Establishing the Software-Defined Networking Based Defensive System in Clouds

by

Tianyi Xing

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree of
Doctor of Philosophy

Approved October 2014 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Guoliang Xue
Arunabha Sen
Deepankar Medhi

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

Cloud computing is regarded as one of the most revolutionary technologies in the past decades. It provides scalable, flexible and secure resource provisioning services, which is also the reason why users prefer to migrate their locally processing workloads onto remote clouds. Besides commercial cloud system (i.e., Amazon EC2[15]), ProtoGENI [9] and PlanetLab [68] have further improved the current Internet-based resource provisioning system by allowing end users to construct a virtual networking environment. By archiving the similar goal but with more flexible and efficient performance, I present the design and implementation of MobiCloud [83] that is a geo-distributed mobile cloud computing platform, and G-PLaNE [85] that focuses on how to construct the virtual networking environment upon the self-designed resource provisioning system consisting of multiple geo-distributed clusters. Furthermore, I conduct a comprehensive study to layout existing Mobile Cloud Computing (MCC) service models and corresponding representative related work in [41]. A new user-centric mobile cloud computing service model is proposed to advance the existing mobile cloud computing research.

After building the MobiCloud, G-PLaNE and studying the MCC model, I have been using Software Defined Networking (SDN) approaches to enhance the system security in the cloud virtual networking environment. I present an OpenFlow based IPS solution called SDNIPS [86] that includes a new IPS architecture based on Open vSwitch (OVS) in the cloud software-based networking environment. It is enabled with elasticity service provisioning and Network Reconfiguration (NR) features based on POX controller. Finally, SDNIPS demonstrates the feasibility and shows more efficiency than traditional approaches through a thorough evaluation.

At last, I propose an OpenFlow-based defensive module composition framework called CloudArmour that is able to perform query, aggregation, analysis, and con-

trol function over distributed OpenFlow-enabled devices. I propose several modules and use the DDoS attack as an example to illustrate how to composite the comprehensive defensive solution based on CloudArmour framework. I introduce total 20 Python-based CloudArmour APIs. Finally, evaluation results prove the feasibility and efficiency of CloudArmour framework.

DEDICATION

In Memory Of My Grandparents

ACKNOWLEDGEMENTS

First, I want to truly express my foremost gratitude to my advisor, Dr. Dijiang Huang, for his mentoring, guidance and support during my PhD study. His excellent guidance, profound insight in research and extraordinary dedication to work helped me to finish my dissertation smoothly and successfully. He is a modest, precise, amiable, understanding person. He is the mentor not only for my study but also for my life. I always feel lucky to be supervised by him. By working with him for several years, I have been shaped into a better person. Without his mentoring, I would not be able to become who I am today.

I would also like to thank my committees for their supports. Dr. Guoliang Xue, Dr. Arun Sen and Dr. Deep Medhi provided me with invaluable advice and comments on my research. Their feedback helped me to improve the dissertation in many ways.

I am also thankful to all members from SNAC lab and friends for their kind supports. I am paticularly thankful to Dr. Yang Qin, Dr. Zhibin Zhou, Yuan Wang, Zhengyang Xiong, Huijun Wu, Bing Li, Ziming Zhao, Yiming Jing, Chun-Jen Chung, for their inspiring discussions.

Lastly, but absolutely not least, my family were always supporting me and stood by me through the good times and bad. I have been living separately with my wife, Chau Lam, since I started my PhD program back in 2010. Due to my busy study, my wife visited me at Phoenix from NYC almost every month and she never complained about it. Without her considerate thought and kind understanding, I would not be able to finish my PhD program.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Cloud computing has grown rapidly in recent few years due to the increasing network bandwidth, mature virtualization techniques, and emerging cloud based business demands. Besides the traditional service models of cloud computing, i.e., IaaS, PaaS and SaaS, new models keep coming out to provide new types of services, e.g., StaaS (Storage as a Service), DaaS (Desktop as a Service), NaaS (Network as a Service), etc. After the neonatal stage of cloud computing, especially within the recent few years, cloud computing has shifted its focus from fixed users oriented to dynamic mobile users oriented, from local to global, from centralized to distributed architecture. Among all issues related to clouds, mobile cloud computing platform, user centric mobile cloud service model, and virtual networking security have been regarded as most critical concerns as well as my research focus.

