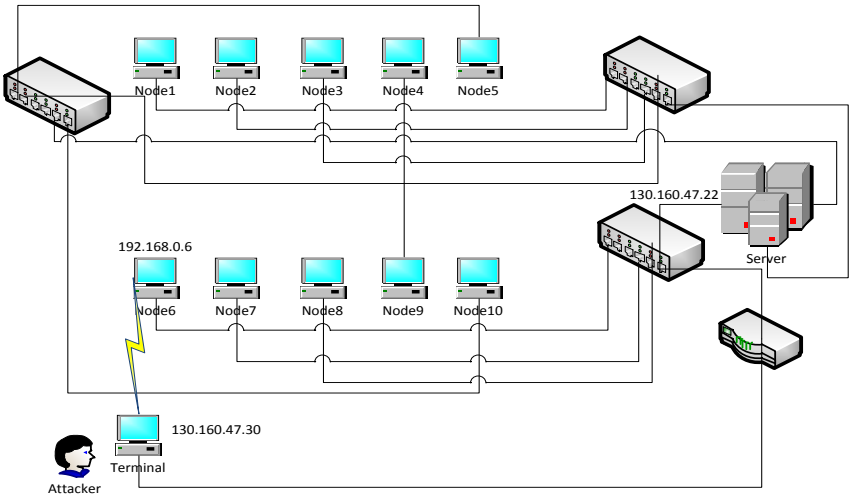


Figure 12: Setup for spoofing a node inside the cluster

The simulation was considered under the assumption that an adversary somehow managed to get control over an IP address inside the cluster. Then flooding was generated through a terminal, which accessed the network via the E900 router. The flood-originating terminal (IP address: 130.160.47.30) used Node6 (IP address: 192.168.0.6) for sending spoof packets towards the server as shown in Figure 12. Actually this type of attack is also known as a *bot* where the legitimate users (Node 6) are manipulated by a handler and trace back is difficult.

Due to the possibility of fewer users in the cloud (private cloud) to analyze any real time data, TCP-SYN flooding is operated from several nodes targeting the Ubuntu server while the server was busy executing SQL queries. In our experiments, the measurements are kept in the Profile Generator for 1000 packets before they are refreshed for new types.

5.2 FLOODING AND FAPA FILTERING ALGORITHMS

Before illustrating our results we will discuss the algorithms used for flooding the system. We also present our algorithm for filtering. To implement flooding we used the C language with Linux sockets as shown in Algorithm 1. TCP and IP headers were constructed after the construction of the raw sockets. A datagram was allocated to represent the packets. Headers are of 20 bytes in length but could vary based on the payload size. Sometimes TCP packets are more than 40 bytes, so for the ease of experimentation we did not conduct flooding with those kinds of TCP packets or IP packets. The loop in Algorithm 1 can be defined to send various numbers of spoof packets for generating flooding packets. In our experiments we have conducted two types of flooding: (1) Uncontrolled flooding, which continuously sends spoof TCP packets to the target; and (2) Controlled flooding, which sends a fixed number of spoof packets. The *Loop* label in Algorithm 1 contains the spoof packet's socket and buffer for header information including the datagram length and address.

As shown in Algorithm 2 filtering is accomplished with the help of a handler. First, all the raw packets are dumped in the packet buffer and the string builder was defined for handling the error messages. Then the network device from which we fetch the raw packets is selected. Next, the handler transfers the headers of each packet one by one and extractions of the headers take place. If the spoof counter has exceeded a defined threshold, then filtering is invoked. An alarm is also generated and broadcasted by the handler if the handler instance is not similar to the expression, optimization values and the net mask.

Algorithm 1: SYN Flood DoS with Linux sockets

// including the library for memory set

// including standard library for exit (0)

// including errno*– for the error number*

// including netinet/tcp.h – declarations for TCP header

//including netinet/ip.h – declarations for IP header

Pseudo header:

Construction of a pseudo header for checksum:

for calculation of tcp *and* ip *packets*

Construction of a Short CSUM:

While(number of bytes > 1)

*sum = sum + *ptr++;*
number of bytes = number of bytes – 2;

Flood Method Begin:
Creation of a raw socket

Allocation of a Datagram to represent the packet

Construction of IP Header structure

Construction of TCP header structure:

Target address is assigned here

Assignment of IP Header fields

Assignment of TCP Header fields:

Window size is allocated
Kernel's IP stack is filled with correct checksum during transmission

Construction of IP checksum

Construction of IP_HDRINCL *to tell the kernel that headers are included in the packet*

Construction of Loop to flood the target

Loop: (Can be infinite or finite)

Sending the packets

if
Packets containing socket, buffer containing header and data, length of the datagram, routing flags and socket address
endif
else
Print Error

end Loop

END SYN Flood DoS

Algorithm 2: SYN Flood DoS Filtering

// including arraylist to keep track of all network devices

// including list to keep the packets as a linked list

// including JPacketHandlerto handle the packet headers

// including PcapPacket – points to the device list

//including protocol – distinguish among TCP, IP, UDP and HTTP headers

Construction of a String Builder:

Handles the error messages

Construction of ArrayList:

Listing of all the network devices
Selection of desired network device for traffic
Setting flags in Promiscuous mode
Open one device in Live and start capturing

Defining a JPacketHandler:
Allocation of memory: tcp, ip, udp, http headers;
Initialization of counters for different packets;

if packet has header with ip4
Extraction of source address byte from ip4
Conversion of byte to string;
if address matches Original address
Original_counter++;
if address matches Suspicious address
Spoof_Counter++;
If Spoof_Counter> Threshold
go to FilterSetting;
end if
end if
end if

Construction of FilterSetting
// including bpf_program *structure.*
// instance of a compiled Packet Filter Program.

Initiation of an instance with Bpf Program
if (compilation of bpf instance, expression, optimize and netMask != handler instance. Cont)
print "generate alarm for flooding"
Propagation of message to all the connected VMs
Filtering of the originator collected by handler
end if

END SYN Flood Filtering

5.3 RESULTS

Our experiments were conducted in several scenarios to discover any prominent behavioral patterns. Traffic capturing was measured with flooding, without flooding, and also with controlled flooding, where we flooded the system with a finite number of packets. Several patterns were identified based on the results of the conducted experiments.

5.3.1 Comparison of packets in an Uncontrolled Environment

In this scenario we checked the ratio between forward and reverse packets in both environments, with and without flooding. All the users (nodes) make TCP requests (SQL queries) for the server to process. The incoming TCP-SYN requests are the forward packets and the SYN-ACK responses from the server are the reverse packets. Figure 13 illustrates the forward and reverse packet ratio for the TCP protocol in a non-flooded environment. Figure 14 identifies the ratio in a flooded environment. The total number of packets captured increases from 150 to 5000 in Figures 13 and 14.

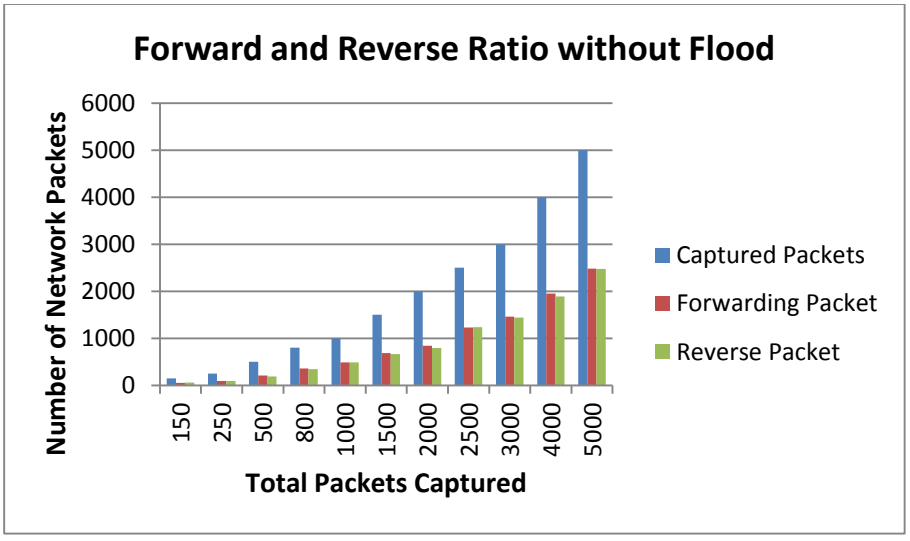


Figure 13: Forward and Reverse Ratio without Flooding

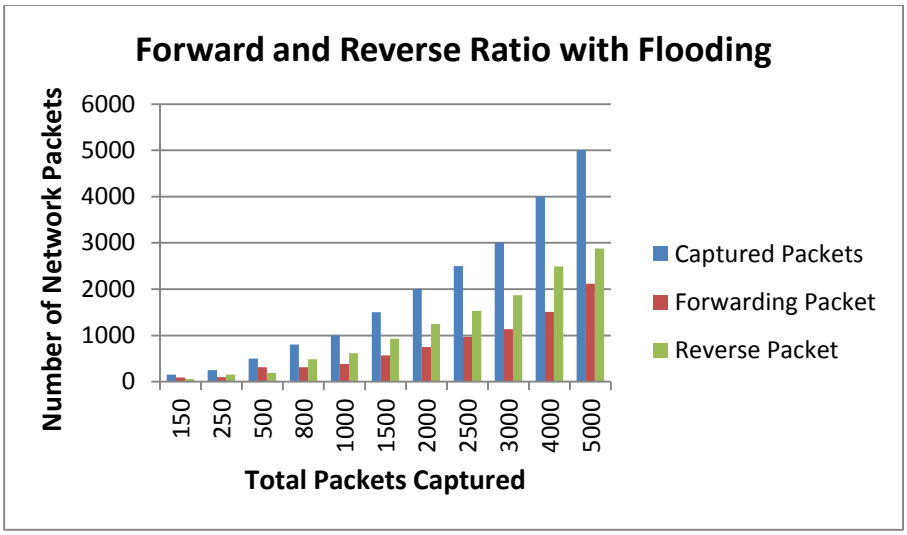


Figure 14: Forward and Reverse Ratio with Flooding

Analysis-

1. In Figure 13 the number of forward packets is almost equal to the reverse packets. This means a user makes a TCP-SYN request, receives the SYN-ACK response from the servers and responds with the ACK message. Our observation is that this is the ideal case in a non-flooded environment.
2. In Figure 14 there is a 19% increase in reverse packets on average in comparison to forward packets. The number of reverse packets ranges from 15% to 25%, whereas no increment is observed with no flooding. According to TCP topology, the amount of SYN-ACK increases only when the server does not receive the final ACK message from the requester. Our observation confirms that flooding causes an increase in the number of reverse packets.
3. Also in Figure 14, the sum of forward and reverse packets in each capturing is equal to the total number of captured packets. But in Figure 13, the sum of forward and reverse packets is less than the total number of captured packets. Because the sum of the packets

is equal to the total, these results demonstrate that different types of packets (IPv4, ETH) are not allowed by the spoof TCP packets to make their way into the flooded environment.

Hence, for this scenario we detected flooding by checking the percentage of reverse packets in comparison to forward packets. Also, we detected missing packets in a flooded environment, as only TCP packets were allowed to pass from the server.

5.3.2 Comparison of packets in a Controlled Environment

In the second scenario, the number of forward packets was compared with the number of reverse packets in a controlled environment. In Figure 15 and Figure 16, flooding was accomplished by an incremental but deterministic approach; we sent 500, 1000, 1500, 2000, 2500, 3000, 4000 and 5000 spoof packets for each observation.

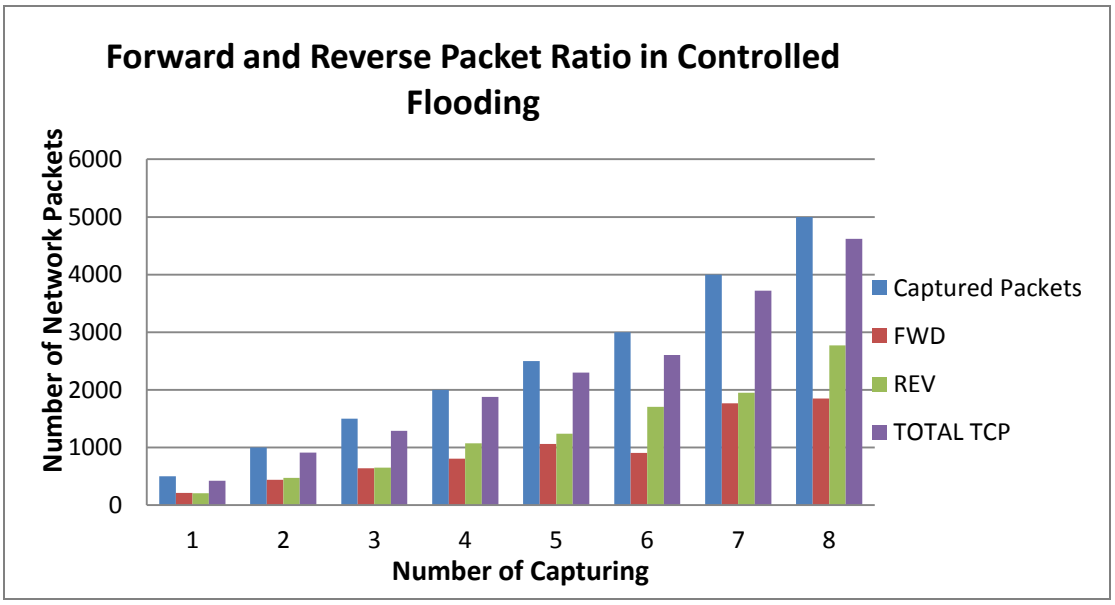


Figure 15: Forward and Reverse Packet Ratio in Controlled Flooding

Analysis-

1. The number of forward packets (FWD) is still less than the number of reverse packets (REV). In a controlled flooding setting, the average percentage of reverse packets increased to 11%. The results ranged from a 3% increment to a 27% increment in reverse packets with controlled flooding.
2. In Figure 15, the sum of forward and reverse packets (Total TCP) is less than the captured packets (Captured Packets), so controlled TCP flooding allows other packets, such as IPv4 and ETH. IPv4 had a low range of 1% to 4% packets in a controlled flooding environment.

In Figure 16, we illustrate the number of IPv4 and Eth packets in comparison to captured packets, which were observed in the controlled flooding environment. The same experiment was conducted with the pre-determined number of flooding packets starting from 500 to 5000 packets.

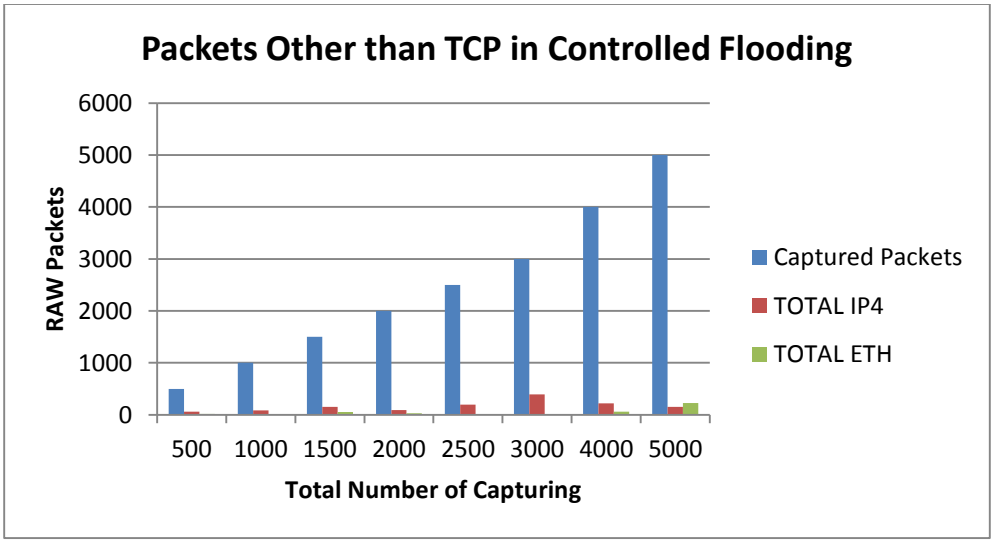


Figure 16: Packets Other than TCP in Controlled Flooding

Analysis-

1. Figure 16 illustrates that IP4 and Eth packets were allowed in the server during the controlled TCP flooding, but their percentages are very low in comparison to TCP packets.
2. Compared to TCP, only 9% of IPv4 packets and 2% of Eth packets were allowed in the average case. These IPv4 packets contain the source address of the originator.

Hence, from this controlled flooding environment, we detected flooding again with the percentage of reverse packets. Different types of packets are allowed here, but not a significant amount. We extracted the IPv4 packets to detect the source of the DoS originator. Tracing back to the initiator of flooding is easy in a controlled flood environment.

5.3.3 Comparison of forwarded packets in both Environments

In this experiment, the server was flooded with continuous TCP flooding by capturing packets in bulk. The difference between the total number of TCP packets in a non-flooded and non-controlled flooded environment is provided in Figure 17. The total number of packets captured ranged from 500 to 4000.

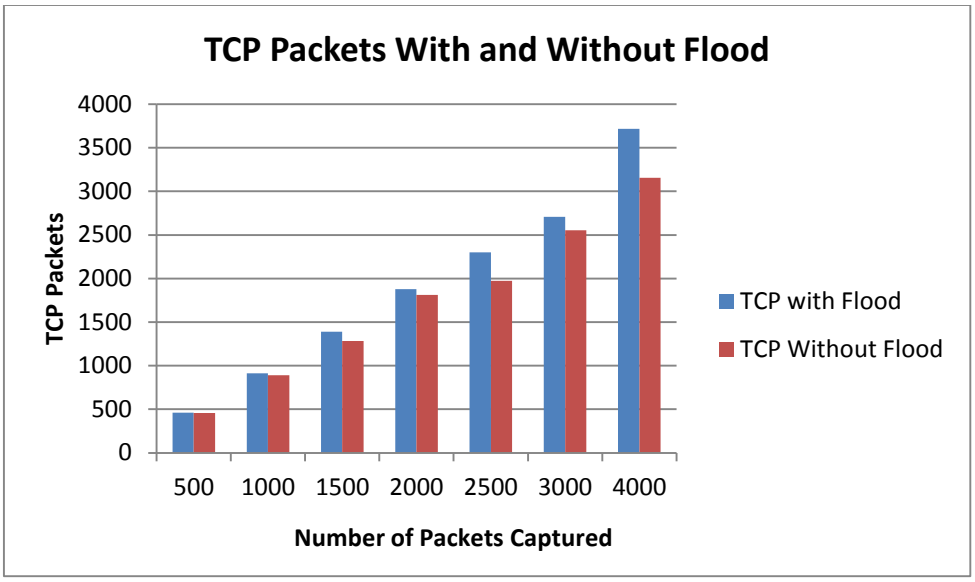


Figure 17: TCP Packets with and without Flooding

Analysis-

1. As Figure 17 illustrates, there is an increase in number of TCP packets in a flooded environment. The TCP packets increase in number from a range of 1% to 14%. The average is a 7% increase in TCP packets in a flooded environment.
2. Other than TCP packets, no other network packets passed the server, so UDP, HTTP requests were starved, as they were absent during the capture.

A similar experiment was conducted for reverse (SYN-ACK) packets leaving the server, in a non-flooded and flooded environment. The results are shown in Figures 18 and 19, respectively, with the number of packets captured ranging from 500 to 4000.

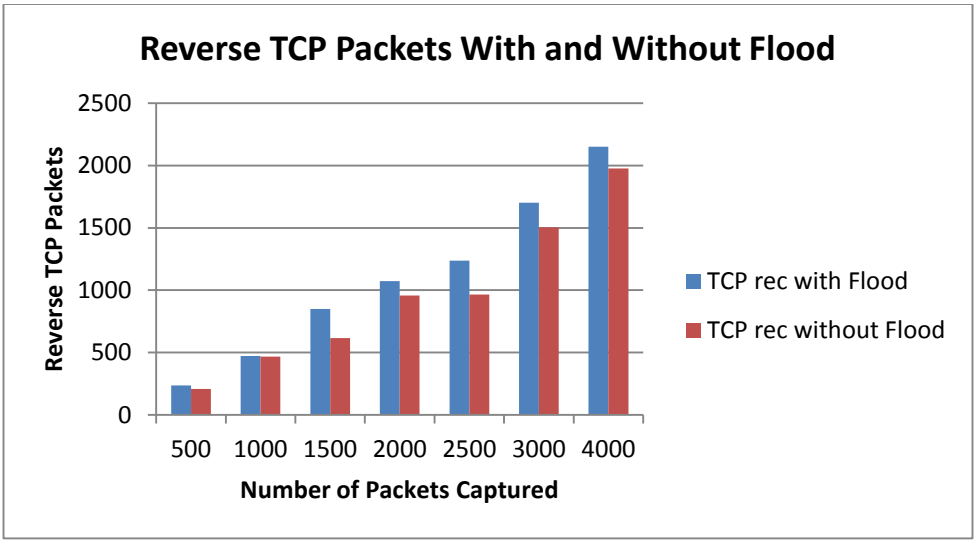


Figure 18: Reverse TCP Packets with and without Flooding

Analysis-

1. In this experiment, the number of reverse packets increased from 6% to 16% for 500 to 2500 captured packets, respectively. After 2500 packets, a dramatic increment in reverse packets resulted in an average increment of 7% in a flooded environment. These results indicate that spoofing with 10,000 or more packets would invoke an extensive amplification of flooding for a cloud system.
2. Such type of amplification could cause replay attacks to damage to the entire distributed system.

5.3.4 Comparison of IP packets in both Environments

In this scenario, we captured incoming packets for the server and compared the IPv4 packets in both the flooded and non-flooded environments, for the number of packets captured ranging from 500 to 4000.

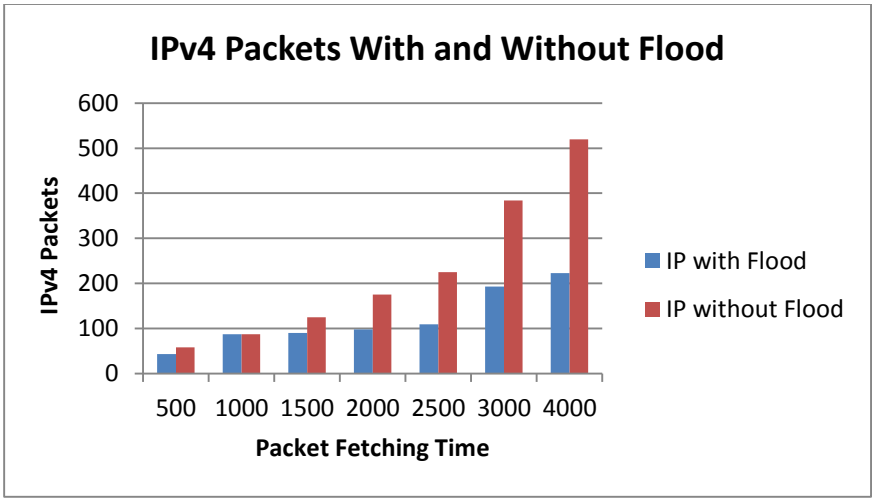


Figure 19: IPv4 Packets with and without Flooding

Analysis-

1. In Figure 19, without flooding the legitimate IPv4 packets were allowed to pass through the servers, so there was a continuous increase in the IPv4 packets in each capture. For 2000 captured packets, we found 13% IP packets, and for 4000 packets the percentage increased to 28%. Statistically, the relation of IPv4 packets with total captured is exponential.
2. IPv4 packets without flooding were greater than the IPv4 packets with flooding. For 1500 packets captured the percentage of IPv4 packets in a flooded environment was 2%. With 3000 packets the percentage increased to 6%. We expected IPv4 packets to double for bigger captures, but with 5000 packets we found only a 7% increase in IPv4 packets. With continuous flooding, the IPv4 increment is almost linear rather than exponential or polynomial.

In the above scenario we found that flooding makes the IPv4 packet passing linear, which is exponential in nature without flooding.

Hypothesis: Traffic behavior can be a marker for flood detection in a cloud system.

5.3.5 Comparison of Payload Size for Different Packets (without flooding)

In this experiment we captured 10 packets individually, *one by one*, rather than using a packet handler for bulk capturing. We conducted the experiment without flooding at first, then with flooding as shown in Figures 20 and 21, respectively. We measure and compare the payload size of each packet type for 10 packets captured.

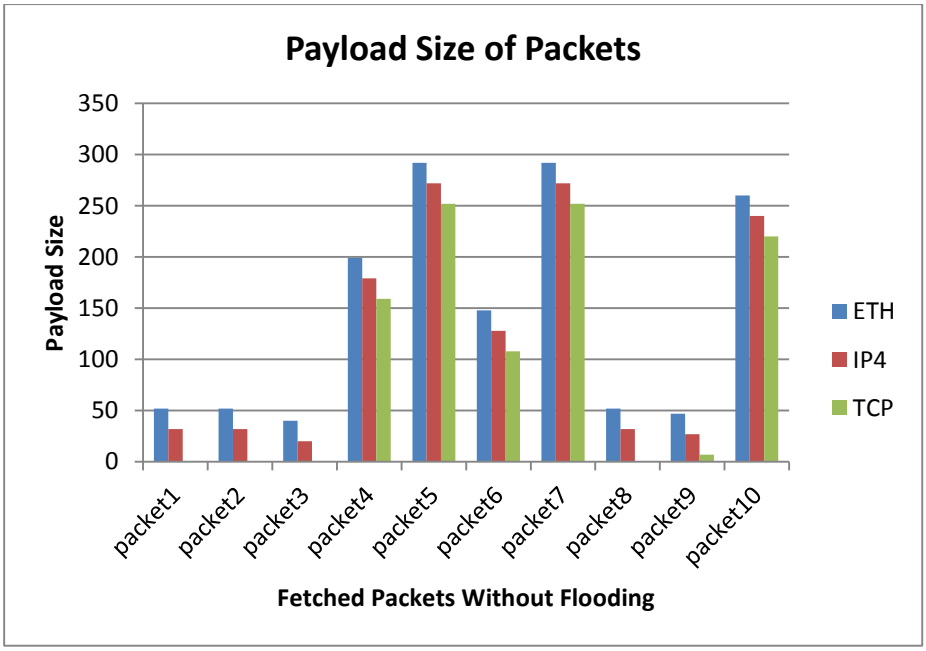


Figure 20: Payload Size of Packets without Flooding

Analysis-

1. We observe from Figure 20 that the size of Eth is always higher than the other packets (IPv4 and TCP). The reason is that in the ideal case Eth is the parent class to wrap the next layer of packets in its payloads. So Eth should be higher than the other packet's payloads.
2. On average, IPv4 occupies 35% and TCP occupies 15% of the payload size.
3. On average, the payload size of Eth is 750 bits, IPv4 is 123 bits and TCP is 100 bits.

We then conducted the experiment in an uncontrolled flooding environment. We compared the payload sizes of each type of packet to check if there was any change in payload size due to flooding.

Hypothesis: Any change in payload sizes could be utilized as a parameter for DoS detection in a cloud system.

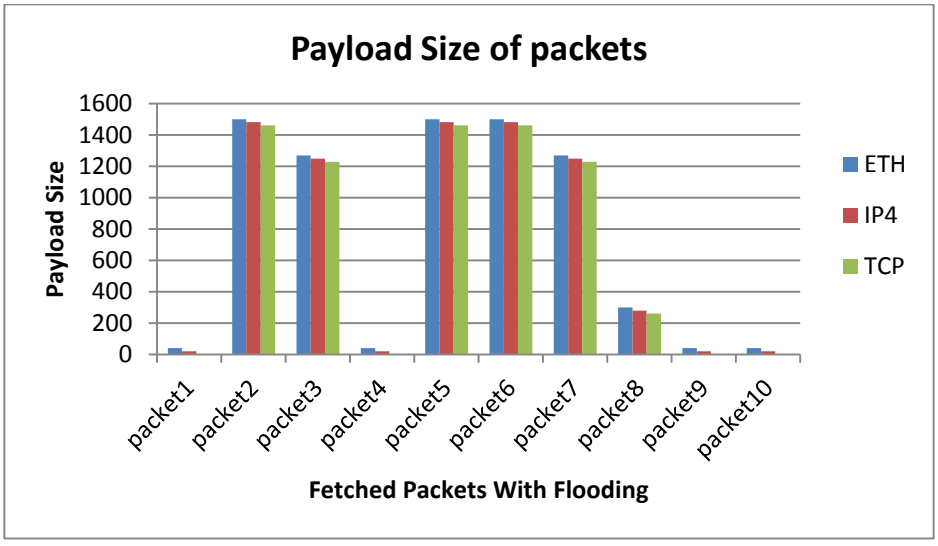


Figure 21: Payload Size of Packets with Flooding

Analysis-

1. Our observation from Figure 21 is that the payload size increases significantly for some packets. The average payload sizes with flooding are 750 bits, 730 bits, and 709 bits respectively, for Eth, IPv4 and TCP packets. However, the bits of payload size range from a low of 40 to a high of 1500 bits for Eth packets, and IPv4 payload size ranges from a low of 60 to a high of 1268 bits.
2. Payload percentages of the total packets are 47%, 33% and 20% on average for Eth, IPv4 and TCP packets, respectively.

From the above experiment, we observe that the payload size increases significantly for different packet types if flooding is conducted. We have not seen any significant increment in average payload percentage, but we definitely encountered a large increment in the actual size of the payloads. Our hypothesis is proved in a cluster by our initial results, demonstrating that we can utilize the payload size increment behavior of traffic to detect DoS.

5.3.6 TCP Window Scale in Variant Environments

In this experiment we observed the effect of flooding on the TCP packet's window length. We captured packets using the loop handler described in Section 4.1, which is designed for continuous capturing and conducted flooding to see if the window scale changed. We performed flooding three times, and we took 5 samples for each scenario and averaged them. Figure 22 illustrates the results of the experiment in which the number of packets is displayed in the horizontal axis and the Window Length is displayed vertically.

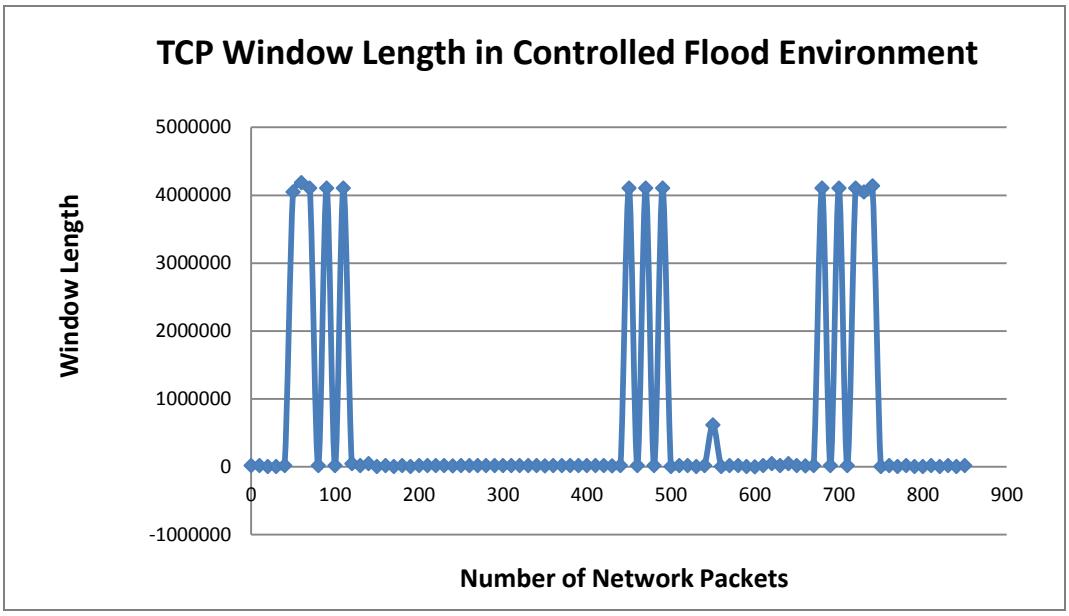


Figure 22: TCP Packet Window Size

Analysis-

1. As shown in Figure 22, flooding was conducted between 50 and 150 packets, then between 500 and 600 packets, and finally between 700 and 800 captured packets.
2. Our observation in this experiment was beyond our expectation due to the rapid rise in the window length of TCP packets. Also, the periodicity of length change completely corresponded with the flooding events. This traffic behavior can be used to monitor the traffic change in the event of a denial of service.

This experiment has demonstrated that we can detect any intrusion activity trying to flood the cloud system by monitoring the window scale change in TCP packets, when the flooding is with TCP-SYN packets. For UDP flooding we expect to encounter a similar pattern in UDP window size as well. In the next section will describe the details of these experiments in a cloud and the impact of flooding (DoS) on the virtual machine instances in the cloud.

5.4 EXPERIMENTAL SETUP FOR PRIVATE CLOUD

The experiments were conducted using a Eucalyptus cloud. For the cloud we used euca2ools, and there were four OptiPlex machines hosting the Ubuntu servers 12.04 LTS to build the cloud. One machine hosted the cluster controller (CC) with an Intel(R) Core(TM) 2 Duo processor 2.6 GHz speed and 120 GB ATA hard disk (Serial ATA bus interface connects the host bus adaptors to mass storage devices) from *Seagate*. Another OptiPlex 755 with an Intel Core 2 Duo with a 2.33 GHz and 120 GB ATA hard disk from Seagate hosted the cloud controller (CLC), the walrus controller (WC) and the storage controller (SC). Both the CC and CLC were connected via a *DLink* Gigabit switch, DGS-2208. An OptiPlex 755 with a 2.83 GHz Intel Core 2 Duo processor and a 160 GB ATA hard disk hosted the first node controller (NC1). The second node controller (NC2) was hosted on a *Foxconn* motherboard with a 2.6GHz processor speed Intel Pentium(R) and a 120GB ATA hard disk from Western Digital. Both NC1 and NC2 were connected via a *DLink* Gigabit switch, DGS-2208. These two switches were connected to the public network via a Linksys *E900* Cisco router.

NC1 hosted 3 and NC2 hosted 4 VMs. We used KVM and vmbuilder to create each of the VMs with 4GB capacity, and a 2GB swap partition and 4GB variable partition space. For our performance analysis in the cloud, we calculated the bandwidth, transfer rates and throughput of seven VMs. All the VM images were created under the directory of *libvirtd* with root privileges provided to the users, so that users can easily run, suspend, resume and shutdown the VMs when required. Openssh was installed via a batch file, because we wanted the VMs to exchange the one time generated key with the physical host NC1 at the beginning of the VM creation. Hence,

two scripts were running in the background when a VM was created, a script for partitioning and another for new password creation with the Openssh installation in the VMs. NC1 was interfaced with 3 virtual networks (vnet0, vnet1, vnet2) and NC2 was interfaced with 4 virtual networks (vnet0, vnet1, vnet2 and vnet3). Table I mentions the IP addresses used for each VM.

Table I: IP table for the Virtual Machines

Node	**Virtual Machine**	**IP**
NC1	*VM1*	192.168.1.250
	VM2	192.168.1.251
	VM3	192.168.1.252
NC2	VM5	192.168.1.173
	VM6	192.168.1.174
	VM7	192.168.1.175
	VM8	192.168.1.176

For the attack, an outside terminal with Ubuntu 12.08 Desktop edition OS and an Intel(R) Core 2 Duo machine with a 2.6 GHz processor was connected via the *E900* Cisco router. This external attacker generated an enormous amount of spoofed TCP packets. We used *iperf* server in the VMs to measure the bandwidth and transfer rates from the external attacker running as a *jperf* client (*jperf* – a graphical interface of *iperf*). Also, we used the bandwidth manager known as *bwm-ng* to measure the throughput in terms of transmitted and received packets for each *vnet* interfaces of both node controllers, NC1 and NC2.

5.5 IMPACT ON VIRTUAL MACHINES

NC1 hosted VM1, VM2 and VM3 and a suspended node VM4. NC2 hosted VM5, VM6, VM7 and VM8. To conduct the experiment we compromised one virtual machine in each physical node to observe the effect. Unless otherwise noted, in the figures appearing in the following subsections, *blue* bar illustrate the results for each VM before the flooding and *red* bars illustrate the after effect of flooding for each VM.

5.5.1 Compromising VM6 in NC2

At first we compromised VM6 in NC2 via the external windows terminal. The TCP flooding was conducted from the external terminal connected via E900 with destination address 192.168.1.174. The bandwidth of VM6 decreased from 94.2 Mb/sec to 67.9 Mb/sec. Figure 23 illustrates the experiment measuring the bandwidth before and after flooding takes place.

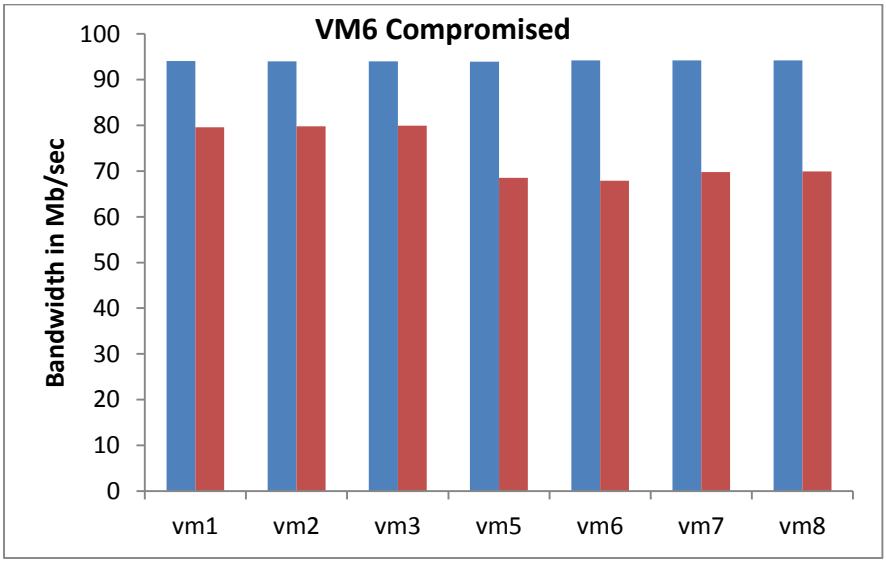


Figure 23: VM6 of NC2 was attacked by TCP packets from a terminal

Analysis-

1. In Figure 23, the bandwidth of VM6 along with its sibling VMs was affected the most with flooding, as it was the compromised VM. The bandwidth of VM6 decreased about 29% compared to the bandwidth without flooding.
2. Bandwidth reduction among the siblings (VMs hosted in same node NC2) was about 27% and among the neighbor VMs (VMs hosted in the neighbor node NC1) bandwidth was reduced about 15.17%. The overall average bandwidth reduction was 17.34% with flooding. NC2 is more affected than NC1.

Next the transfer rate of network packets was measured when VM6 was under attack with TCP packets, Figure 24 illustrates the transfer rates before and after flooding takes place.

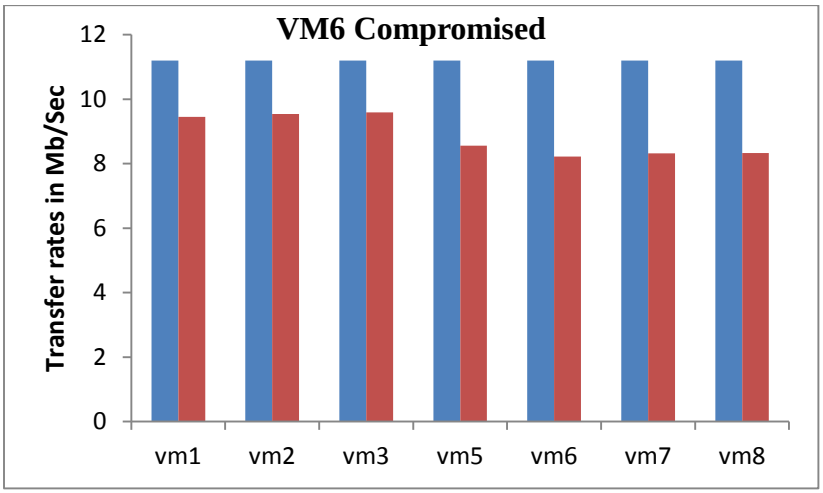


Figure 24: VM6 of NC2 was attacked by TCP packets from a terminal

Analysis-

1. In Figure 24, the transfer rates for VM6 decreased about 26.6% when flooding occurred.

2. The average transfer rate for the NC2 virtual machines is 25.38%, whereas for NC1 it is 14.94%. The overall average transfer rates dropped about 21% with flooding. VMs within the same node NC2 as the compromised VM, suffer more than in the neighbor node NC1 when flooding occurs.

5.5.2 Compromising VM1 in NC1

Next to determine if these results would be consistent across each NC, we perform the same experiment by compromising VM1 in the other node NC1. TCP flooding was conducted from the external terminal with destination address 192.168.1.250. VM1 bandwidth decreased from 94.1 Mb/sec to 69.4 Mb/sec with flooding. Figure 25 highlights the outcome of flooding by illustrating the bandwidth before and after flooding takes place.

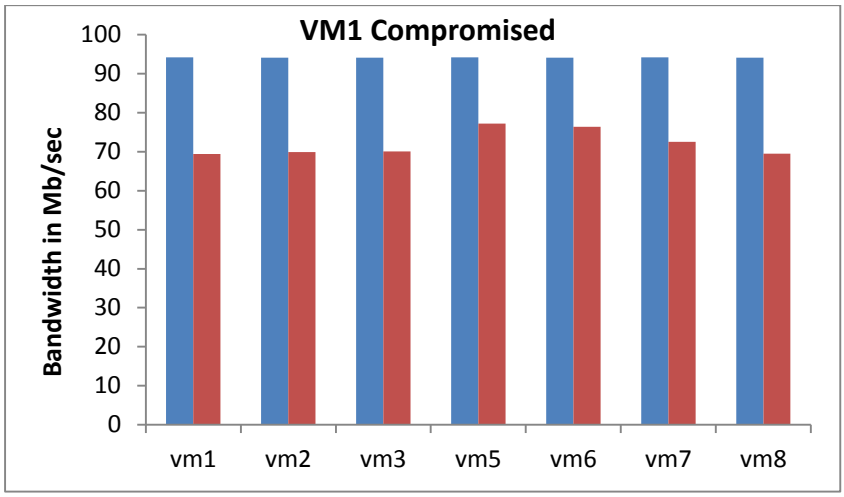


Figure 25: VM1 of NC1 was attacked from TCP packets by an external adversary

Analysis-

1. In Figure 25, the bandwidth of VM1 decreased about 26.3% when flooding occurred.

2. The average reduction in bandwidth is 23.53%. In NC1 the average reduction is 23.9% and in NC2 the reduction average is 21.51%. Again, the NC containing the compromised VM is more affected than the NC without the compromised VM.