1.1 Mobile Cloud Computing System

Today, the Internet web service is the major way that we access information from fixed or mobile terminals. Information is stored on Internet clouds, where computing, communication, and storage services are common services provided for Internet users. In a non-distant future, many of our queries will be beyond current Internet scope and will be about the people, the physical environments that surround us, and virtual environments that we will be involved. With the Internet environment getting improved, mobile phones will overtake PCs as the most common web access entities worldwide by 2014 as predicted by Gartner [53]. Current mobile devices have many advanced features such as mobility, communication and sensing capabilities, and can

serve as the personal information gateway for mobile users. However, when running complex data mining and storing operations, the computation, energy, and storage limitations of mobile devices demand an integrated solution relying on cloud-based computation and storage support. As a result, a new research field, called Mobile Cloud Computing (MCC), is emerging to meet the increasing demand and address the issues.

The trend of the MCC system is not just aiming to provide services for users in some certain areas, but is especially looking forward to establishing connections among mobile users all over the world. Due to the mobility of MCC users, a geographically distributed cloud system is a natural choice that allows users to connect to the cloud resource that is geographically "close" to their mobile devices, which usually means less communication delay compared to the centralized approach. Here, a geographically distributed MCC system refers to an infrastructure combining multiple cloud clusters (with dedicated computing, storage, and communication resources) located at different locations. Unlike the centralized service provisioning data centers, in geographically distributed data centers, users' requests will be responded by the closest and least loaded data centers, that guarantees better user experience. Many cloud service providers, like Google and Amazon, place their data centers all over the world to provide rapid resource and service access for end users.

1.2 Mobile Cloud Computing Service Model

One the mobile cloud computing platform is physically established, another significant research topic goes to the logic part, i.e., service model. In MCC, a mobile entity can be considered as either a physical mobile device or a mobile computing/storage software agent within a virtualized cloud resource provisioning system. In the latter view of the cloud system, a software agent's main functionality is the mobility

associated with the software codes. In other words, mobile cloud applications may migrate or compose software codes in the distributed MCC resource-provisioning environment. Mobile cloud services will account for delay, energy consumption, real-time entity presence, information caching capabilities, networking and communication connectivity, data protection and sharing requirements, and so on. By achieving these features, we are able to create a new world composed by both physically networked systems and virtualized entities that are mapped to the physical systems preserving and in some cases extending their functions and capabilities.

MCC distinguishes its research focuses on a tight interaction between, and construction and integration of, the Cyber Physical System (CPS) and Cyber Virtual System (CVS), in which the CPS is immensely composed by computational and physical smart and mobile entities, and the CVS is mainly formed by cloud-based virtualized resource and services. Recent developments of Augmented Reality (AR) [13, 72, 17, 22] have demonstrated some of the application capabilities of MCC.

1.3 Cloud Network Security Issues

Cloud computing technologies have been widely adopted today due to its resource provisioning capabilities, such as scalability, high availability, efficiency, and so on. However, security is one of the critical issues that have not been fully addressed [14]. Attackers may compromise vulnerable virtual machines (VMs) to form botnets, and then deploy distributed denial-of-service (DDoS) attacks or send spams, which have become a major security concern of using cloud services. Several critical cloud network security issues are highlighted as below:

1. Abuse and Nefarious Use of Cloud Computing. IaaS providers offer their customers the illusion of unlimited compute, network, and storage capacity. By abusing the relative anonymity behind these registration and usage models, spam-

mers, malicious code authors, and other criminals have been able to conduct their activities with relative impunity. PaaS providers have traditionally suffered most from this type of attacks; however, recent evidence shows that hackers have begun to target IaaS vendors as well [46]. Future areas of concern include password and key cracking, launching dynamic attack points, hosting malicious data, botnet command and control, etc.

2. Malicious Insiders. The threat of a malicious insider is well-known to most organizations. This threat is amplified for consumers of cloud services by the convergence of IT services and customers under a single management domain, combined with a general lack of transparency into provider process and procedure. In traditional computer networking systems, security protection is usually deployed at the edge of the system, for example, the firewall system. However, an attacker can break the firewall or DMZ and get access into the internal network, these attack consequences can be very servere. Since all resources in the same domain is trusted among each other by default, insider attacks can cause more damage than outsider attacks.

3. Data Integrate. Storage is one of the most important and common scenarios in clouds. Therefore, compromising stored data, e.g., deletion or alteration of records without a backup of the original content, becomes another critical security issue in clouds. The authentication and authorization of the data must securely guarantee that unauthorized or unauthenticated parties must be prevented from gaining access to privacy data. The threat of data compromise increases in the cloud, due to the number of and interactions between risks and challenges which are either unique to cloud, or more dangerous because of the architectural or operational characteristics of the cloud environment.

4. Virtualization Hijacking. One of the significant characteristics of the cloud computing is the virutalization, which enables better resources utilization and fine

grained resource isolation. IaaS vendors provide their services by sharing the physical infrastructure in a scalable fashion. However, the underlying components building up the infrastructure (e.g., CPU, GPU, etc.) were not dedicated designed to deliver strong isolation capability in a multi-tenant environment. To address this issues, hypervisor is designed and introduced to fill the gap between the physical infrastructure and guest operating system. However, the existing hypervisor is not flawless and can still be compromised in that it enables users to gain access to inappropriate level of control to guest OS. A defense in depth strategy is recommended, and should include compute, storage, and network security enforcement and monitoring. Strong compartmentalization should be also employed to guarantee that individual customers do not impact the operations of other tenants running on the same cloud service provider. Customers should not have access to any other tenant's actual or residual data, network traffic, etc.

1.4 Outline

My research interests mainly involve the cloud infrastructure design and implementation, cloud service design and implementation, software defined network, cloud network security, etc. The outline of this dissertation is arranged as follows: Chapter 2 discusses the related work for all related research areas, and a brief summary of them. Chapter 3 mainly discusses the research challenges and expected objective for my research. MobiCloud and G-PLaNE system are discussed in chapter 4 and 5 respectively. Chapter 6 introduces a new MCC model and corresponding exemplar architecture. Chapter 7 discusses a SDN-enabled IPS system in cloud environment. A SDN-enabled modular composition framework called CloudArmour is discussed in chapter 8. Finally, chapter 9 concludes this dissertation and discusses the future work.

Chapter 2

STATE OF THE ART

2.1 Cloud System & Platform

Global Environment for Network Innovations (GENI) [10], is a project exploring the future global networking infrastructure where different types of resource provisioning platforms are residing. GENI projects can be divided into backbone networks, programmable hosts, wireless testbeds and specialized aggregates. Different GENI platforms, e.g., PlanetLab [68], ProtoGENI [9], and OpenFlow Networks [54], have different concentration in terms of provisioning resources, network architecture, programmable networks, etc. For example, ProtoGENI has integrated a large group of

Table 2.1: Comparison table of GENI Projects

Project	Major Resource	Sensing Capability	Programmable Networks	Extension Simplicity
PlanetLab	Fedora VM	No	No	Difficult
ProtoGENI (Emulab)	PC and VM	**USRP**	**Yes**	Difficult
OpenFlow Networks	OF Switch	No	**Yes**	DD
GENICloud (opencirrus)	Physical node	No	No	NA
Seattle	Experimenter Software	No	No	**Easy**
ORBIT	Dedicated node	No	**Yes**	NA
DOME	Linux VM	No	No	NA
DETER (Emulab)	PC	No	**Yes**	NA
Kansei	Sensing node	**Yes**	No	NA
ViSE	Debian VM	**Yes**	**Yes**	NA
GpENI	Fedora VM	No	**Yes**	DD
MobiCloud	Windows, Linux VMs	**Yes**	**Yes**	**CaaS**

DD: Dedicated Device, NA: Not Allowed

resources available from the world to provide resources with network programmability and sensing features. Seattle [26] has an efficient design that can easily make spare nodes join their available resource pool to be further utilized to provide python based experiments. All GENI related projects [68, 9, 54, 25, 26, 69, 74, 21, 16, 43, 76] and our proposed MobiCloud are summarized in Table 2.1.