The attack on a VM resulted in a reduction in the VMs bandwidth that ranged from 26% to 29%. The average bandwidth reduction across all VMs on the same NC as the compromised VM ranged from 24% to 27%. The average bandwidth reduction across all VMs on a different NC as the compromised VM ranged from 22% to 15%.

Figure 26 illustrates the transfer rates of network packets while flooding was conducted on VM1 of NC1.

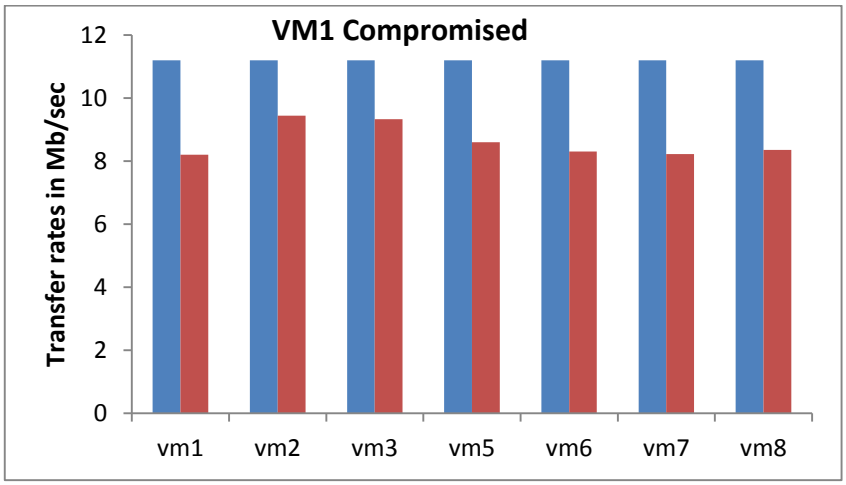


Figure 26: VM1 of NC1 was attacked from TCP packets by an external adversary

Analysis-

1. The transfer rate of VM1 has been reduced to a maximum of 26.79 % due to flooding.
2. The average transfer rate dropped about 22.87%. In NC1, the average reduction is 19.73%, whereas in NC2, the neighbor host was affected with a 25.22% reduction in transfer rates.

The attack on a VM resulted in a reduction in the VM's transfer rate of approximately 27% in both experiments. The average transfer rate reduction across all VMs on the same NC as the compromised VM ranged from 25% to 20%. The average bandwidth reduction across all VMs on a different NC as the compromised VM ranged from 15% to 25%.

From the above data we make the observation: *Compromised VMs in a Cloud not only affect the co-resident VMs, but the bandwidth and transfer rates of the VMs residing on neighbor nodes are also impacted. For transfer rates, we are unable to determine if sibling VMs on the same node as the compromised VM are more affected than their neighbor VMs on different nodes.*

5.5.3 Throughput Measurement with bwm-ng

In our next experiment we wanted to see the effect of flooding on throughput. To do so, we used bwm-ng in the node controllers (NC1 and NC2) and flooding was conducted on each VM individually (both NC1 and NC2).

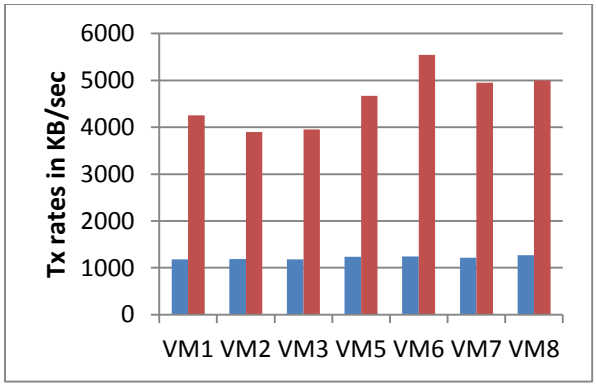


Figure 27: Tx rates in KB/sec external

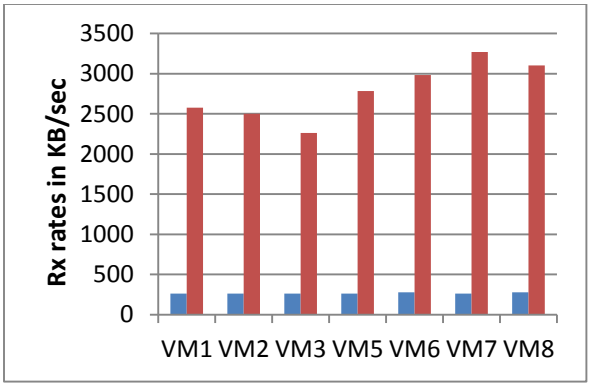


Figure 28: Rx rates in KB/sec

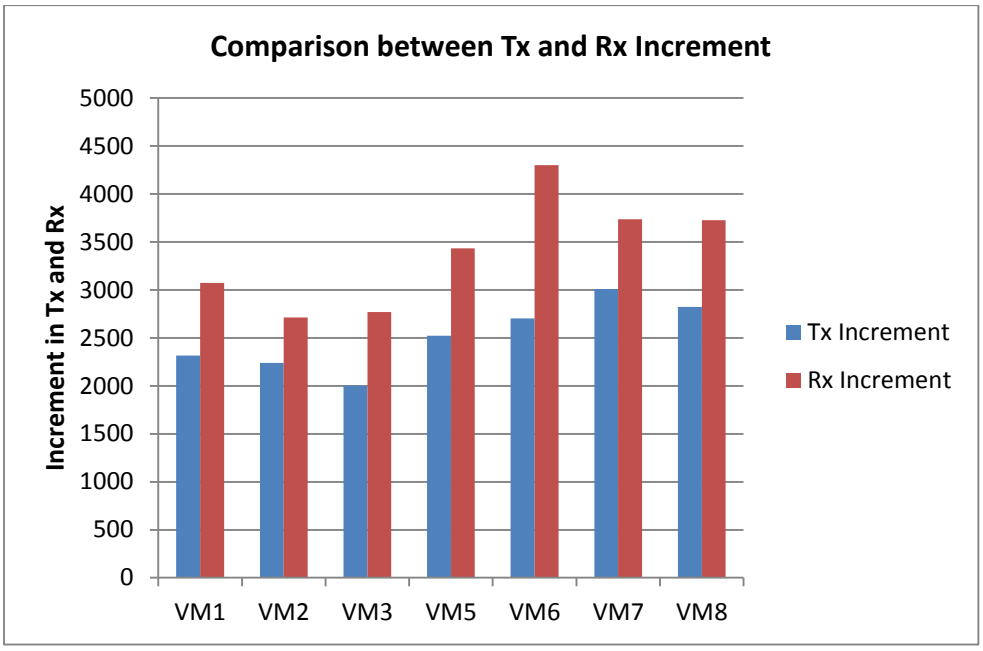


Figure 29: Tx and Rx increment in each VM of the cloud

Analysis-

1. Figure 27 illustrates that the transmission rates increased up to 4299.48 KB/sec with flooding. On average the transmission rate went up from 1216.71 KB/sec to 3392.53 KB/sec, while flooding was conducted in each VM. Flooding a VM resulted in an increase in 3 times the transmission rate for each VM in the cloud.
2. As illustrated in Figure 28, the receive rates increased up to 3007.93 KB/sec. On average the Rx rate went up from 266.94 KB/sec to 2515.61 KB/sec while the system was flooded with TCP packets. This increment in a flooded environment was almost 10 times greater compared to a non-flooded situation in the cloud.

3. In Figure 29, only the increments in each VM were plotted with respect to Tx and Rx rates. The effect of flooding on Tx and Rx rates is greater in the VMs of NC2 (VM5, VM6, VM7, VM8) than NC1 (VM1, VM2, VM3).

During our experiments we observed no change in Tx and Rx rates in the non-compromised virtual networks. Therefore, we make the observation: *Flooding from an external attacker on a targeted VM affects the Tx and Rx rates to a great extent on the virtual network of the compromised VM but it does not hamper the other virtual networks of the network interface (co-resident and neighbor vnets are not affected).*

In the next chapter we describe how the results from the experiments in this chaper are incorporated into our FAPA model.

Chapter 6

PROTOTYPE FILTERING

6.1 INITIAL RESULTS

After studying the traffic behavior from the experiments described in section 5.3, we incorporated the observed behaviors in the FAPA filtering mechanism as follows. Specifically, we check: 1) the ratio between the forward and the reverse packets, 2) the payload sizes of packets received and 3) the window scale change to identify the event of an attack. The Comparison Checking module of the FAPA filtering mechanism incorporates results from our experiments in section 5.3 as follows. If the number of reverse packets in the traffic flow is more than the forward packets then we investigate the suspicious node according to its payload. From our payload investigation, we observed that if the payload size increases abruptly compared to a regular payload size, which ranges from 40 to 750 bits, then we randomly check the window scale length of the traffic flow. A significant increment in window scale length triggers the event of filtering. Normally the window length is within 60 or 120 bytes, but due to flooding that length could increase to 300KB to 400KB, as shown in Figure 22. At the beginning of filtering, each packet's byte address is converted into an actual IP address and then compared to the original IP address (host IP address) that was obtained from the IP address table.

6.1.1 Comparison of Different Packets with Filtering

In this experiment, we determine the effectiveness of our filtering strategy. This experiment was conducted on our cluster described in Section 5.1, where node 6 is the non-legitimate node. The

experiment begins with flooding and no filtering. Once flooding is detected, filtering is applied and then the spoof packets are eliminated. The types of packets we highlighted in this experiment were: TCP packets, UDP, HTTP, Spoof IPv4, Original IPv4 and Miscellaneous IPv4. All IPv4 packets with suspicious addresses are considered as spoof IPv4 packets. IPv4 packets with legitimate IP addresses are considered original packets (other than node 6 the rest of the nodes are legitimate) and the rest of the IP packets are considered as miscellaneous packets.

Figure 30 illustrates the number of network packets for TCP, when packet filtering was utilized after flooding was detected. Figure 30 illustrates the number of each type of packet sampled during filtering. Each sample is 500 network packets captured. In Figure 30, blue represents the number of packets as flooding begins but before filtering is applied. The red, green, purple, teal and orange colors represent the progressive improvement achieved after filtering is applied. Each of these bars represents the periodic capturing of packets once filtering was invoked.

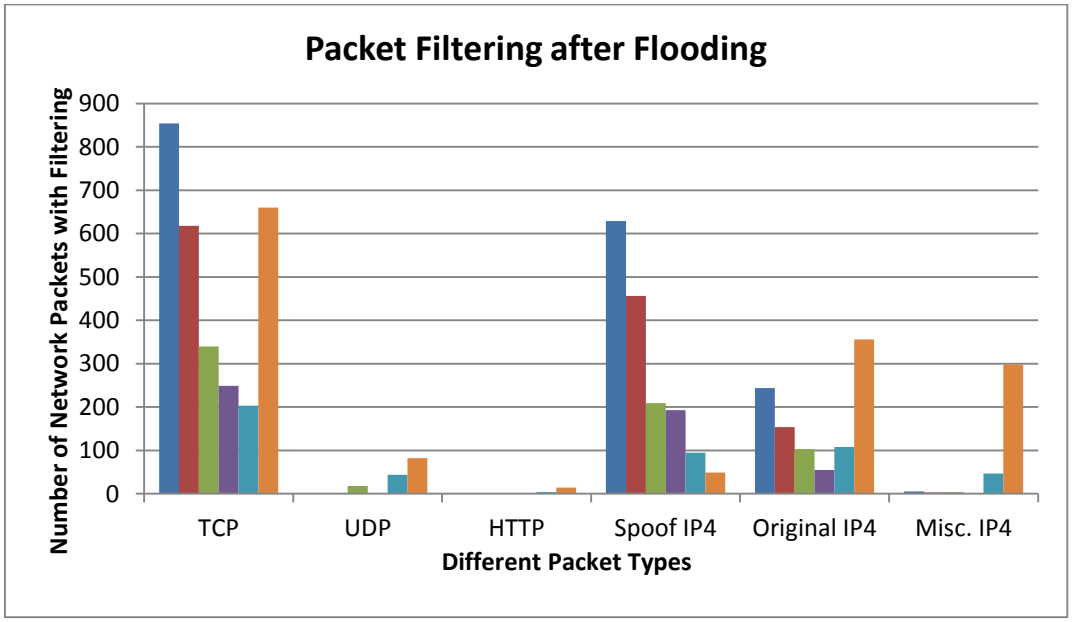


Figure 30: Packet Filtering after Flooding

Analysis-

1. From Figure 30 above, it is observed that in the beginning the legitimate TCP packets began decreasing as flooding was conducted with spoof TCP-SYN packets. The number of legitimate TCP packets decreased by 76% as illustrated by the blue to teal bars.
2. Then we applied filtering and in the next capturing we found significant improvement as measured by TCP packets passing through the server. The percentage of TCP packets increased to 54% as illustrated by the orange bar.
3. Spoof IPv4 packets started to decrease because flooding was eliminated by filtering. At the beginning of filtering, the packets decreased by only 28%, but later we filtered about 88% of the spoof IP4 packets with filtering.
4. The Original IPv4 packets behaved similarly to the TCP packets. In the beginning, the original IPv4 packets decreased by 37% but with filtering it started to increase. According to our observations, there was a 97% increment in original IPv4 packets among the VMs after filtering was invoked, as illustrated by the orange bar.
5. The Miscellaneous IPv4 packets started to increase after filtering was applied, but at the beginning of flooding they were not noticeable at all.
6. UDP and Http packets were very negligible at the beginning when flooding was conducted before filtering. Later, both types of packets started to arrive at the server as filtering decreased the amount of spoof TCP packets.

We made some significant improvement by filtering the spoof packets while conducting TCP flooding. As mentioned earlier in Chapter 4, the filtering was conducted from a user terminal

and not from the routers. The results indicate it is possible to provide a robust security in a cloud system from the user's end, rather than having complete dependency on the providers.

6.1.2 Comparison in Capture Time

While experiments in the previous section demonstrated the ability to detect and filter packets when flooding occurs, there is an overhead associated with identifying and filtering the spoof packets. In order to measure this overhead, in this experiment we conducted a time analysis of for capturing packets in three scenarios: without flooding, with flooding, and with filtering. Findings in this experiment will illustrate the overhead of filtering in terms of milliseconds.

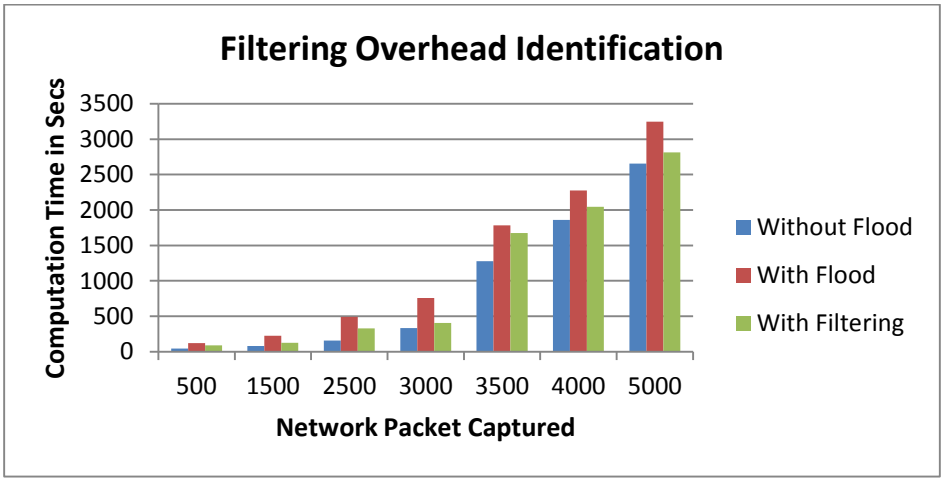


Figure 31: Filtering Overhead Identification

Analysis-

1. It is observed in Figure 31, that flooding increases the processing time, as expected. Flooding increases the computational time by 45% on average.

2. After filtering, an improvement in processing time was expected. Filtering improved the computational time by 25% on average.
3. However, filtering increases the capturing time when no flooding occurs.

Let's consider *t*, as the time taken to capture traffic packets without flooding and no filtering, t_f, is the time to capture traffic packets with flooding and consider t_F, as the time consumed with FAPA filtering in the Validation checking module. We consider α as the correlation between the flooding and filtering. Based on the value of α we can determine if any DoS attack is flooding the system or not. Suppose we name α as the *filtering constant*, whose value should be less than one. If α>1 then filtering would provide no additional advantage over flooding.

Utilization of cloud with flooding = $(t_f\text{-}t) \,/\, t$ …….... (1)

Utilization of cloud with filtering = $(t_F\text{-}t) \,/\, t$…… (2)

Combining (1) and (2) we can determine the value of the *filtering constant* α,

$$\frac{(t_f - t)}{t} = \alpha \frac{(t_F - t)}{t}$$

Using the above formula we have constructed the table below based on different types of capturing:

Table II: *filtering constant* α findings

Network Packets	t	t_f	t_F	α
500	45	120	90	0.60
1500	78	222	125	0.33
2500	158	492	330	0.51
3000	333	755	406	0.17
3500	1276	1784	1676	0.79
4000	1859	2275	2043	0.44
5000	2657	3248	2814	0.27

From Table II it is seen that we have achieved α from a low value 0.17 to a high value 0.79. The lowest value is the best achievement while 3000 network packets were captured during the flooding. But on average α was 0.44<1, also a good indication of the efficiency with FAPA filtering.

Here the observation is: *though filtering consumes more time than not filtering in a non-flooded environment, the time consumed by filtering does not exceed the time consumed by flooding*.

6.2 MODIFIED FAPA FILTERING ALGORITHM

In Section 6.1 our FAPA filtering scheme is based on threshold values, which were collected from the behavior of the traffic characteristics in the event of an attack. We considered the ratio

between TCP reverse and forward packets, TCP Window scale, number of UDP and Eth packets etc. All these traffic attributes could have variable default thresholds depending on the size and type of the application running in the cloud user's VM. In this section we describe how the FAPA filtering algorithm is modified to generate a profile for the arriving packets. Every arriving packet has some nominal values embedded inside the packet. Those values could be utilized to pursue the tracing of spoof packets and filtered.

Each packet has a TTL value. That value identifies the number of hops of each packet [2]. Upon receipt of a packet, the source IP address is extracted from the IPv6 packets. IPv6 has a total of 128 bits. We divide the IP address into n segments, Y_0, Y_1, Y_2 ….Y_{n-1}. Each of these segments could have values from 0 to N-1 where $N = 2^{32}/n$. The Maximum Hop count could be up to 128 based on the type of OS running in the VM. For the cloud server, all the running nodes provide each segment's information for each hop value.

0	$A_{0,0}$	$A_{0,1}$	$A_{0,2}$	$A_{0,3}$	...	$A_{0,N-1}$
1	$A_{1,0}$	$A_{1,1}$	$A_{1,2}$	$A_{1,3}$		$A_{1,N-1}$
2	$A_{2,0}$	$A_{2,1}$	$A_{2,2}$	$A_{2,3}$		$A_{2,N-1}$
3	$A_{3,0}$	$A_{3,1}$	$A_{3,2}$	$A_{3,3}$		$A_{3,N-1}$
….	….	….	….	….		….
127	$A_{n-1,0}$	$A_{n-1,1}$	$A_{n-1,2}$	$A_{n-1,3}$		$A_{n-1,N-1}$

In every row one segment of a hop count is available. An accumulation of 128 segments represents the actual hop count of each IP address. Each entry in the table is represented by $A_{i,j}$

where the value of i ranges from 0 to N-1 (the maximum value is 31) and j ranges from 0 to n-1 (the maximum allowable value is 127). All the table entries hold two types of values: one is the current statistics *Present*[i,j] and the other is *Normal*[i,j]. If the cloud user's terminal is compromised by a DDoS, it adjusts the value of *Normal*[i,j] based on the *Present*[i,j], which is arranged by the incoming packet's source IP address coming from external terminals.