2.2 Mobile Cloud System and Services

Table 2.2: Summary of MCC Services and Applications

MCC Services and Applications		Service Models		
MCC Service Types	Representative Approaches	MaaSC	MaaSP	MaaSB
Mobile Cloud Computation	CloneCloud [28]	✓		
	MAUI [32]	✓		
	ThinkAir [49]	✓		
Mobile Cloud Storage	Dropbox, Box, iCloud, GoogleDrive and Skydrive [31]	✓		
	WhereStore [77]	✓		
	STACEE [60]	✓	✓	
Security and Privacy	CloudAV [63]	✓		
	Secure Web Referral Services for Mobile Cloud Computing [50]	✓		
	Zscaler [12]	✓		
	Google Wallet [37]	✓		
Context Awareness	An Integrated Cloud-based Framework for Mobile Phone Sensing [34]		✓	✓

I summarize existing MCC services and applications in Table 2.2. I defined 3 service models, Mobile as a Service Consumer (MaaSC), Mobile as a Service Provider (MaaSP), Mobile as a Service Broker (MaaSB), based on the role of mobile devices in the MCC environment. The service models are classified based on the role of the mobile device, such as service consumer, service provider, and service broker. More detail on the service model will be discusses in chapter 6. I also discuss corresponding

representative projects. Each service or application can be categorized into one or multiple service models. MaaSC is the most common MCC service model because most of existing mobile devices are still restricted by their computation and energy capacities. As an example, clonecloud [28] provides the computation task offloading service for mobile devices. In this case, the mobile device is the service consumer since it only gets benefit from the service provided by cloud rather than providing services for other users.

2.2.1 Mobile Cloud Computation

Computation task offloading is a demanding feature for mobile devices relying on Internet clouds to perform resources-intensive computation tasks. Partitioning computation tasks and allocating them between mobile devices and clouds can be very inefficient during the application runtime considering various performance metrics such as energy consumption, CPU power, network delay, etc. How to efficiently and intelligently offload the computation tasks onto the cloud is one of the main research issues of MCC. CloneCloud [28] and MAUI [32] are two pioneer work in this area. They both can automatically offload computing tasks to the cloud.

CloneCloud serves as an application partitioner as well as an execution runtime environment that allows unmodified mobile applications seamlessly offloading parts of the executions from mobile devices onto a cloud server. The offloading decision is made by optimizing execution time and energy usage for mobile devices. Contrast to CloneCloud, MAUI allows modifying offloading applications at the coding level to maximize the energy saving of mobile devices. Thinkair [49] demands dedicated virtual machines (VMs) in clouds as part of a complete smartphone system, and removes the restrictions on applications/inputs/environmental conditions by using an online method-level offloading.

2.2.2 Mobile Cloud Storage

Storage capacity is another constraint of mobile devices. There are many existing storage services for mobile devices, e.g., Dropbox, Box, iCloud, Google Drive, and Skydrive [31]. Besides manually uploading the files or data onto the cloud, one desired feature of mobile cloud storage services is the automatic synchronization between mobile devices and the cloud. Multimedia data generated by mobile devices demands a stable and high available storage solution. This is the reason that many smartphone operating systems natively implant the multimedia data synchronization feature, e.g., iCloud for iOS, Skydrive for Window Phone, Google Drive for Android, etc. Moreover, mobile users' behavior data such as location traces, browsing history, personal contacts, preference settings need to be kept in a reliable and protected storage space. Most existing commercial cloud storage solutions are built on a centralized data center, which is appropriate for Internet Clouds.

Storage mobility has been gradually becomes a recent research focus. WhereStore [77] is a location-based data storage solution for smartphones. It uses filtered replication (a filter expressing the set of data items that are likely to be accessed in the near future) along with each device's location history to distribute data items between smartphones and the cloud. STACEE [60] proposes a peer-to-peer cloud storage where mobile phones, tablets, set-top-boxes, modems and networked storage devices can all contribute as storage within these storage clouds. It provides a peer-to-peer (P2P) cloud storage solution and addresses the storage issue for mobile users as a QoS-aware scheduling problem.

2.2.3 Security and Privacy

Security related services aim to provide data security protections through the cloud. Security of mobile devices can be enhanced under the help of cloud security mechanism including cloud-based secure proxy, remote anti-virus, remote attestation, etc.

CloudAV [63] advocates such a cloud-based security model for malware detection for end hosts by providing antivirus as an in-cloud security service. Secure web referral services [50] enable the antivirus and anti-phishing services through the cloud. The referral services depend on a secure search engine to validate URLs accessed by a mobile device to prevent mobile users from accessing phishing websites.

Zscaler [12] is one of the most well-known commercial cloud-based security company that provides policy-based, secure internet access for mobile devices. It provides a comprehensive cloud-based security solution including three main components: ZEN (proxy), CA (central authority), and Nanologs server (log server). Various cloud-based security services are built based on these components. For example, the *ByteScan* service enables each ZEN to scan every byte of the web request, content, responses and all related data to block malicious actions and data such as viruses, cross site scripting (XSS), botnets, etc.; The *PageRisk* service relies on the ZEN to computes a PageRisk index for every page loaded and enables the administrator to control content served to their users based on an acceptable risk evaluation; the *NanoLog* service enables administrators to access any transaction log in realtime.

An increasing number of security features can be enabled in cloud, in which a reliable and secure connection between a mobile device and the cloud is the main challenge for this type of solutions. Google Wallet [37] is developed on a cloud-mobile dual trust root model, where the cloud is in charge of the application level security

such as credit card transactions and user credential management, and the Google Wallet mobile device is protected by strong trust computing elements on the board to prevent malicious attacks on the mobile devices.

2.2.4 MCC Context Awareness

Nowadays, a smart mobile device usually serves as an information gateway for mobile users involving various personalized activities such as checking emails, making an appointment, surfing the web, locating some interested spots, analyzing personal behavior data based on data mining and machine learning, etc. For example, in [34], each mobile device has a dedicated Mobile Cloud Engine (MCE) including three modules: decision module, publish subscribe module, and context awareness module. The decision module handles and regulates the transactions among the different parts of the MCE. The publish subscribe module is responsible for establishing the data flow between the mobile application and the MCE. Finally, the context awareness module provides context information to the application. The state-of-the-art solutions lack a unified approach suitable to support diverse applications, while reducing the energy consumption and providing intelligent assistance to mobile users.

2.3 Traditional Security Solutions for DDoS attack

Among all malicious behaviors, Distributed Denial of Service (DDoS) is one of the most critical issues in cloud virtual networking environment since the DDoS attack is hard to defend once the internal resources are controlled by attackers. Thus, in my research work, DDoS has been used to demonstrate the security defensive solution. According to the definition from WWW Security FAQ [75] on Distributed Denial of Service (DDoS) attacks: “A DDoS attack uses many computers to launch a coordinated DoS attack against one or more targets. Using client/server technology, the

perpetrator is able to multiply the effectiveness of the DoS significantly by harnessing the resources of multiple unwitting accomplice computers, which serve as attack platform". The countermeasure of DDoS includes identification, trace, and prevention action.

In [33], author classified the solution of DDoS attack into different ways, such as filtering, disabling unused services, changing IP address, load balancing, honeypots, etc. Data-mining technology is normally applied in DDoS attack detection. In [78], authors used an incremental mining approach to detect large-scale attacks for network intrusion detection system. This work is distinguished because traffic information is obtained in real time and the analysis is not done by static data. Another work [88] presented a multiple layers game-theoretic framework for DDoS attack and defense evaluation. An innovative point in this work is the strategic thinking of attacker's perspective benefit the defense decision maker in this interaction between attacks and defenses. However, two works above are not suitable for deploying dynamic network threats countermeasure and has no real time security solution for real time attacks. In [89], authors proposed a dynamic resource allocation strategy to counter DDoS attacks against individual cloud customers. When a DDoS attack occurs, they employ the idle resources of the cloud to clone sufficient intrusion prevention servers for the victim in order to quickly filter out attack packets and guarantee the quality of the service for benign users simultaneously. However, this paper focused on how to allocated idle resource for IPS but did not discuss how the IPS prevent the DDoS attack.