This automatic upgrade of hop values for each packet is hampered when there is an outside attacker. Segmentation of a source IP address starts after an attack is detected in the system. A spoof address will not have the coordinated values of each segment because the *Present*[i,j] and *Normal*[i,j] values are altered by the attack. Once a DDoS attack is detected by the Comparison module of FAPA and an alarm is generated, then all the incoming packets in the victim will be segmented in to 32 pieces according to the hop, so the values of each segment could vary. If $f(Y_i)$ is the function of current and normal statistics, then the score of each segment would vary as follows:

$$f(Y_i) = \begin{cases} Present[i,j] - Normal[i,j] & Present[i,j] > Normal[i,j] \\ \infty & Normal[i,j] = 0 \\ 0 & Present[i,j] < Normal[i,j] \end{cases}$$

Any segment Y_i with any kind of abnormality will be considered as an attack packet. To distinguish the attack packets from normal packets, we used the following norm as a metric by specifying the incoming packet Y as:

$$||F(Y)|| = \sum_{i=1}^{n-1} f(Yi)$$

The FAPA filtering compares the newly calculated ||F(Y)|| with a given threshold μ. If ||F(Y)||≥ μ*x then an attack is occurring, where the smaller the value of x is, the stronger the DDoS attack is commencing in the cloud system. The value of x is set globally based on the hop-count's packets. In order to determine the value of x, P and Q are the globally set parameters for each hop count m. The initial values of P_m and Q_m are set as the current profile and the ideal profile, respectively, when the hop count is equal to a given m, whereby $x = P_m/Q_m$. P and Q are set for each hop count m based on the node distances from the server, typically depending on the OS type running in the cloud VM. The hop count size varies based on the OS type. For example, a typical Windows OS has 32 bits to 64 bits, the MAC OS has 128 bits, etc. If x is 0, it will show that no legal packet has arrived before, so we consider the arriving packet with that hop-count as an attacking packet. If x ≥ 1 then the arriving packet is a legal packet out of the scope of the attack level regarding the hop count. Finally, the μ is calculated by taking the product of the specific segment and the maximum difference between the current and normal statistics as shown below:

$$\mu = n * \underset{i=0,j=0}{\text{MAX}} |Present[i,j] - Normal[i,j]|$$

As an example, if a packet with a non-legitimate IP address (never appeared before) arrives, then Present[i,j]>0 but *Normal*[i,j]=0 and $f(Y_i) = \infty$. As a result, $||F(Y)|| = \infty$, and it is considered as an attack packet. But in the case of a legitimate address, that packet will have a valid *Normal*[i,j] value, so $f(Y_i)$ won't be ∞. F(Y) is used to calculate the total segment values when a vulnerable packet has arrived in the source, and filtering takes place when value exceeds the threshold μ.

For any kind of external attacks our FAPA filtering will consider the hop count values of the IP address segments and the compromised packets' IP addresses will be conveyed to the rest of the nodes in the cloud through broadcast messages. This filtering will not be able to handle inside attackers. If the adversary fabricates an inside IP address, then the current and nominal statistics may not be able to highlight the difference significantly to assure the vulnerability of the packet. In the next section, we use the modified the filtering module to handle several external attacks from different adversaries.

6.3 DEPLOYMENT OF MODIFIED FAPA ON SAMPLE DATA

Our initial experiments in section 6.1 show that we have accomplished a moderate improvement in computational time by filtering. However, the number of packets we measured was not enough to draw a robust conclusion. In this section we describe an experiment we have conducted on sample data, which is a collection of millions of compromised packets.

In order to test our FAPA strategy, sample data was collected from CAIDA's 2010 DDoS Attack Dataset [19]. In this dataset, flooding was generated through ICMP packets. The dataset consisted of about 10 million echo requests and response packets for a server compromised over 24 hours. We calculated the echo requests and the echo responses, and we measured the differences and increments of the response packets per minute. Then we deployed FAPA with the CAIDA dataset, whereby the complete sample file was used in the experiment. The source IP addresses with the destination unreachable responses from the server were considered as

malicious users, and those IP packet requests were filtered by the FAPA comparison and validation modules prior to a response from the server.

The CAIDA sample data file has a *.pcap* extension, so it was opened in *Wireshark*, a Network protocol analyzer. Based on the *ip.dst*, the captured file was filtered which actually identifies all the ICMP packets destined to the victim. On average 350 packets/sec were delivered to the victim. Figure 32 illustrates the situation with the help of both ICMP and TCP packets. Considering the size of the captured file, only a significant part of the file is highlighted here. Within 120 seconds about 50k packets were forwarded to the victim of the network. In Figure 32 the red line shows TCP packets and the black line shows ICMP packets.

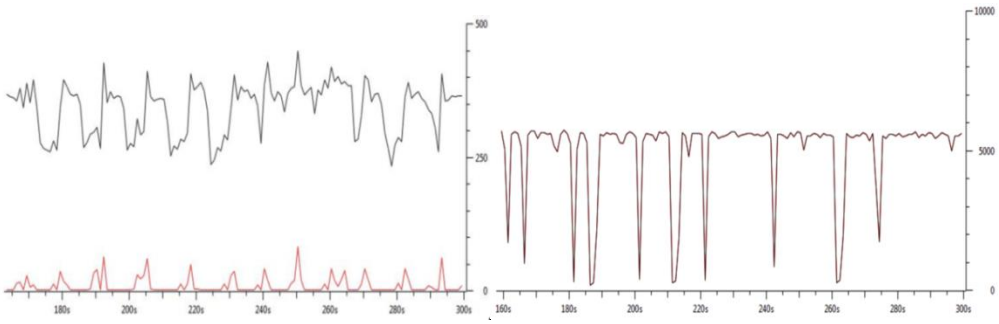


Figure 32: ICMP packets send towards victim

Figure 33: ICMP echo response from victim

The modified filtering, discussed in Section 6.2 was applied based on the ip.src (victim's IP address) for analyzing the reply packets from the victim. Figure 33 illustrates the situation after flooding has taken place. It was observed that on average, 4711.8 packets per second were delivered from the victim to anonymous IP addresses for acknowledgement. All these ICMP

echo requests were “Destination Unreachable” messages. According to Figure 33, at the time of the capture, about 700k echo responses were propagated from the victim within 120 seconds. Statistically, the response packets were 14 times of the total request packets within the first 120 seconds. After applying the modified FAPA filtering mechanism on the CAIDA captured file we observed significant improvement in filtering the DDoS ICMP packets.

Table III: Summary of the Improvement

Total Packet Received	9431735
ICMP Dest. Unreachable	270105
Bytes Captured	565702045
Bytes Compromised	23581699
Percentage of Compromised bytes	4%
Avg. Mbit/sec	82.8
Avg. Dest. Unreachable Mbit/sec	3.452

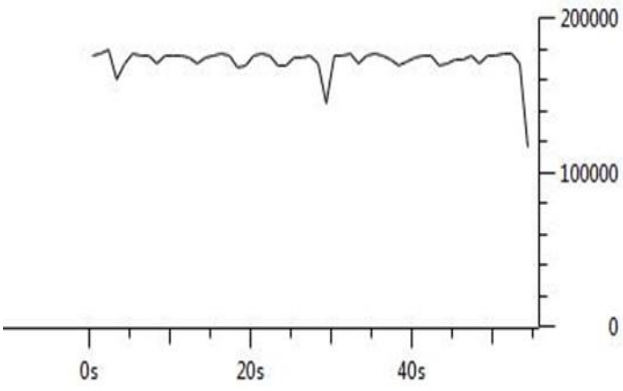


Figure 34: CAIDA Network Packets with Filtering

Table III illustrates that the percentage of compromised bytes decreased to 4% when the modified filtering was applied. The *Destination Unreachable* packets decreased to 3.4 per second, meaning 408 packets in first 120 seconds. The ratio between forwarded packets toward the victim and response packets from the victim is 1.16. So filtering has successfully blocked incoming spoof packets by 88%, and the response packets from the victim were decreased. Figure 34 highlights the improvements for a portion of the traffic from the captured file. Table III, enhances the visibility of the improvement achieved from the external spoof IP addresses.

Filtering in FAPA causes a drop in compromised packets from cloud. By employing the FAPA model on the client's side, users will have complete knowledge of any changes in traffic patterns. Cloud users do not have to rely on the providers' feedback, in case of an unusual phenomenon. In FAPA, a cloud user not only benefits from the whereabouts of traffic patterns but also can measure the computational time of the running instances. In the next section we show experimental results in which we leverage FAPA in a completely flooded environment, where DoS attacks are conveyed from various virtual instances. Here, virtual instances were launched from Amazon EC2.

6.4 DEPLOYMENT OF EC2 TO FLOOD PRIVATE CLOUD

In our previous experiments in Section 5.5, we studied the effect of flooding on the siblings of a compromised VM on the local cloud and also on the VM instances on a neighbor node, all within a small private local cloud. Next, we scale up our experiments by attacking a local cloud from a commercial cloud. Utilizing a commercial cloud allows us to increase the number of instances

generating the TCP flooding in the commercial cloud targeting the local cloud instances. We study the impact of TCP flooding on the targeted VM on the private cloud as well as on its neighbors.

6.4.1 METHODOLOGY

We used Amazon Web Services as the commercial cloud for our study. While there are various services provided by the Amazon Management Console, such as networking, database, storage, and app services, for this study we need light-weight compute nodes to be deployed as attackers. The nodes will direct a SYN packet attack towards a targeted machine. Amazon's EC2 (Elastic Compute Cloud) dashboard offers such type of instances. EC2 dashboard provides various launch configurations, such as Linux, Red Hat, Ubuntu and Windows in both 32 and 64-bit platforms. As our local cloud was built on an Ubuntu platform, we chose Ubuntu. The type of instance is a micro instance, called t1.micro, and provides a small amount of consistent CPU resources. It has 1 virtual core and 2MiB, and provides 2.4 GB per second of sequential reading as well as 2.6 GB per second of sequential writing [37]. To create a new instance, a key pair is downloaded onto the user's local machine from the remote cloud. This key pair will confirm the secure SSH connection between the remote server in which the instance is running and the local machine. We used the same key pair to launch local instances in our private cloud in order to establish the SSH connection with the remote Amazon instances. Flooding is then conducted from the Amazon instances towards the local VM instances.

We created several custom rules in the AWS instances for the communication between the AWS images and the local cloud virtual machines, which required changing some of the rules in the AWS default security groups. First, we changed the Custom TCP rule to allow TCP packets from all the available ports from 0 to 65568. We kept port 22 for SSH and deleted the Custom TCP rule that allows TCP through port 80 only. This was necessary because we use *iperf* to send legitimate TCP packets from the client, and the server will listen only in port 5001 by default if the server listening is using *iperf* as well. *Iperf* is a network testing tool that can generate TCP and UDP data streams. If AWS instances are sent only through port 80 then the server will not listen to any packets. It allows the user to select her specific port number for a TCP connection. Thus, the server can listen to a specific port for TCP packets. Also keeping the other ports open widens the attack vector when TCP flooding is generated from the commercial instances.

After the instances are launched, we used *putty generator* in the local terminal and the key pair file to convert it from a *.pkk* file provided by Amazon to a *.pem* file. First, the *putty generator* loads the *.pkk* file and we save the private key. In response it asks the user to save the key without a passphrase. We save it without a passphrase and the *.pkk* file is converted to a *.pem* file. Next, we use putty to open each of the AWS images in the local terminal. Here we used the instance identifier (IP address) and the newly converted *.pem* file to make the *SSH* connection with the local terminal. Then we installed our tools, such as Iperf, gcc, etc., needed to conduct the experiment.

In the private cloud, the physical nodes establish *SSH* communication with the running local virtual instances using the same key pair that was downloaded for the AWS instances in the local

terminal. *VMbuilder* allows launching an instance with a self-defining key pair, so before launching the local instances the images were stored in the *libvirt* directory provided by the *VMbuilder*. This directory is used as an image repository. The selected key pair was inserted into this image repository for each local VM with the batch file. When the batch files are triggered, *SSH* connection is established with the physical nodes on the private cloud via the key pair. Thus, the remote AWS images were able to send TCP-SYN packets to the local virtual machines. Throughout the rest of the study we will refer to the AWS instances as bots (handlers responsible for generating attack in a botnet) and the local instances in the private cloud as compromised VMs.

6.4.2 EXPERIMENTAL RESULTS

We conducted several experiments considering the impact on network bandwidth and transfer packets rates of the virtual instances.

6.4.2.1 Impact on the Local VM Bandwidth after Deployment of EC2 Bots

For this experiment, we began with one bot enabled with a TCP flooding program in order to target a local VM. The experimental setup is illustrated in Figure 35. Custom TCP rules were declared and *SSH* connections were established both ways: one from the remote bot to the local terminal and another from the local terminal to the soon to be compromised private cloud VM.

After observing the bandwidth in the local VM when one bot was deployed, the number of bots was increased in a progressive manner to observe the impact on bandwidth. Figure 35 illustrates the findings. IAM (Identity Access Management) refers to the images running in the cloud.

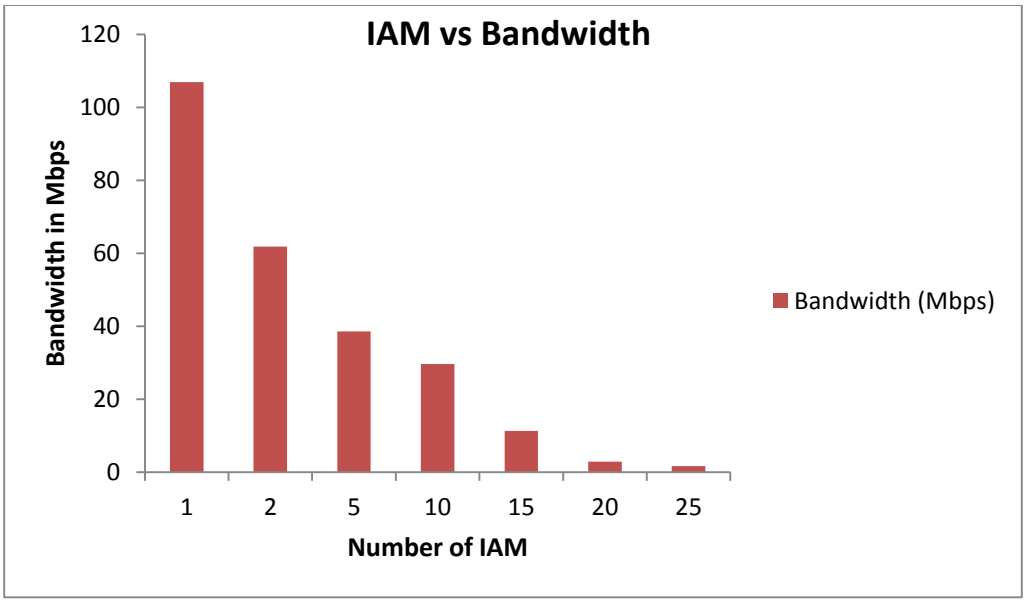


Figure 35: Bandwidth impact on the local VM

As shown in Chapter 5, after SYN flooding was conducted from 1 bot, the bandwidth of the compromised VM decreased about 45.1 Mbps from the bandwidth without flooding. Two bots were deployed, and then the number of bots was increased by 5 each successive time. On average the bandwidth decrement was 46.9%, compromising half the current bandwidth as the number of bots increased. After deploying 20 bots there was a 74% bandwidth drop in the local VM.

6.4.2.2 Impact on Local VM Transfer Packets after Deployment of EC2 Bots

In this experiment, the scenario and steps are similar to the previous experiment, but the network parameter for this study is the number of transfer packets. Figure 36 below illustrates the impact of flooding on the transfer packets on the compromised VM.

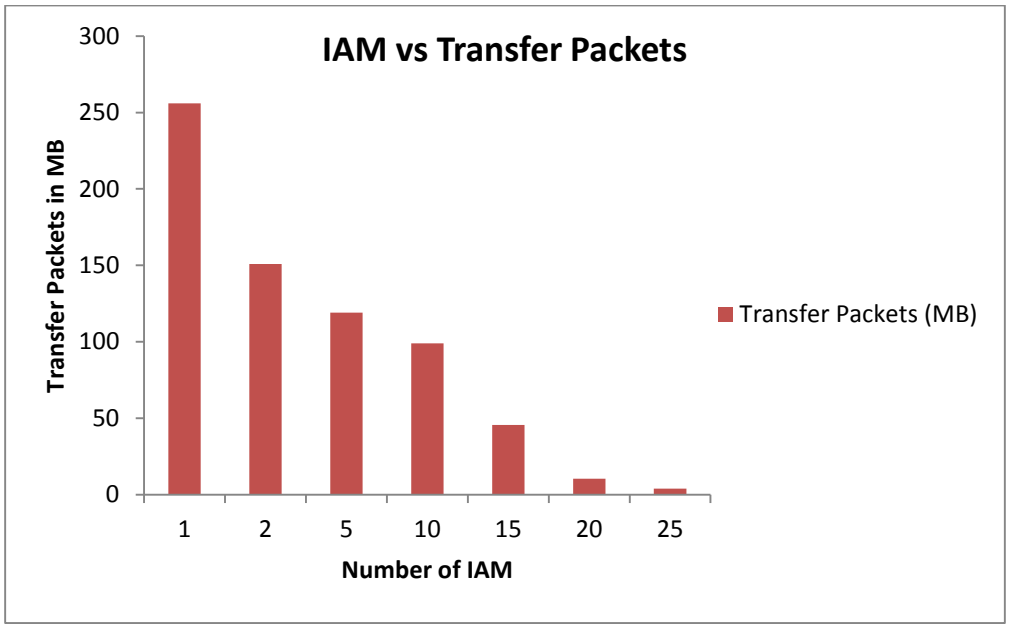


Figure 36: Transfer Packets impact on the local VM

We observed that after commencing the TCP flooding attack by 1 bot on the local instance, the transfer packets decreased by 105 MB, which is a 41.1% decrement. Again, we increased the number of bots in the botnet and as a result continued to compromise the number of transfer packets of the local VM. On average every attack decreased the number of transfer packets in the local VM about 45.5%. The deployment of 20 bots in the botnet resulted in an almost 77% decrement in transfer packets of the compromised VM.

6.4.2.3 Bandwidth Fluctuation in the Local Cloud VM

For this experiment, an Iperf server was installed in order to listen to all of the TCP packets that compromised the local instances in our Eucalyptus private cloud. As shown in section 6.3.2.1, the local private cloud VM was forced to validate the attack packets from the bots, and eventually, the bandwidth could not serve the resources for the legitimate packets. Iperf was set to listen every 2 seconds, and provided a maximum and minimum bandwidth value available for each attack. Figure 37 below shows the results.

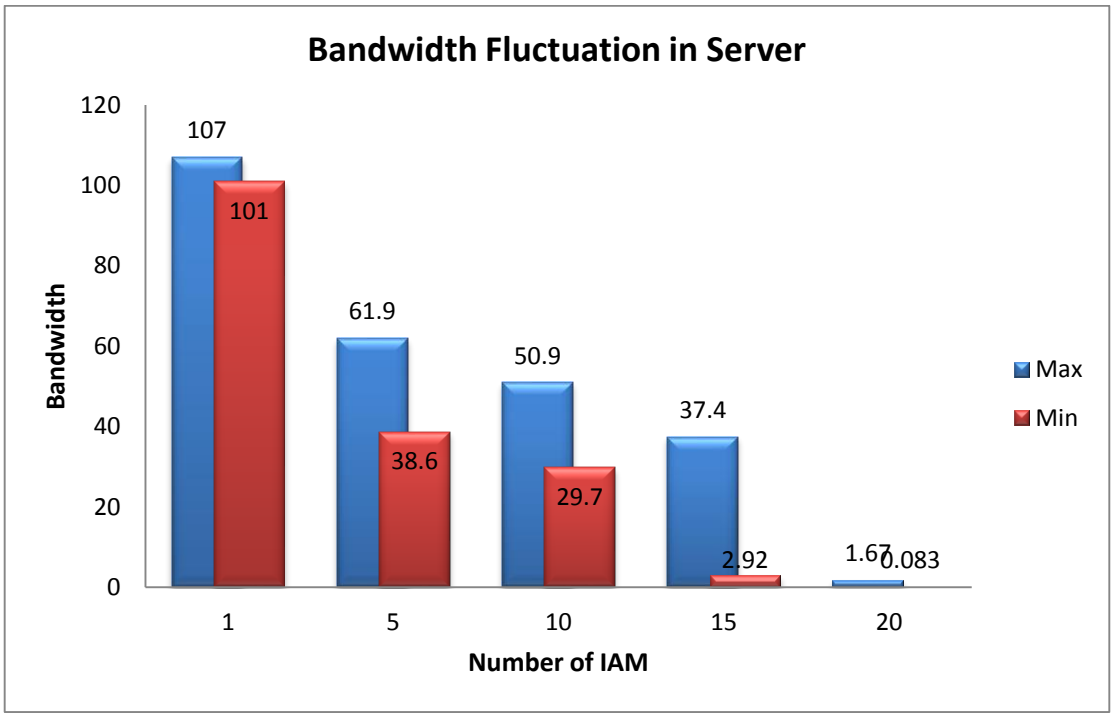


Figure 37: Bandwidth fluctuation in server end (local VM)

The first deployment of a bot resulted in a difference of about 6 Mbps between the max and min bandwidth. After the deployment of an additional 4 bots, the difference increased to 32.3 Mbps. As the number of bots increased progressively, the difference between the max and min

increased rapidly. After the deployment of 15 bots, the bandwidth fluctuation difference increased to 92%. On average the difference was 54%, which is significantly different than the first deployment, which was only 6%.

6.4.2.4 Transfer Packet Fluctuation in the Local Cloud VM

In this experiment *iperf* was again used in the local VM and flooding was conducted through the remote bots. Instead of measuring the bandwidth, we measured the number of transfer packets sent to the local VM and calculated the percentage of fluctuation with respect to the number of remote bots as illustrated in Figure 38.

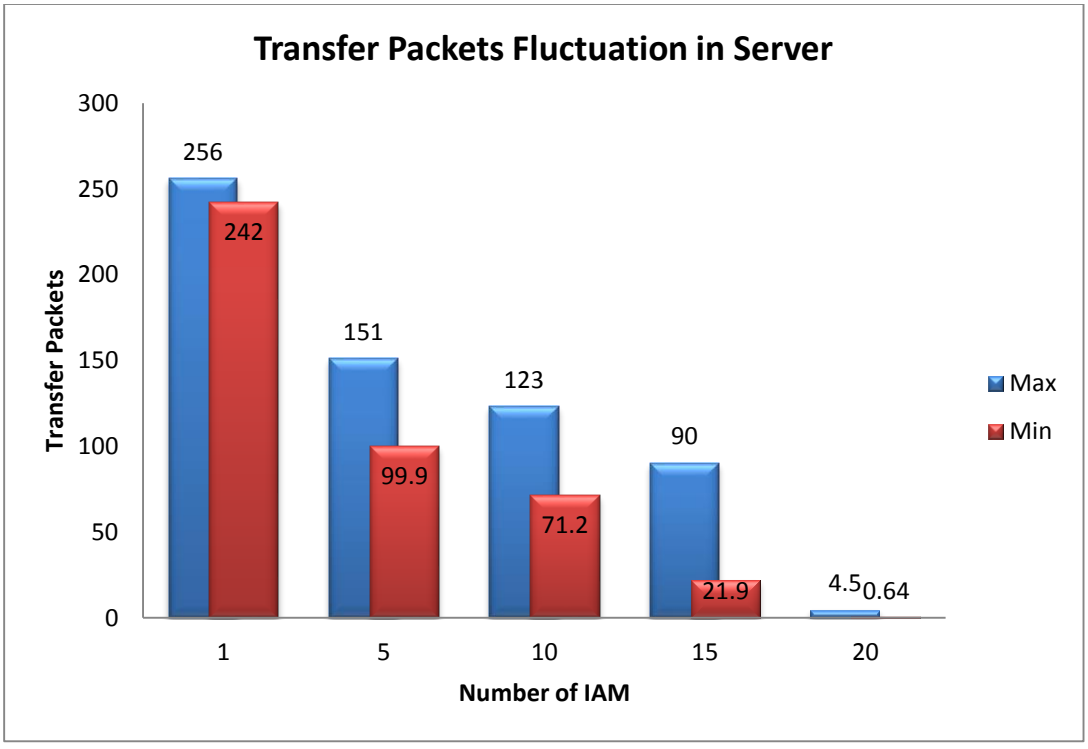


Figure 38: Transfer Packets fluctuation in server end (local VM)

With a single bot trying to flood the local VM, the difference between the maximum and the minimum number of transfer packets is about 14 MB. With 5 bots the difference increased dramatically to approximately 80 MB, which is 79.8%. The progression of bots in the botnet resulted in an increment in the maximum difference of 88.33% for 20 bots. On average the number of transfer packet fluctuation is 50.24%. Also notable, the increment in bots resulted in about a 40 MB increment in transfer packet fluctuation on average for the local VM.

6.4.2.5 Bandwidth Fluctuation in AWS Instances

The above experiment was conducted in the client's end within the virtually created botnet inside AWS. For this experiment an Iperf client was installed in the bots and the bots were engaged in compromising the private cloud. The Iperf client sent legitimate TCP packets to the target instance, and every 2 seconds calculated its own bandwidth in terms of Mbps. At the same time, a SYN flooding program was running in the background and sending packets to the target VM instance. Thus, the non-legitimate packets were clogging the allocated bandwidth of the client. Figure 39 illustrates the maximum and minimum bandwidth values of the client during the time of an attack.

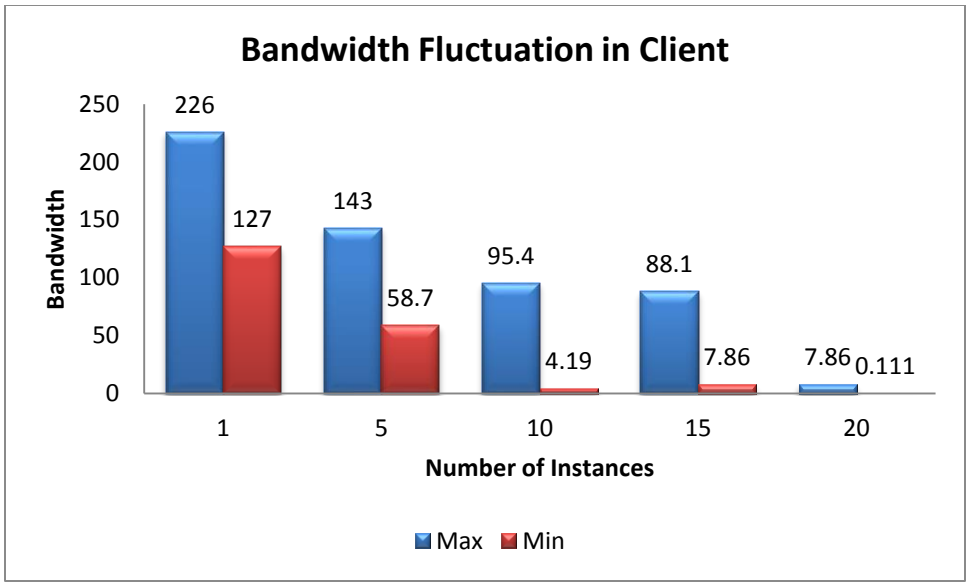


Figure 39: Bandwidth Fluctuation within AWS

With a single bot, the bandwidth difference was 99 (44%). With the addition of new bots, the difference in bandwidth fluctuated but the percentile of difference increased from 43.8% to 98.6%. On average the difference was about 73 Mbps. These results demonstrate that flooding not only compromises the bandwidth in the server end but in the client end as well. Also, this study indicated that bandwidth fluctuation in the client could be used as a monitoring parameter in the cloud system to protect users from inside threats.

6.4.2.6 Transfer Packet Fluctuation in AWS Instances

A similar experiment was conducted in order to observe the change in transfer packets in the client's end. With the Iperf client deployed in the bots, the amount of transfer packets was calculated at an interval of 2 seconds with a testing time of 20 seconds.

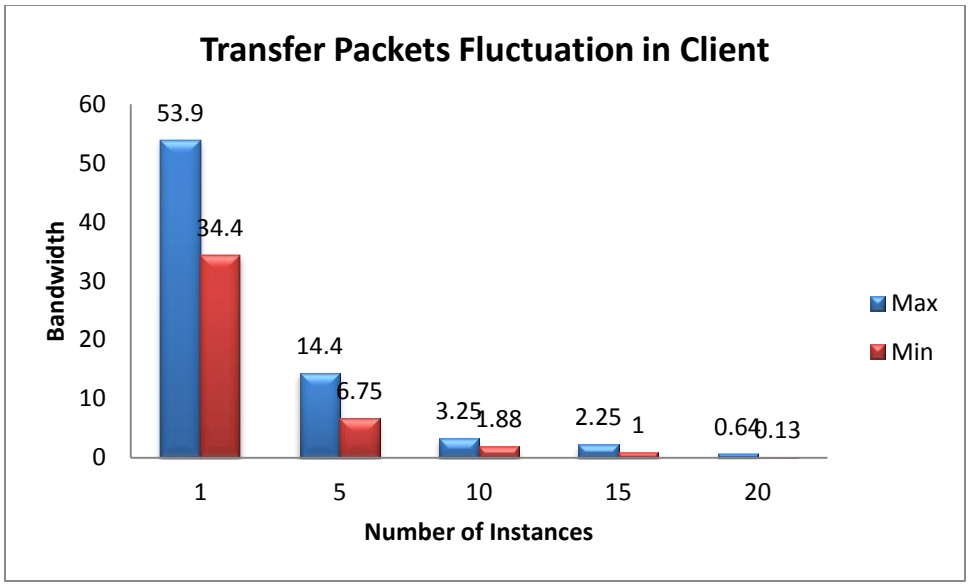


Figure 40: Transfer Packets Fluctuation within AWS

At the beginning, the bot was sending a maximum of about 54 MB of packets and a minimum of about 34 MB. As illustrated in Figure 40, the difference in transfer packets between the max and min parameter was approximately 36%. Later the difference increased to almost 80%. On average the transfer packets fluctuated about 53.3%. The mean transfer fluctuation was about 6.1 MB, which could also be utilized as a point of observation while monitoring a cloud system for any kind of intrusion or compromised node.

6.4.2.7 Impact on Neighbor's Bandwidth

To perform this experiment, we utilize two different physical nodes containing VM instances, referred to as NC1 and NC2. NC1 has 3 VMs and NC2 has 4 VMs. We installed Iperf in all the VMs on physical node NC1 and NC2 on the private cloud. The SYN flooding was conducted from a botnet inside AWS and each of the bots targeted a VM existing in the second physical node NC2 of the private cloud. Thus, the Iperf server was listening to legitimate TCP packets

coming from the AWS micro instance (Iperf client) as well as being flooded by non-legitimate TCP packets from the same micro instance (flooding program). We again increased the number of bots in a progressive manner to observe the impact on the bandwidth of neighbor VMs. Figure 41 illustrates the impact on bandwidth. We refer to VMs on NC2 as sibling nodes of the compromised VM, and those VMs on NC1 as neighbor VMs.

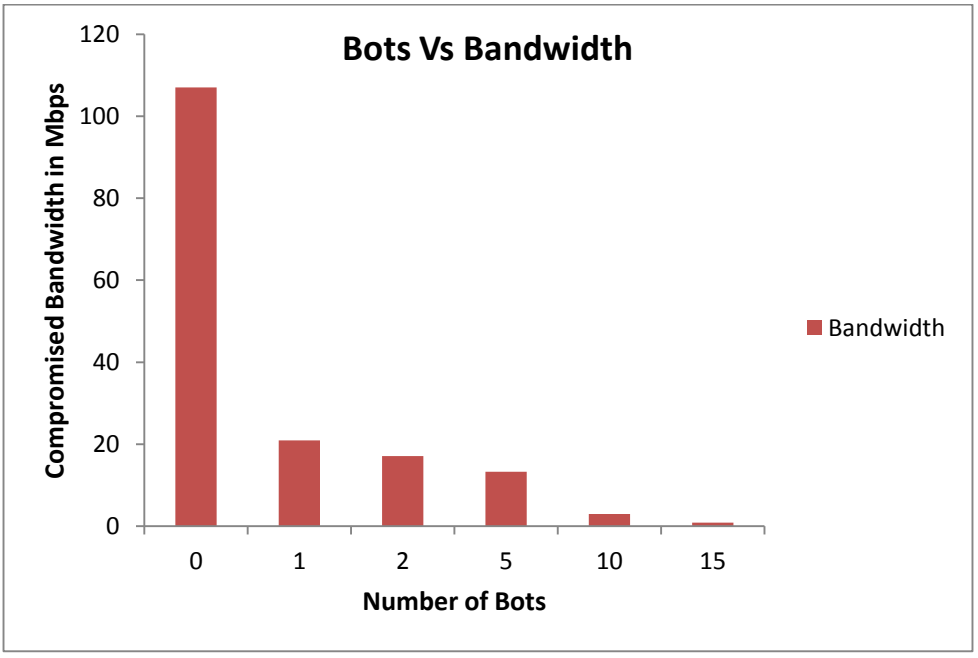


Figure 41: Impact on neighbors' bandwidth

When the targeted VM was flooded from a single bot, almost 80% bandwidth of the neighbor VMs was immediately consumed. The bandwidth decreased from 107 Mbps to 21 Mbps after the first attack. We then increased the number of bots in order to observe the impact on the neighbor VMs. During the course of flooding with additional bots, the bandwidth of the neighbor VMs decreased about 21.2 Mbps. About 54% of its bandwidth was compromised during the attack. Also, after the deployment of 15 bots the bandwidth decreased to only 651 Kbps, whereas the

neighbor VMs had 107 Mbps of bandwidth before flooding. One more observation was that the bandwidth decrement was not as rapid/sharp after the deployment of the first bot. Hence, it can be inferred that most of the resources are consumed at the beginning of the attack. Unfortunately, this occurs when the system is completely unaware of any intrusions.

6.4.2.8 Impact on Neighbor's Transfer Packets

This experiment was conducted in the same manner as the previous experiment but we measured the impact on the number of neighbor machine transfer packets instead of measuring the bandwidth as illustrated in Figure 42. In this experiment we again increased the number of bots to observe the impact of flooding on the neighbor VMs in the private cloud.

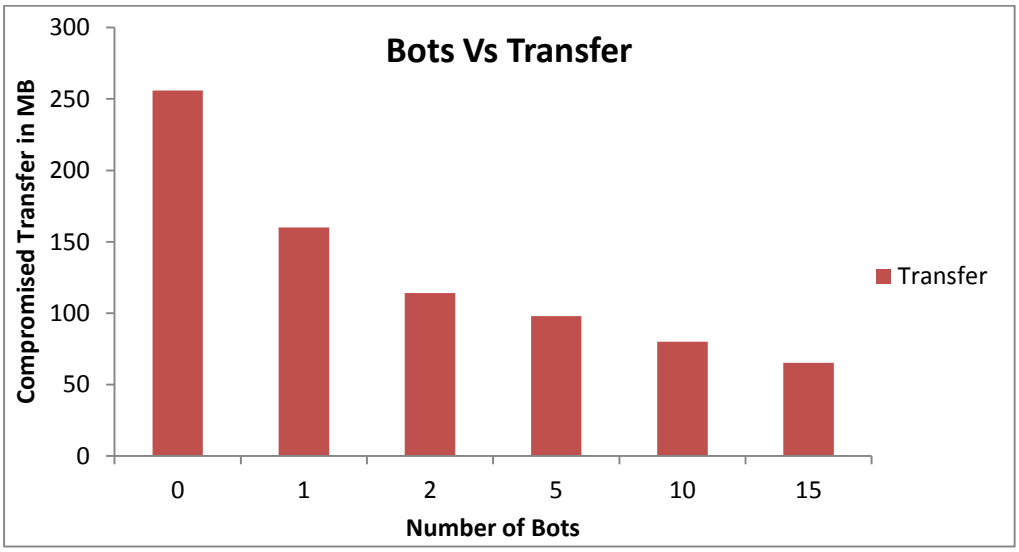


Figure 42: Impact on neighbors' transfer packets

Before flooding, the VMs were transferring 256 MB every 20 sec. Once the first bot was deployed and flooding took place, the transfer rate shrank to 160 MB, and the neighbor VMs lost about 96 MB of data. There was approximately a 38% decrement in transfers. As bots were added, the transfer rate continued to decrease, and on average there was about a 24% decrement in the number of transfer packets. Similar to the previous bandwidth experiment, the transfer decrement was not as rapid as during the first deployment of the bot. On average the transfer dropped about 24%, but the first bot alone caused about a 38% drop.

6.5 DOS ATTACK TOOLS INTERVENTION

In the previous section we observed the impact of a DoS flooding attack that we generated ourselves from the virtual instances from a commercial cloud, with respect to bandwidth and transfer packets. In this section we utilize several popular DoS attack tools to study their impact on local VM instances with respect to TX and RX rate, transmitted and received packets, and GBytes received and transmitted. Lastly, we deploy FAPA filtering to observe the improvement in network parameters during flooding. After the first deployment of a bot, since FAPA is a reactive approach, we anticipate FAPA will response immediately after the first intrusion.

6.5.1 ENVIRONMENTAL SETUP FOR DOS ATTACK TOOLS

The DoS attack tools we used are LOIC (Low Orbit Ion Cannon) and XOIC (Extreme Orbit Ion Cannon). LOIC is an open source DoS attack tool that can be used to check server vulnerability. An admin can easily conduct a stress test on the server by providing the IP address, specific

packet type for flooding generation, port number and also a timer. LOIC can also take a URL as its parameter, by which one can compromise a web server through HTTP flooding. XOIC is a new DoS attack tool that only requires an IP address and port number to launch an attack, but it can be used to send messages to a target machine as well. Both of these DoS attack applications were used on the local virtual machines in the private cloud to check the vulnerability of our local cloud. We then deployed FAPA filtering to resolve the flooding issues.

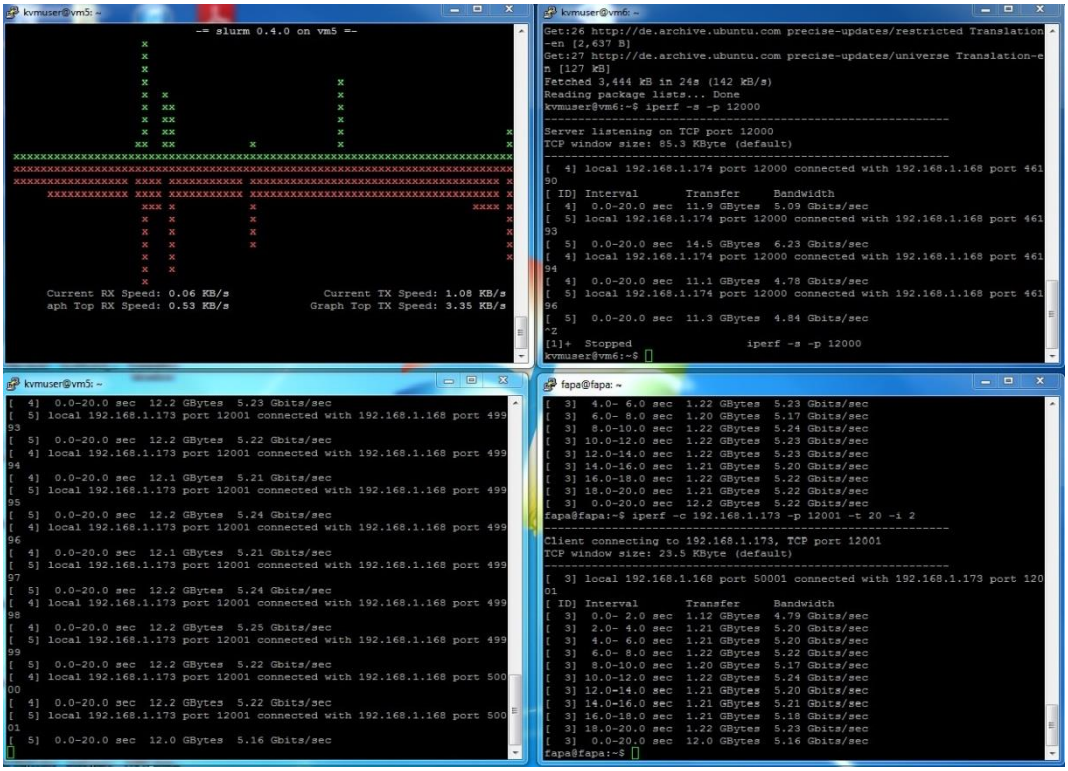


Figure 43: Different Scenarios with LOIC

As illustrated in the Figure 43, the experiment was conducted in the following five different scenarios:

1. Measuring the traffic scenario without any workload on a VM

2. Creating a workload by deploying *Iperf* to send legitimate packets from a VM to the target VM
3. Injecting a DoS attack from a Windows7 terminal with LOIC
4. Injecting a DoS attack from a Windows7 terminal with XOIC
5. Deploying FAPA filtering to intervene in the DoS attacks.

Traffic measurement was conducted by *slurm*, a tool suitable for Ubuntu machines. Through *slurm*, we were able to measure the top speed of the transfer rate TX and receive rate RX, the number of packets transmitted and received, and the number of bytes transmitted and received. For the workload, *Iperf* sent TCP packets through a specific port in 2 second intervals for 20 seconds each time. The average was considered for analysis of the traffic scenario under different circumstances.

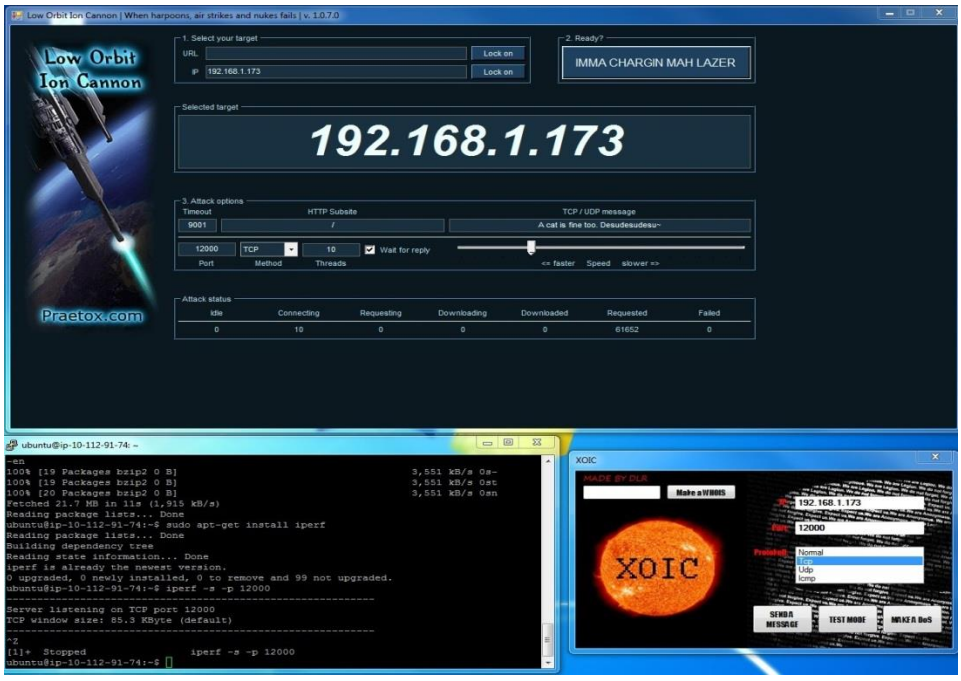


Figure 44: User Interface of LOIC and XOIC

As can be seen from Figure 44, we used port 12000 for TCP flooding and our timeout was 9001 milliseconds with 10 threads. Threading LOIC accomplishes an amplification of spoof requests from its connections to the target server. In another words, once flooding overflows the server with enough spoof data, the victim will struggle to serve legitimate requests. Figure 44 shows this occurred when LOIC made 10 connections, meaning the adversary acquired 10 ports of the network interface for continued TCP flooding.

6.5.2 EXPERIMENTAL RESULTS WITH LOIC AND XOIC

6.5.2.1 DoS Impact on Speed

In this experiment, we observed the impact of a DoS attack with respect to the RX and TX speeds of a virtual machine. The virtual machine was attending to a workload requested from another VM. Statistical results for the five scenarios are presented, which includes the results from FAPA filtering. An analysis is provided after Figure 45.

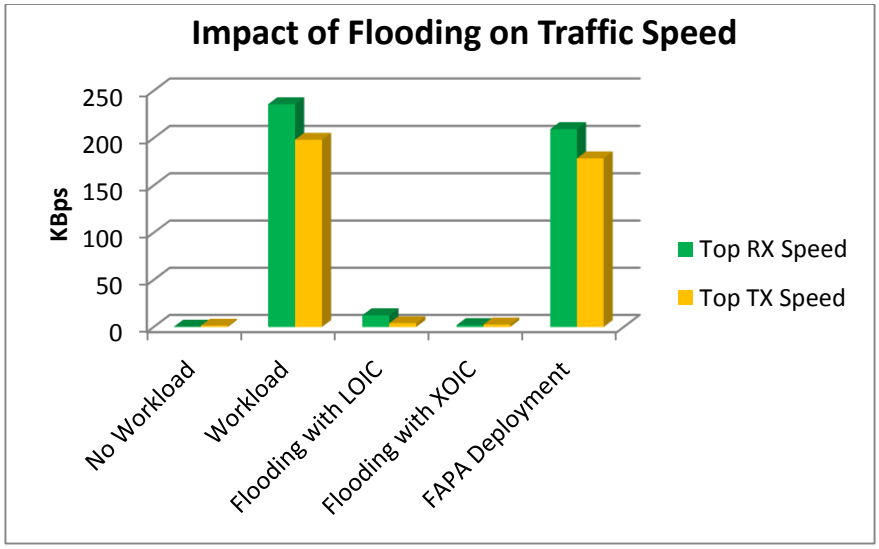


Figure 45: DoS impact on traffic speed

Analysis-

1. In the initial phase the TX and RX top speeds were very low, 1.43 and 0.42 KBps, respectively. This is expected, because without any workload or any data processing the VM was idle.
2. Once the workload was sent by *Iperf* in 2 second intervals from another VM to the target VM, the RX and TX top speeds increased to 235 and 198 KBps, respectively.
3. LOIC was deployed to conduct TCP flooding on the target VM for 5 minutes. The RX speed dropped dramatically by 95% and TX by 98%. The LOIC attack was terminated. The dramatic nature of the decrease was surprising.
4. XOIC was deployed to interrupt the RX and TX speeds, are impacted by the DoS attack generated from LOIC and XOIC.
5. Finally, we deployed FAPA and recovered about 74% of RX speed and 81% of TX speed, meaning 26% of RX speed and 19% of TX speed remained compromised during this operation.

6.5.2.2 DoS Impact on Amounts of Packets

In this experiment, we measured the number of packets received and transmitted in GBps as we observed the impact of a DoS attack. Similar to the previous experiment, all five circumstances were considered and an analysis is provided below in Figure 46.

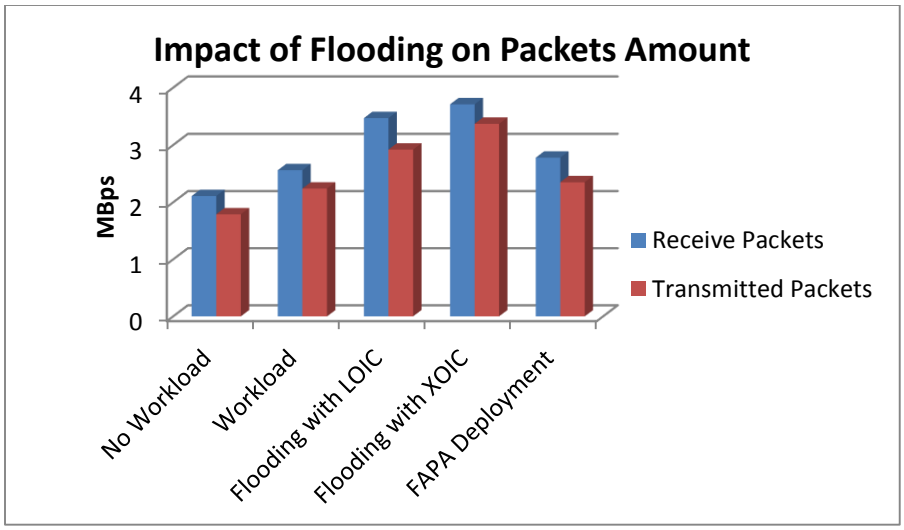


Figure 46: DoS impact on amounts of packets

Analysis-

1. Without any workload the VM was receiving about 2.11 MBps and transmitting about 1.8 MBps. This was computed as the average over 10 readings without any workload. We assumed this to be the usual cloud traffic due to message passing among the running VMs and the communication bridge between eth0 and vnet1 (the virtual network interface).
2. After the workload was injected from *Iperf* the receive amount increased about 21% and transmitted packets by 25%. This is reasonable because *Iperf* was deploying packets in KBps in a 2 second interval with a timeout of 20 sec.
3. LOIC was deployed to commence TCP flooding. The receive packets increased by 65% and transmitted packets were increased by 63%.
4. The LOIC attack was terminated and we began the XOIC DoS attack on the VM. The flooding attack reduced the receive packets by 75% and transmitted packets by 88%.

5. FAPA filtering recovered about 91% of the receive packets and 95% of the transmitted packets. Overall, there was a complete loss of 5% of transmitted packets and 8% of receive packets.

6.5.2.3 DoS Impact on GB Transmitted

In this experiment, we measured the Giga Bytes transmitted with respect to all four scenarios by *slurm*. In the analysis section of this experiment we will denote the amplification by times rather than percentage for a better understanding.

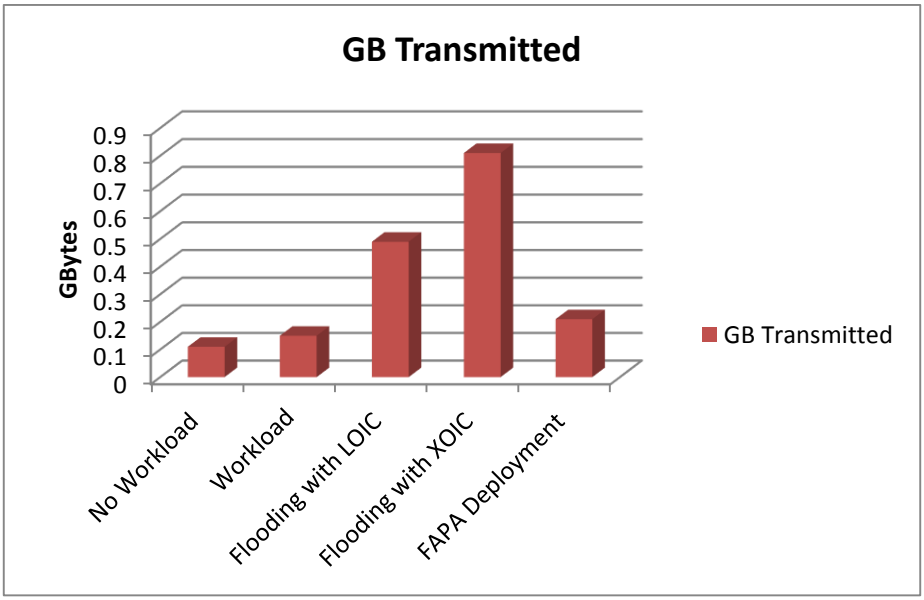


Figure 47: Impact of DoS on GB transmission

Analysis-

1. Without any workload the VM was transmitting about 0.1 GBytes as illustrated in Figure 47.

2. Once Iperf sent a legitimate workload to the VM, the response was about 0.15 GBytes, which is only 1.5 times higher compared to no workload.
3. After LOIC deployment, the VM was transmitting about 4.5 times more GBytes inside the cloud system. This is a huge overload with respect to a single VM in the network.
4. XOIC deployment amplified the transmission by 7.4 times. This means a compromised cloud with 100 nodes could amplify the transmission by more than 50 times and cause the system to collapse in a relatively short period of time.
5. FAPA deployment managed to filter about 60% of the non-legitimate transmissions.

Hypothetically, if FAPA is deployed in all the running VMs of the cloud network, then amplification of a DoS attack will be curtailed by 3/5th.

6.5.2.4 DoS Impact on GB Received

This experiment is similar to the previous one, except we now observe the DoS impact on Giga Bytes received. Results appear in Figure 48.

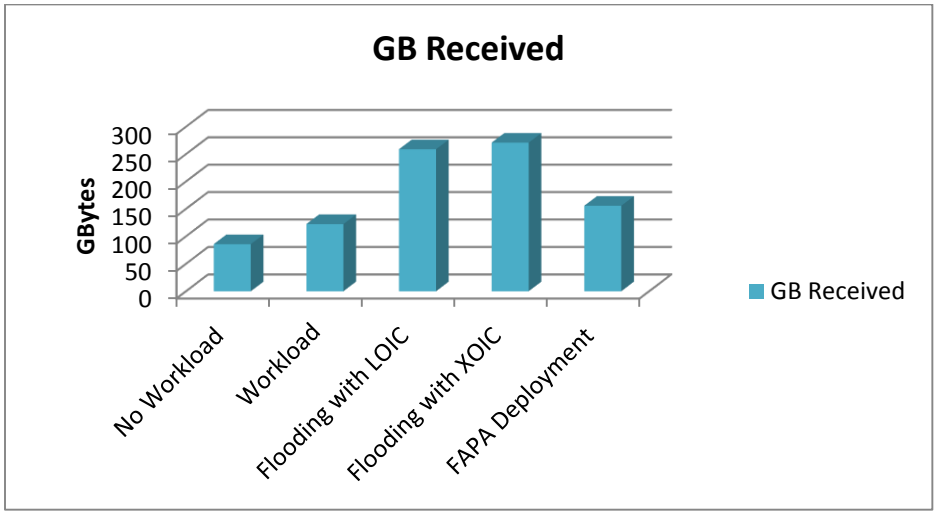


Figure 48: Impact of DoS on GB received

Analysis-

1. In the *No-Workload* scenario the target VM was receiving about 85.7 GBytes of data. We considered this GB amount as a general case after computing the average over several readings. Later we observed a rapid increment after flooding.
2. With the workload from Iperf, the target VM started to receive 122 GBytes of data, which is only 1.4 times the data it was receiving initially.
3. After the TCP DOS was deployed by LOIC, the victim VM received about 3 times of GB data, a more than 100% increment in traffic in the receiving port of the victim VM.
4. During the XOIC attack, VM was receiving about 3.2 times of GB data, which is more than a 120% traffic increment in the receiving port of the target VM. This overhead traffic can cause the legitimate requests starve to death.
5. Finally with the deployment of FAPA, 73% of the legitimate data was recovered with a 27% loss in GBytes received.

The results from the experiments involving the EC2 instances, the LOIC and XOIC, and the sibling and neighbor instances demonstrated that FAPA was able to protect the victim and neighbors. However, some of the illegal packets may have escaped FAPA and entered the system (false positives) or some legal packets may have been rejected by FAPA (false negatives). In Chapter 7, experiments are presented that measure the false positive and false negative rates of FAPA with respect to the victim machine and its neighbors.

Chapter 7

PERFORMANCE ANALYSIS

In the experiments we conducted for this chapter, our goal was to obtain the false positive and false negative rates of FAPA. FAPA retrieves legal packets but may erroneously identify a legal packet as an illegal packet (false negative) or may identify illegal packets as legal packets (false positive). Our experiments were conducted for two different scenarios.

1. Measure the false positive and false negative rates on the target virtual instance on which the flooding attack will be conducted.
2. Measure the false positive rate on the neighbor virtual machines (VMs). Due to the scalable nature of clouds, the colocated VMs could also be affected by the flooding attack.

7.1 EXPERIMENT 1: *FALSE POSITIVE RATES ON TARGET VM*

For this experiment we used an external terminal to conduct flooding attacks on a virtual machine, identified as VM5, located within the physical node, identified as NC1. Flooding was conducted every 5 minutes and we took about 7 samples from 1 minute to 60 minutes. The steps we followed to measure the false positive rates were:

1. Total incoming packets were captured in each interval. The TCP packets identified as malicious contained the IP address of the external terminal and the IP address of the virtual machine VM5. We call these packets illegal packets, mal injection or spoof packets in this study.

2. In each interval FAPA's filtering process was deployed in the target machine VM5 to identify those that are spoof packets. We then measured the number of the output packets from VM5 which were allowed to pass through the system because they were not identified as spoof packets. We also call output packets unprocessed packets or unhandled packets.
3. We subtracted the unprocessed packets from the number illegal packets and calculated the total number of packets processed by FAPA. Processed packets are those which were correctly identified as spoof packets by FAPA's filtering process.

Processed packets = Illegal packets – Unprocessed packets

4. Thus, we determined the false positive rate of FAPA in each interval.

False positive rate = Unprocessed packets / Total Illegal packets

Figure 49 below shows the results of our experiments.

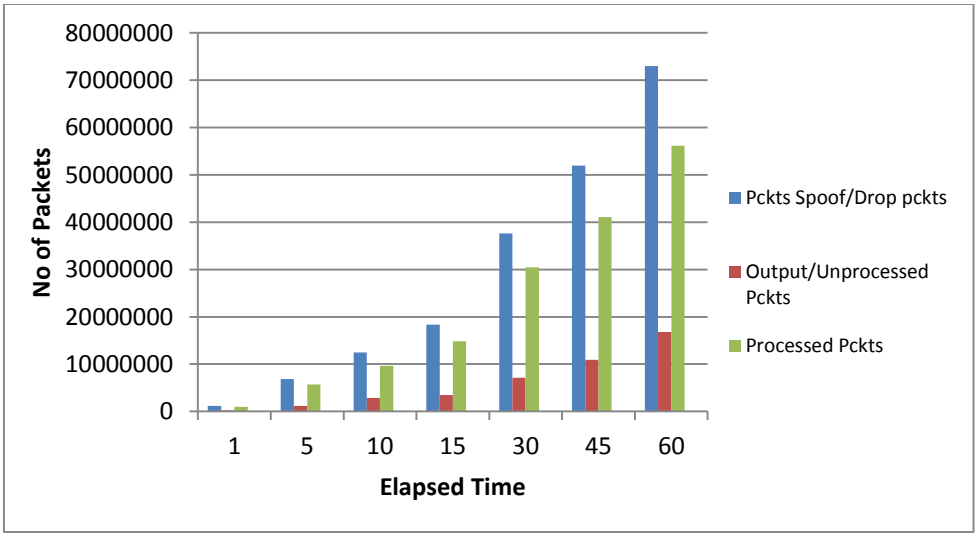


Figure 49: Processed and Unprocessed illegal Traffic by FAPA

Analysis-

1. In Figure 49 the blue bars represent the total number of illegal packets. In each interval the number of illegal packets increased linearly.
2. The red bars represent the number of unprocessed illegal packets, which were not filtered out by FAPA and were allowed to enter the system. In the first several intervals the unprocessed packets increased 3 to 4 times compared with the previous count. Later intervals did not increase more than 1.5 times of the previous count. This result demonstrates FAPA improves in its ability handle illegal traffic packets.

The green bars represent the number of processed illegal packets which were not allowed to enter the system. These results prove the ability of FAPA to identify about 81% of the illegal packets on average in each interval.

Figure 50 shows the false positive rates in each interval for the victim VM5.

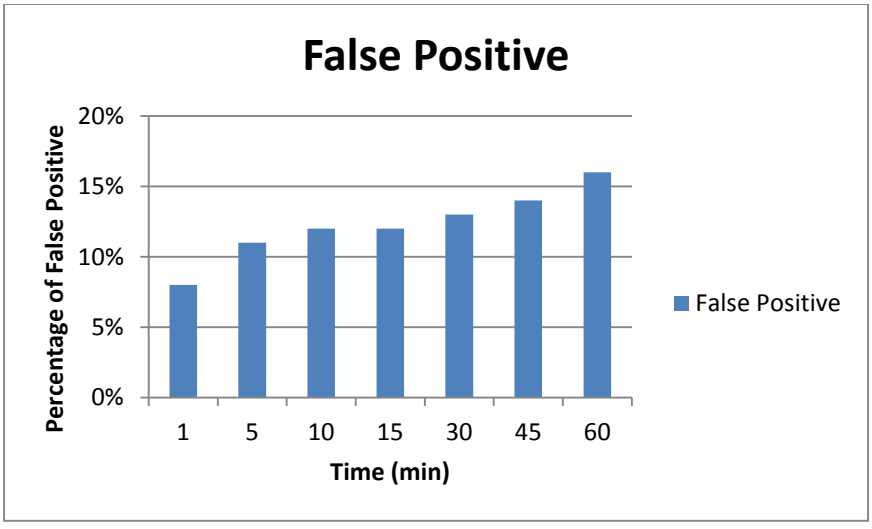


Figure 50: False Positive rates on victim VM5

Analysis:

1. As illustrated in Figure 50, FAPA on average allowed about 12% of illegal packets to enter the system, resulting in a false positive rate on the targeted virtual machine of around 12%.
2. The maximum false positive rate was 16% during an interval of 60 minutes. About 73 million spoof packets were sent to target machine VM5. The total number of spoofed traffic packets was in the billions, demonstrating FAPA's ability even under heavy loads.

7.2 EXPERIMENT 1: *FALSE NEGATIVE RATES ON TARGET VM*

As in the previous experiment we conducted flooding on the target machine VM5 from an external terminal and captured traffic on the Ethernet interface. The steps we followed for this experiment are:

1. We considered the total number of packets, including both legal and illegal traffic.
2. We filtered the illegal packets from the total captured by collecting the IP addresses of both the target machine (VM5) and the attacker (external terminal).
3. We calculated the legal packets by subtracting the mal injection packets from the total number of captured packets.

$$Legal\ Packets = Total\ Captured - Mal\ Injection$$

4. For this experiment, we included a bit field in the IP packet, where the ACK=FALSE and SYN=FALSE. This condition allowed identification of the legal packets which did not receive any acknowledgements and eventually were rejected by FAPA in VM5. We call these packets, Rejected Legal Packets.

5. We computed the false negative rate of FAPA using the following formula:

False negative rate = Rejected Legal packets / Total Legal packets

Figure 51 below illustrates the results of the experiments.

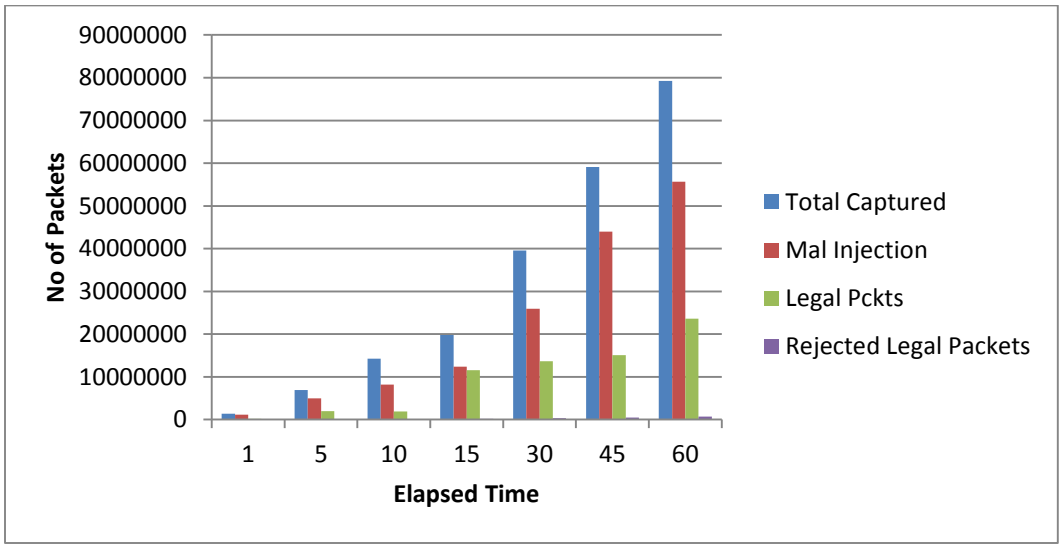


Figure 51: Processed and Unprocessed Legal Traffic by FAPA

Analysis:

1. In Figure 51, the blue bars represent the total number of packets in each interval, which increase exponentially due to the number of spoof TCP packets trying to enter the target VM5.
2. The red bars represent the number of mal injection packets in each interval.
3. The green bar is the legal packets. These legal packets were allowed to enter the system and were processed by the virtual machine under the flooding condition. On average about 15% of the total captured packets were legal packets allowed to enter the cloud system.
4. The average percentage of rejected legal packets was only about 0.9% of the total packets and is represented by the violet bars. Due to their significantly small number, these bars can

be hardly seen in Figure 51. FAPA is successful in distinguishing the legal messages from the illegal packets while filtering.

Figure 52 illustrates the false negative rates of FAPA.

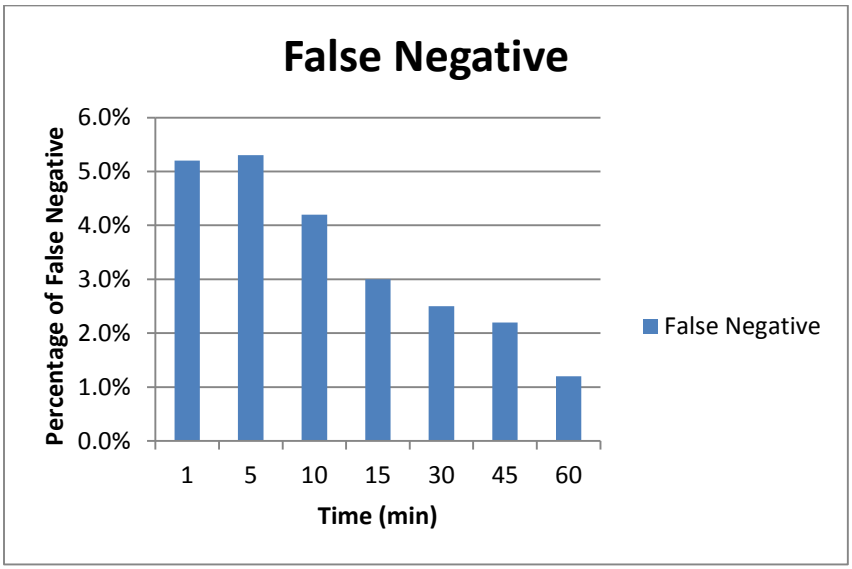


Figure 52: False Negative rates on victim VM5

Analysis:

1. In Figure 52, the average false negative rate of FAPA is 3.54%.
2. The largest false negative rates occurred during at first minute, about 5%. At later times, FAPA was more successful in identifying the legal instances from the illegal instances as the rejection of legal packets dropped below 2%.
3. Hence, we assume that when a flooding event begins FAPA might face some difficulties in identifying the legal instances, but as flooding progresses it can recover most of the legal requests of the cloud users.

From our results, we observe that when protecting a private cloud, FAPA might allow some illegal instances to enter the system but will provide a very low false negative rate.

7.3 EXPERIMENT 2: *FALSE POSITIVE RATES ON NEIGHBOR VMS*

The goal of this experiment is to determine the false positive rates in the neighbor nodes. To be more specific, by neighbor we mean we considered the traffic loads of the colocated virtual machines with the victim. In physical node NC1, the flooded virtual machine was VM5 and the neighbors we considered for measuring traffic are VM6, VM7 and VM8. The steps we followed are:

1. First we measured the number of packets coming from the victim to the neighbor *vnets* (virtual networks).
2. We then captured the number of unprocessed packets which could not be identified as malicious by FAPA in each of the neighbor nodes.
3. Finally we compute the false positive rates in these neighbor nodes:

$$\textit{False Positive Rate} = \textit{Packets Sent} / \textit{Total Illegal Packets}$$

For this experiment we studied the traffic for each of the neighbors at five intervals at 5, 10, 15, 30 and 60 minutes. In Figure 53 the results are illustrated.

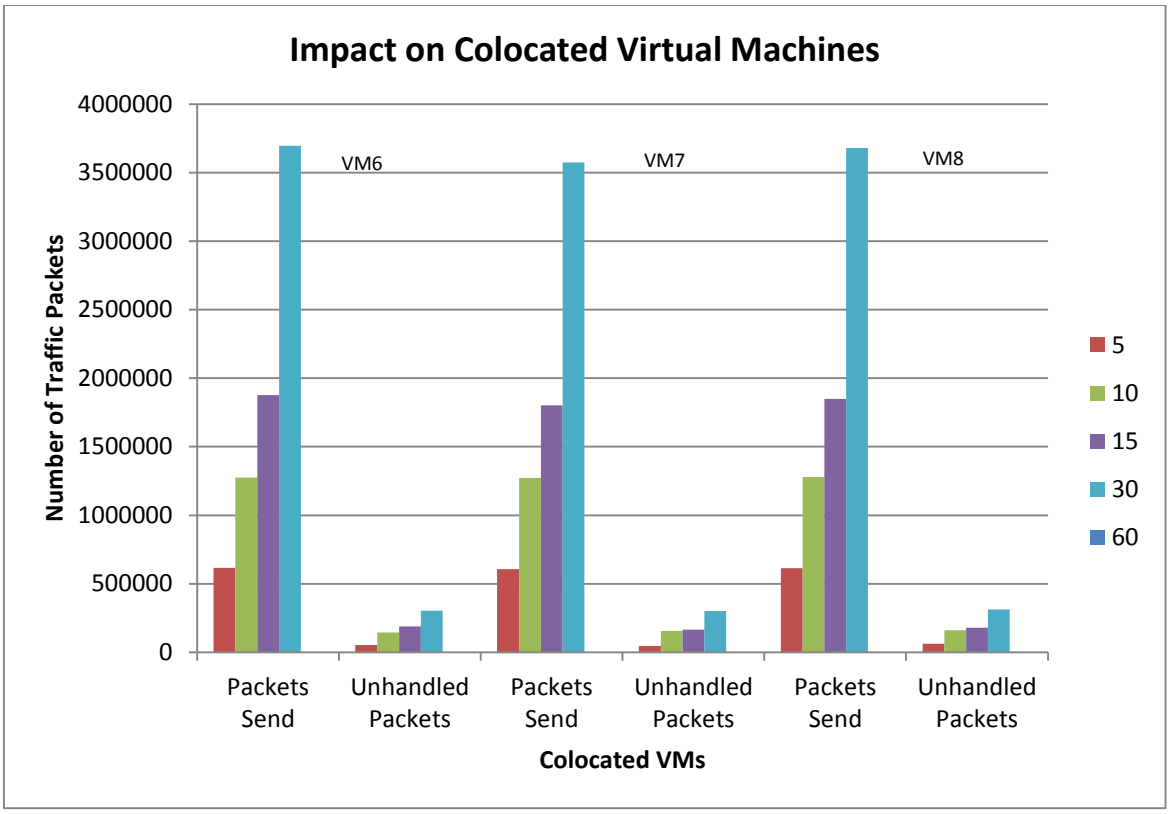


Figure 53: Processed and Unprocessed illegal Traffic in neighbor virtual machines

Analysis:

1. In Figure 53, the X axis displays the number of sent packets and unhandled packets at the 5 time intervals for VM5's three neighbors VM6, VM7, and VM8.
2. At each interval the number of packets sent to the neighbor VMs from the targeted victim VM5 increased exponentially as expected. This is due to the elastic nature of a cloud, whereby the victim tries to offload some of the TCP requests to its neighbors.
3. The trend for unhandled packets in each neighbor node was not exponential, but rather linear. This means that FAPA did not let a large number of illegal packets compromise the neighbor virtual instances. Even when flooding was conducted for an hour, FAPA only allowed 0.5% of the mal injection packets into the cloud system.

4. On average FAPA let only 0.6% of total illegal packets enter the neighbor nodes. These results demonstrate that FAPA has protected the private cloud from flooding.

In Figure 54 the rate of false positives in the neighbor VMs is shown. We emphasize that the false positive rates are with respect to the traffic coming from the victim machine and not the actual external terminal from which the attack was generated. We listed the average of the five intervals for each of the neighbor virtual machine instances.

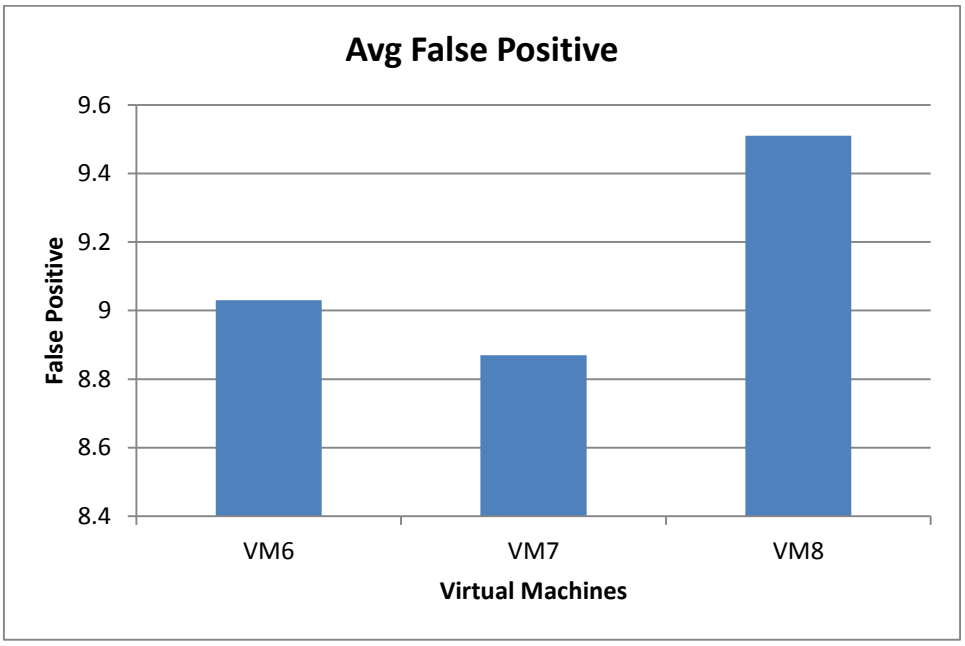


Figure 54: False Positive Rates of neighbor virtual machines

Analysis:

1. The average false positive rate is 9.13%.
2. The maximum false positive rate is 9.51% in virtual machine VM8. Although FAPA performed the worst for VM8 compared to the other two VMs, two things must be

mentioned. Firstly, all the virtual instances showed false positive rates less than 10%. Secondly, when all of the spoofed traffic from the external machine is considered, the average for the false positive rate was below 1%.

FAPA handled the flooding scenario by successfully blocking more than 88% of the illegal packets, as demonstrated by our analysis of the false positive rate. FAPA also allowed only a very small number of illegal packets to compromise the neighbor nodes by blocking most of the illegal packets, as shown by the very small false positive rates for neighbors. These results have confirmed that FAPA is suitable for a Service Oriented Architecture (SOA) which emphasizes providing highly uninterrupted services to its customers.

Chapter 8

FUTURE DIRECTIONS

Although FAPA performed satisfactorily under a DoS attack, we anticipate that it can still be improved. Hence, our intension for the future is to involve the Hypervisor, and if possible, the migration of compromised virtual machines. Also, detection against TCP flooding as achieved by FAPA is a limited solution for clouds, because an adversary can penetrate the system through other types of network packets, such as UDP, HTTP, and fake ICMP echo requests.

In general, a cloud hypervisor is responsible for scheduling the resources in the virtual instances based on the frequency of requests from the clients, the kernel requirements from the guest OS running on top of the virtual machines or be some scheduled reallocation of resources. Because it is only responsible for scheduling, most of a hypervisor's the resources remain underutilized.

In future we are planning on utilizing these resources to do additional monitoring for the running VMs. The advantages of this additional monitoring will be to inspect the vulnerable VMs for spoof packets before the packets reach FAPA filtering. There has been some work involving VM introspection by VM library with ESX platform of VMware cloud. Also there has been some work involving a Graphic Turing test by throwing an alphanumeric image as a challenge to authenticate the legitimacy of a cloud user. But none of these consider the impact of a DoS attack on colocated virtual machines or on neighbor machines that are located in a remote physical node. So if we can engage the hypervisor to detect the adversary packets before the victim VM becomes completely compromised, then we can proclaim a proactive approach by FAPA. Such

modification could be a landmark for the cloud security research endeavor. Also, FAPA will have less overhead when securing the virtual instances.

Another approach is the migration of a compromised VM to another server if it has enough resources to handle the workload of the migrating VM. Most of the hypervisors store the images of its existing virtual machines in a secure location; for example, KVM has *libvirt* as its image repository. This repository can be easily accessed by the hypervisor, and it has all the necessary information of resource scheduling among the servers. If the compromised VM can be offloaded to the closest available server, then a considerable amount of network bandwidth can be secured from the DoS attack. In addition, it should reduce the overhead cost of FAPA by mitigating some of its workload for the handling of compromised traffic from the victim machine.

UDP (User Datagram Protocol) packets are audio data carrier and HTTP packets are contained with URL addresses. If a cloud user is suffering from UDP flooding, then storing or performing analysis of the audio files will take long time. Unnecessary UDP packets will consume a lot of CPU time, causing the instances of a cloud user to starve. HTTP (Hypertext Transfer Protocol) is an application protocol deployed for the distribution, retrieval, and collaboration of data with servers for the *World Wide Web*. There are some methods defined in HTTP to retrieve information about the server states but do not change anything in the server. Other methods like POST, PUT, and DELETE are intended for actions that make changes in the server end, such as financial transactions (commonly known as e-commerce). If an adversary gets control over the HTTP packets and starts sending HTTP packets with identical POST requests multiple times, then the server is affected by unnecessary replies. Also TRACE, TRACK and DEBUG methods

are considered potentially insecure as attackers can utilize them to retrieve information or bypass security controls during attacks.

In the future we would also like to conduct flooding with UDP and HTTP packets in our private cloud at first, then in the EC2. The goal is to try to reduce the effect of UDP or HTTP flooding (also known as X-DOS or H-DOS) by deploying FAPA in the compromised nodes. UDP packets should be comparatively easy to retrieve because the packets have Source and Destination ports. These port numbers will be compared to the assigned port numbers in the flooding source. For HTTP flooding, we can retrieve bits of the PUT, TRACE, and TRACK, GET fields as mentioned earlier. If any field causes the status change in the server's end, then DoS can be detected.

Involvement of a hypervisor in terms of monitoring VMs and migration of compromised VMs could enhance the security assurance of a private cloud to a great extent. Also, if we consider the most recent business trends (merging of startup companies), collaboration among small-scale private clouds could benefit from resource sharing among their existing hypervisors with these additional features, and ensuring more security on their clients' virtual machines.

Chapter 9

CONCLUSION

A cloud is vulnerable to numerous types and approaches to flooding attacks. We have proposed a model (FAPA) to detect and eliminate DoS attacks. By considering different types of DoS attacks, our goal was to make the cloud more dynamic and adaptive. After conducting a number of experiments, we were able to identify some distinguishing features of traffic patterns in a cloud environment under the influence of flooding. These features were incorporated into our FAPA model to help protect against DoS attacks.

We have implemented a prototype of our FAPA model. Results using this prototype have shown the ability to detect spoof packets and eliminate those packets in a progressive manner. Elimination of spoof packets in this progressive manner provides the cloud user an additional advantage. FAPA is employed on the client's side, so a user is able to keep track of filtering and can reboot her desired instances after the completion of filtering. Also, disputes between providers and customers can be resolved with evidence of a traffic log.

Results have demonstrated the ability of our proposed FAPA model to detect and filter packets to eliminate a DoS attack. All the experiments were conducted based on TCP flooding, and filtering was used to detect and eliminate a DoS attack. Also, we explored the impact of TCP flooding on a compromised virtual machine as well as its siblings and neighbors in a cloud environment. FAPA was able to filter almost 88% of the spoofed TCP packets in a cluster environment, and it also successfully mitigated compromised ICMP echo responses in the CAIDA example. FAPA

handled the ratio between forward and reverse packets as well as increased the original IP packets by decrementing the spoofed IP packets.

Detection against TCP flooding from only one VM is not enough for modern clouds, because an adversary can flood the system from TCP-SYN packets through a botnet. An adversary can also penetrate the system through UDP packets, HTTP packets, and fake ICMP echo requests. Hence, we deployed FAPA to apprehend compromised packets launched from a commercially used public cloud, EC2. Before the involvement of FAPA filtering we were also able to acknowledge the amplification of the DoS impact on private cloud instances with respect to colocated and remote VMs. The performance analysis of FAPA confirmed that the DoS impact on the victim as well as its neighbors was eliminated successfully. Finally, a study of the false positive and false negative rates demonstrated that FAPA successfully blocked more than 88% of the illegal packets, and allowed only a very small number of illegal packets to compromise the neighbor nodes by blocking most of the illegal packets.

FAPA could be beneficial to cloud users in terms of increasing the adaptability of cloud use and deducting additional costs. Cloud providers would not need to buy expensive tools for the elimination of DoS or DDoS, and can invest more money on providing services for the cloud customers. Also, if the consumer could build mutual trust among the locally launched virtual instances, then FAPA could be deployed in the border routers of the local domains. This would allow FAPA to become even more economical and effective than deploying it in all the virtual machines. Customer satisfaction as well as business goals will be achieved through transparent traffic logging and secure services by the private cloud as provided by FAPA.

REFERENCES

[1] Lonea, Alina Mădălina, Daniela Elena Popescu, Octavian Prostean, and Huaglory Tianfield. "Evaluation of Experiments on Detecting Distributed Denial of Service (DDoS) Attacks in Eucalyptus Private Cloud." In *Soft Computing Applications*, pp. 367-379. Springer Berlin Heidelberg, 2013.

[2] Jin, Cheng, Haining Wang, and Kang G. Shin. "Hop-count filtering: an effective defense against spoofed DDoS traffic." In *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 30-41. ACM, 2003.

[3] Bremler-Barr, Anat, and Hanoch Levy. "Spoofing prevention method." InINFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, vol. 1, pp. 536-547. IEEE, 2005.

[4] Duan, Zhenhai, Xin Yuan, and Jaideep Chandrashekar. "Constructing Inter-Domain Packet Filters to Control IP Spoofing Based on BGP Updates." In*INFOCOM*. 2006.

[5] Mühlbauer, Wolfgang, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. "Building an AS-topology model that captures route diversity." In*ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 195-206. ACM, 2006.

[6] CA, CERT Advisory. "21 TCP SYN flooding and IP spoofing attacks."*Disponiblea http://www. cert. org/advisories/CA-1996-21. html* (1996).

[7] Cheswick, William R., Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet security: repelling the wily hacker*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[8] Ferguson, Paul. "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing." (2000).

[9] Liu, Xin, Xiaowei Yang, David Wetherall, and Thomas Anderson. "Efficient and secure source authentication with packet passports." In *USENIX SRUTI*. 2006.

[10] Yaar, Abraham, Adrian Perrig, and Dawn Song. "StackPi: New packet marking and filtering mechanisms for DDoS and IP spoofing defense." *Selected Areas in Communications, IEEE Journal on* 24, no. 10 (2006): 1853-1863.

[11] Park, Kihong, and Heejo Lee. "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets." In *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 15-26. ACM, 2001.

[12] Moore, David, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. "Internet quarantine: Requirements for containing self-propagating code." In*INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, pp. 1901-1910. IEEE, 2003.

[13] Zhang, Beichuan, Raymond Liu, Daniel Massey, and Lixia Zhang. "Collecting the Internet AS-level topology." *ACM SIGCOMM Computer Communication Review* 35, no. 1 (2005): 53-61.

[14] Oliveira, Ricardo V., Dan Pei, Walter Willinger, Beichuan Zhang, and Lixia Zhang. "In search of the elusive ground truth: the internet's as-level connectivity structure." In *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1, pp. 217-228. ACM, 2008.

[15] Xu, Jun, Jinliang Fan, Mostafa H. Ammar, and Sue B. Moon. "Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme." In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pp. 280-289. IEEE, 2002.

[16] Gao, Lixin. "On inferring autonomous system relationships in the Internet."*IEEE/ACM Transactions on Networking (ToN)* 9, no. 6 (2001): 733-745.

[17] Beverly, Robert, and S. Bauer. "ANA Spoofer Project." (2006).

[18] Kawamoto, D. "DNS recursion leads to nastier DoS attacks." *ZDNet. co. uk* 17 (2006).

[19] Claffy, K., Dan Andersen, and Paul Hick. "The CAIDA anonymized 2010 internet traces." *2011-01-28)[2011-06-20]. http://www. caida. org/data/pas-sive/passive_2011 dataset. xml* (2013).

[20] Lee, Heejo, Minjin Kwon, Geoffrey Hasker, and Adrian Perrig. "BASE: An incrementally deployable mechanism for viable IP spoofing prevention." In*Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pp. 20-31. ACM, 2007.

[21] Li, Jun, Jelena Mirkovic, Mengqiu Wang, Peter Reiher, and Lixia Zhang. "SAVE: Source address validity enforcement protocol." In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1557-1566. IEEE, 2002.

[22] WorkingGroup, WIDE MAWI. "Mawi working group traffic archive." (2013).

[23] Paxson, Vern. "An analysis of using reflectors for distributed denial-of-service attacks." *ACM SIGCOMM Computer Communication Review* 31, no. 3 (2001): 38-47.

[24] Shannon, Colleen, David Moore, and Emile Aben. "The CAIDA backscatter-2008 dataset." (2008).

[25] Snoeren, Alex C., Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. "Single-packet IP traceback." *IEEE/ACM Transactions on Networking (ToN)* 10, no. 6 (2002): 721-734.

[26] Subramanian, Lakshminarayanan, Volker Roth, Ion Stoica, Scott Shenker, and Randy Katz. "Listen and whisper: Security mechanisms for BGP." In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*. 2004.

[27] Wang, Feng, and Lixin Gao. "On inferring and characterizing internet routing policies." In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pp. 15-26. ACM, 2003.

[28] Weigle, Michele C., Prashanth Adurthi, Félix Hernández-Campos, Kevin Jeffay, and F. Donelson Smith. "Tmix: a tool for generating realistic TCP application workloads in ns-2." *ACM SIGCOMM Computer Communication Review* 36, no. 3 (2006): 65-76.

[29] Yang, Xiaowei, David Wetherall, and Thomas Anderson. "A DoS-limiting network architecture." *ACM SIGCOMM Computer Communication Review* 35, no. 4 (2005): 241-252.

[30] Gupta, B. B., Ramesh C. Joshi, and Manoj Misra. "Dynamic and auto responsive solution for distributed denial-of-service attacks detection in ISP network." *arXiv preprint arXiv:1204.5592* (2012).

[31] Gil, Thomer M., and Massimiliano Poletto. "MULTOPS: a data-structure for bandwidth attack detection." In *USENIX Security Symposium*. 2001.

[32] Lee, Wenke, Salvatore J. Stolfo, and Kui W. Mok. "A data mining framework for building intrusion detection models." In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pp. 120-132. IEEE, 1999.

[33] Floyd, Sally, Steve Bellovin, John Ioannidis, Kireeti Kompella, Ratul Mahajan, and Vern Paxson. *Pushback messages for controlling aggregates in the network*. Internet Draft, work in progress, 2001.

[34] Hwang, Kai, Min Cai, Ying Chen, and Min Qin. "Hybrid intrusion detection with weighted signature generation over anomalous internet episodes." *Dependable and Secure Computing, IEEE Transactions on* 4, no. 1 (2007): 41-55.

[35] Zunnurhain, Kazi, and Susan V. Vrbsky. "Security in cloud computing." In*Proceedings of the 2011 International Conference on Security & Management*. 2011.

[36] Bornkessel, Daniel, Nathaniel Friedman, Garrett LeSage, and Cornelius Schumacher. "System and method for managing a virtual appliance lifecycle." U.S. Patent Application 12/476,144, filed June 1, 2009.

[37] Amazon, E. C. "Amazon elastic compute cloud (Amazon EC2)." *Amazon Elastic Compute Cloud (Amazon EC2)* (2010).

[38] Jensen, Meiko, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. "On technical security issues in cloud computing." In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pp. 109-116. IEEE, 2009.

[39] Gruschka, Nils, and Luigi Lo Iacono. "Vulnerable cloud: Soap message security validation revisited." In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pp. 625-631. IEEE, 2009.

[40] A Vouk, Mladen. "Cloud computing–issues, research and implementations."*CIT. Journal of Computing and Information Technology* 16, no. 4 (2008): 235-246.

[41] Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee et al. "A view of cloud computing."*Communications of the ACM* 53, no. 4 (2010): 50-58.

[42] Christodorescu, Mihai, Reiner Sailer, Douglas Lee Schales, Daniele Sgandurra, and Diego Zamboni. "Cloud security is not (just) virtualization security: a short paper." In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 97-102. ACM, 2009.

[43] Kim, Taehyun, Yeongrak Choi, Seunghee Han, Jae Yoon Chung, Jonghwan Hyun, Jian Li, and JW-K. Hong. "Monitoring and detecting abnormal behavior in mobile cloud infrastructure." In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pp. 1303-1310. IEEE, 2012.

[44] Hall, Mark, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. "The WEKA data mining software: an update."*ACM SIGKDD explorations newsletter* 11, no. 1 (2009): 10-18.

[45] Breiman, Leo. "Random forests." *Machine learning* 45, no. 1 (2001): 5-32.

[46] A Vouk, Mladen. "Cloud computing–issues, research and implementations." *CIT. Journal of Computing and Information Technology* 16, no. 4 (2008): 235-246.

[47] Gruschka, Nils, and Luigi Lo Iacono. "Vulnerable cloud: Soap message security validation revisited." In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pp. 625-631. IEEE, 2009.

[48] Accetta, Mike, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young. "Mach: A new kernel foundation for UNIX development." (1986): 93-112.

[49] Wentzlaff, David, Charles Gruenwald III, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, and Anant Agarwal. "A unified operating system for clouds and manycore: fos." *Proceedings of the 1* (2009).

[50] Garfinkel, Tal, and Mendel Rosenblum. "A Virtual Machine Introspection Based Architecture for Intrusion Detection." In *NDSS*. 2003.

[51] Litty, Lionel, and David Lie. "Manitou: a layer-below approach to fighting malware." In *Proceedings of the 1st workshop on Architectural and system support for improving software dependability*, pp. 6-11. ACM, 2006.

[52] Payne, Bryan D., Martim Carbone, Monirul Sharif, and Wenke Lee. "Lares: An architecture for secure active monitoring using virtualization." In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pp. 233-247. IEEE, 2008.

[53] Liu, Huan. "A new form of DOS attack in a cloud and its avoidance mechanism." In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pp. 65-76. ACM, 2010.

[54] Headquarters, Americas. "Cisco Data Center Infrastructure 2.5 Design Guide." (2007).

[55] Douligeris, Christos, and Aikaterini Mitrokotsa. "DDoS attacks and defense mechanisms: classification and state-of-the-art." *Computer Networks* 44, no. 5 (2004): 643-666.

[56] Ferguson, Paul. "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing." (2000).

[57] Kim, Taehyun, Yeongrak Choi, Seunghee Han, Jae Yoon Chung, Jonghwan Hyun, Jian Li, and JW-K. Hong. "Monitoring and detecting abnormal behavior in mobile cloud infrastructure." In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pp. 1303-1310. IEEE, 2012.

[58] Tariq, Usman, Yasir Malik, and Bessam Abdulrazak. "Defense and Monitoring Model for Distributed Denial of Service Attacks." *Procedia Computer Science* 10 (2012): 1052-1056.

[59] Livingston, Frederick. "Implementation of Breiman's random forest machine learning algorithm." *ECE591Q Machine Learning Journal Paper* (2005).

[60] Doukopoulos, Xenofon G., and George V. Moustakides. "The fast data projection method for stable subspace tracking." In *Proceedings 13th European Signal Processing Conference, EUSIPCO 2005*. 2005.

[61] Khorshed, Md Tanzim, A. B. M. Ali, and Saleh A. Wasimi. "Classifying different denial-of-service attacks in cloud computing using rule-based learning." *Security and Communication Networks* 5, no. 11 (2012): 1235-1247.

[62] Gens, Frank. "IT cloud services user survey, pt. 2: Top benefits & challenges."*IDC eXchange* (2008).

[63] Mansfield-Devine, Steve. "DDoS: threats and mitigation." *Network Security*2011, no. 12 (2011): 5-12.

[64] Bedi, Harkeerat Singh, and Sajjan Shiva. "Securing cloud infrastructure against co-resident DoS attacks using game theoretic defense mechanisms." In*Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pp. 463-469. ACM, 2012.

[65] Habib, Irfan. "Virtualization with kvm." *Linux Journal* 2008, no. 166 (2008): 8.

[66] Zunnurhain, Kazi, and S. Vrbsky. "Security attacks and solutions in clouds." In *Proceedings of the 1st international conference on cloud computing*, pp. 145-156. 2010.

[67] Zunnurhain, Kazi, and Susan V. Vrbsky. "Security in cloud computing." In *Proceedings of the 2011 International Conference on Security & Management*. 2011.

[68] Roschke, Sebastian, Feng Cheng, and Christoph Meinel. "Intrusion detection in the cloud." In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pp. 729-734. IEEE, 2009.

[70] Yi, Fasheng, Shui Yu, Wanlei Zhou, Jing Hai, and Alessio Bonti. "Source-based filtering scheme against DDOS attacks." *International Journal Database of Theory and Application* (2008).

[71] Kodada, Basappa B., Gaurav Prasad, and Alwyn R. Pais. "Protection Against DDoS and Data Modification Attack in Computational Grid Cluster Environment." *International Journal of Computer Network & Information Security*4, no. 7 (2012).

[72] Rahmani, Hamza, Nabil Sahli, and Farouk Kamoun. "DDoS flooding attack detection scheme based on F-divergence." *Computer Communications* 35, no. 11 (2012): 1380-1391.

[73] Hwang, Kai, Sameer Kulkareni, and Yue Hu. "Cloud security with virtualized defense and reputation-based trust mangement." In *Dependable, Autonomic and Secure Computing, 2009. DASC'09. Eighth IEEE International Conference on*, pp. 717-722. IEEE, 2009.

[74] Bakshi, Aman, and B. Yogesh. "Securing cloud from ddos attacks using intrusion detection system in virtual machine." In *Communication Software and Networks, 2010. ICCSN'10. Second International Conference on*, pp. 260-264. IEEE, 2010.

[75] Sabahi, Farzad. "Virtualization-level security in cloud computing." In*Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, pp. 250-254. IEEE, 2011.

[76] Schwarzkopf, Roland, Matthias Schmidt, Christian Strack, Simon Martin, and Bernd Freisleben. "Increasing virtual machine security in cloud environments."*Journal of Cloud Computing* 1, no. 1 (2012): 1-12.

[77] Chonka, Ashley, Yang Xiang, Wanlei Zhou, and Alessio Bonti. "Cloud security defence to protect cloud computing against HTTP-DoS and XML-DoS attacks."*Journal of Network and Computer Applications* 34, no. 4 (2011): 1097-1107.

[78] Wen, Fu, and Li Xiang. "The study on data security in Cloud Computing based on Virtualization." In *IT in Medicine and Education (ITME), 2011 International Symposium on*, vol. 2, pp. 257-261. IEEE, 2011.

[79] Lombardi, Flavio, and Roberto Di Pietro. "CUDACS: securing the cloud with CUDA-enabled secure virtualization." In *Information and Communications Security*, pp. 92-106. Springer Berlin Heidelberg, 2010.

[80] Basak, Debashis, Rohit Toshniwal, Serge Maskalik, and Allwyn Sequeira. "Virtualizing networking and security in the cloud." *ACM SIGOPS Operating Systems Review* 44, no. 4 (2010): 86-94.

[81] Sabahi, Farzad. "Secure Virtualization for Cloud Environment Using Hypervisor-based Technology." *Int. Journal of Machine Learning and Computing* 2, no. 1 (2012).

[82] Lombardi, Flavio, and Roberto Di Pietro. "Secure virtualization for cloud computing." *Journal of Network and Computer Applications* 34, no. 4 (2011): 1113-1122.

[83] Younis, MM Younis A., and K. Kifayat. "Secure cloud computing for critical infrastructure: A survey." *Liverpool John Moores University, United Kingdom, Tech. Rep* (2013).

[84] Zunnurhain, Kazi. "FAPA: a model to prevent flooding attacks in clouds." In *Proceedings of the 50th Annual Southeast Regional Conference*, pp. 395-396. ACM, 2012.

[85] Zunnurhain, Kazi, S. Vrbsky and Ragib Hasan, "FAPA: Flooding Attack Protection Architecture in Cloud System," *International Journal of Cloud Computing, Inderscience Publishers, a publishers of distinguished academic, scientific and professional journals*.

[86] Geneiatakis, Dimitris, Georgios Portokalidis, and Angelos D. Keromytis. "A multilayer overlay network architecture for enhancing IP services availability against dos." In *Information Systems Security*, pp. 322-336. Springer Berlin Heidelberg, 2011.

[87] Ibrahim, Amani S., James Hamlyn-Harris, John Grundy, and Mohamed Almorsy. "Cloudsec: a security monitoring appliance for virtual machines in the iaas cloud model." In *Network and System Security (NSS), 2011 5th International Conference on*, pp. 113-120. IEEE, 2011.

[88] Xu, Jia, Jia Yan, Liang He, Purui Su, and Dengguo Feng. "CloudSEC: A Cloud Architecture for Composing Collaborative Security Services." In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 703-711. IEEE, 2010.

[89] Morein, William G., Angelos Stavrou, Debra L. Cook, Angelos D. Keromytis, Vishal Misra, and Dan Rubenstein. "Using graphic turing tests to counter automated ddos attacks against web servers." In *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 8-19. ACM, 2003.

[90] Zunnurhain, Kazi, and S. Vrbsky, "FAPA: Performance Analysis with Victim and Sibling Virtual Machines," In *Proceedings of the 52nd annual Southeast regional conference*, pp. 171-176. ACM, 2014.

[91] Ristenpart, Thomas, Eran Tromer, Hovav Shacham, and Stefan Savage. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds." In *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199-212. ACM, 2009.

[92] Wadhwa, Bimlesh, Aditi Jaitly, and Bharti Suri. "Cloud Service Brokers: An Emerging Trend in Cloud Adoption and Migration." In *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, pp. 140-145. IEEE, 2013.

APPENDIX A
COMMUNICATION BETWEEN A PRIVATE AND PUBLIC CLOUD INSTANCE

This guide explains how to establish a connection between a commercially launched virtual instance and a locally launched virtual machine. The locally launched instance is built under Eucalyptus platform with Ubuntu 12.10 LTS enterprise cloud edition.

A.A Preliminary Note

When using a Windows 7 terminal, we need a putty terminal and a putty generation installed in the Windows terminal. Also, we can assume we have some public virtual machines launched in EC2.

A.B Key Encryption

1. We need to download the key pair from *AWS console*.
2. Then we need to load the key pair file in the *putty-gen*.
3. We need to select the type of key we are generating, for that purpose we need to select the *SSH-2 RSA* from the Type of key to generate the tab in putty-gen.
4. Then hit the *Generate* button.
5. After the key is generated we need to save the key. We need to select the *Save private key* option from the Save the generated key tab in putty-gen.

A.C Key Configuration

1. We need to install putty in the terminal.
2. We need to select the *Session* tab from *Category* on the left of the putty window.
3. Put the IP address of the local virtual machine with which we want to establish the communication.
4. For *Connection type* select SSH.
5. Then expand the SSH option in the *Category* list on the left.
6. Select the *Auth* option.
7. Next browse the directory where we have saved the generated private key file.
8. Finally we will hit the *Open* button.

A.D opening TCP ports for Public instance

1. We need to launch a public instance.
2. Select the option for ports by right clicking on the VM in the AWS console.
3. Then we need to allow access for TCP ports, which is generally Default port 80 only.
4. Opening all the ports might cause the instance to become unsecure, so we must specify a specific port for that.

Finally, we need to configure our private cloud instance with the generated private key (initially downloaded from the AWS console). More about this will be explained in the next Appendix. Once the private cloud is launched, the communication can be established by sending packets through *iperf* using the same port that was opened in the EC2.

APPENDIX B
BUILDING AND DEPLOYING VIRTUAL MACHINES USING UBUNTU 12.10 LTS

In this appendix we will install and use KVM for creating and launching virtual machines on an Ubuntu 12.10 LTS server. It will guide how to create image-based virtual machines. KVM is Kernel Based Virtual Machine and it makes use of hardware virtualization.

B.A Preliminary Note

We need to follow the steps in this tutorial with root privileges, but we can also use the *sudo* command before the instruction, it works the same.

B.B Installing KVM and vmbuilder

We need to check if the CPU we are using supports hardware virtualization with the following command

sudo egrep '(vmx|svm)' –color=always /proc/cpuinfo

There should be a display of messages in the Ubuntu terminal.

If nothing is displayed then the processor doesn't support hardware virtualization, and we need to consider another machine.

To install KVM and vmbuilder, use the following command:

Sudo apt-get install Ubuntu-virt-server python-vm-builder kvm

Then we need to add the user to which we are currently logged in to the "libvirtd":

adduser 'username' libvirtd
adduser 'username' kvm

We need to log out and log back in again to allow the membership to take effect. To check the successful installation of KVM, run the following command:

virsh –c qemu:///system list

It should display something similar to:

Id Name State
———————————————

If something else is displayed then there is an error in the installation.

Now, we need to set up a network bridge so that virtual machine can be accessed from other hosts as if they were in the same physical server. We need to install the bridge-utils package:

sudo apt-get install bridge-utils

Also, we need to configure a bridge by going to the directory first:

Sudo nano /etc/network/interfaces

Before the modification, the interface file looks like:

```
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 192.168.0.100
        netmask 255.255.255.0
        network 192.168.0.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
        dns-nameservers 8.8.8.8 8.8.4.4
```

After the modification it should look like:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# The loopback network interface

auto lo
iface lo inet loopback

# The primary network interface

auto eth0
iface eth0 inet manual
auto br0
iface br0 inet static

        address 192.168.0.100
        network 192.168.0.0
        netmask 255.255.255.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
        dns-nameservers 8.8.8.8 8.8.4.4
        bridge_ports eth0
        bridge_fd 9
        bridge_hello 2
        bridge_maxage 12
        bridge_stp off
```

Then we need to restart the network:

sudo /etc/init.d/networking restart

Before creating the virtual machines, we need to reboot the system. If we do not reboot then we might get an error such as open "/dev/kvm: Permission denied" in the virtual machine logs in the "/var/log/libvirt/qemu" directory.

B.C Creation of an Image-Based Virtual Machine

Usually the newly created images are stored in this directory: /var/lib/libvirt/images. We should create individual directories for individual virtual machines, such as, "/var/lib/libvirt/images/vm1", "/var/lib/libvirt/images/vm2" and "/var/lib/libvirt/images/vm3", and so on. Trying to create a second vm in "/var/lib/libvirt/images/vm1" will create an error message saying "Ubuntu-kvm already exists".

We will use the "vmbuilder" tool to create VMs. "vmbuilder" uses a template to create virtual machines - this template is located in the "/etc/vmbuilder/libvirt/" directory.
First we create a copy:
sudo mkdir –p /var/lib/libvirt/images/vm1/mytemplates/libvirt
sudo cp /etc/vmbuilder/libvirt/ /var/lib/libvirt/images/vm1/mytemplates/libvirt/*

Now we partition our VM. We have to create a file named "vmbuilder.partition":
sudo nano /var/lib/libvirt/images/vm1/vmbuilder.partition

Next, define the desired partition. For FAPA define the partition as follows:

root 2000
swap 4000
———

/var 2000

This defines a root partition with 2GB, a swap partition of 4GB and a variable partition of 2GB. Now we need to install the "Openssh-server' when we create the VM. Therefore, we create a script called "boot.sh" that will be executed when the VM is booted for the first time. It will install "Openssh-server" (with a unique key) and also force the user to change the password when the user logs in for the first time:

sudo nano /var/lib/libvirt/images/vm1/boot.sh

** During this script we appended some internal task for installing the Openssh server with a private key, so when the VM was launched, the same encryption was generated in both the local VM and the public VM to find each other during a TCP request.

We need to edit the boot.sh Script to contain:

```
# This script will run the first time the virtual machine boots
# It is ran as root.
# Expire the user account
passwd -e administrator

# Install openssh-server
apt-get update
apt-get install -qqy --force-yes openssh-server
```

Now use help:

sudo vmbuilder kvm Ubuntu --help

to learn about available options. To create our first VM, vm1, we go to the VM directory:
cd /var/lib/lbvirt/images/vm1/

Then run "vmbuilder":

```
vmbuilder kvm ubuntu --suite=precise --flavour=virtual --arch=amd64 --
mirror=http://de.archive.ubuntu.com/ubuntu -o --libvirt=qemu:///system --
ip=192.168.0.101 --gw=192.168.0.1 --part=vmbuilder.partition --templates=mytemplates -
-user=administrator --name=Administrator --pass=howtoforge --addpkg=vim-nox --
addpkg=unattended-upgrades --addpkg=acpid --
firstboot=/var/lib/libvirt/images/vm1/boot.sh --mem=256 --hostname=vm1 --bridge=br0
```

The build process may take few minutes.

Afterwards, an XML configuration file can be found for "/etc/libvirt/qemu/" (=> "/etc/libvirt/qemu/vm1.xml"):

```
ls -l /etc/libvirt/qemu/
root@server1:/var/lib/libvirt/images/vm1# ls -l /etc/libvirt/qemu/
total 8
drwxr-xr-x 3 root root 4096 May 21 13:00 networks
-rw------- 1 root root 2082 May 21 13:15 vm1.xml
root@server1:/var/lib/libvirt/images/vm1#
```

The disk images are located in the ubuntu-kvm/ subdirectory of our VM directory:

```
ls -l /var/lib/libvirt/images/vm1/ubuntu-kvm/

root@server1:/var/lib/libvirt/images/vm1# ls -l /var/lib/libvirt/images/vm1/ubuntu-
kvm/
total 604312
-rw-r--r-- 1 root root 324337664 May 21 13:14 tmpE4IiRv.qcow2
-rw-r--r-- 1 root root 294715392 May 21 13:15 tmpxvSVOT.qcow2
root@server1:/var/lib/libvirt/images/vm1#
```

B.D Managing and Connecting to the Virtual Machine

VMs can be managed through "virsh", the virtual shell. To connect to the virtual shell run:
Virsh –connect qemu:///system

This is what the virtual shell looks like:

```
Welcome to virsh, the virtualization interactive terminal.

Type: 'help' for help with commands
      'quit' to quit

virsh #
```

To show all virtual machines, running and inactive:

```
virsh # list --all
 Id Name                 State
----------------------------------
  - vm1                  shut off
  - vm2                  shut off

virsh #
```

Before we start a new VM for the first time, we must define it from its xml file (located in the "/etc/libvirt/qemu/" directory):

sudo virsh define /etc/libvirt/qemu/vm1.xml

Now we can start the VM:

sudo virsh start vm1

After a few moments, we should be able to connect to the VM with an SSH client, such as PuTTY; then log in with the default username and password. After the first login the user will be prompted to change the password.

sudo virsh list –all

```
Id Name                 State
----------------------------------
  1 vm1                  running
```

To connect to the VM: *sudo ssh kvmuser@192.168.1.246*

The user will be asked to add the RSA key pair to the known hosts file, and change his password for the "kvmuser" account on vm1.

Successful connection to the VM!