Since DDoS attack is different from other types of attacks in terms of the great number of the attack sources, packet marking technique is widely used for IP traceback to locate the attack sources. In [36], the authors present a marking mechanisms for DDoS traceback, which injects a unique mark to each packet for traffic identi-

fication. As a Probabilistic Packet Marking(PPM) method, it has a potential that leads attackers to inject marked Packet and spoofed the traffic. [20] is another important traceback method by using Deterministic Packet Marking (DPM). The victim could track the packets from the router which splits the IP address into two segments. Differing from previous methods, the authors present an independent method to traceback attacker based on entropy variations in [90]. However, most of these works did not handle the IP spoofing effectively, and packet modification needs to be enabled, which introduces more overhead and even vulnerabilities.

2.4 SDN-enabled Cloud Network Solutions

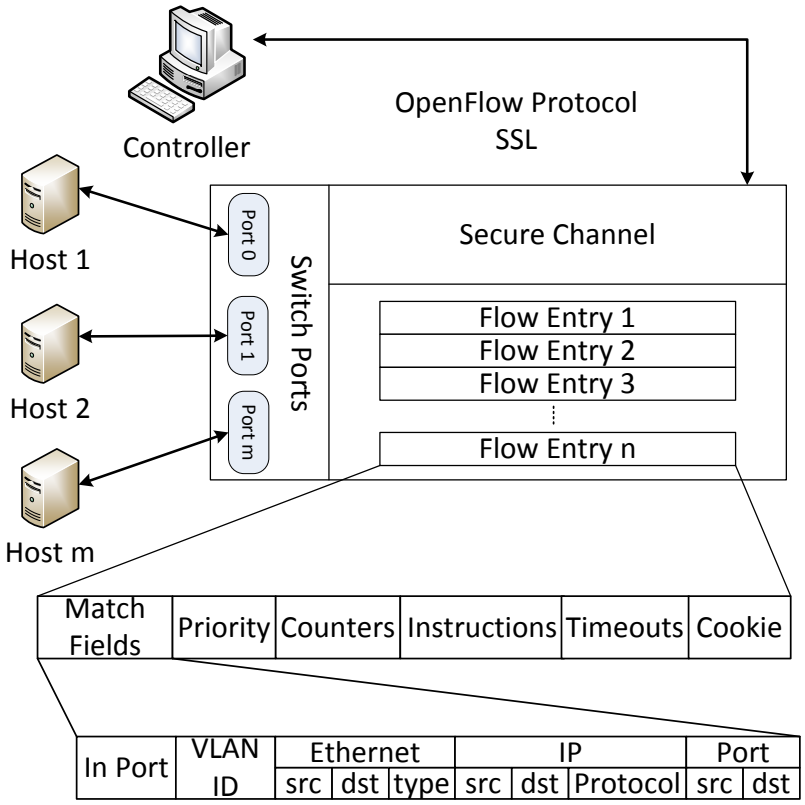


Figure 2.1: The OpenFlow Architecture.

OpenFlow is a representative standard and protocol implementing SDN concept. As shown in Fig. 2.1, it introduces a centralized controller and defines standard in-

terfaces to the controller for installing a packet-forwarding rules in the flow table that is defined in [64]. The flow table is in the data plane and is able to rapidly handle incoming packets at line rate. It mainly has fields such as match fields, priority, counters, instructions, etc. The statistics counter field of the flow table records statistics of traffic passing through all the OpenFlow devices, such as individual and aggregate flow statistics, flow table statistics, port statistics, queue statistics, etc. In the OpenFlow architecture, a controller executes all control tasks of the switches and is also used for deploying new networking frameworks, such as new routing protocols or optimized cross-layer packet-switching algorithms. When a packet arrives at an OpenFlow switch (OFS) or Open vSwitch (OVS), the switch processes the packet in the following three steps:

1. It checks the header fields of an incoming packet and tries to match any entry in the local flow table. If there is no matching entry in the flow table, the packet is encapsulated and is sent to the controller for further processing. There can be multiple flow control rules. It follows a best matching procedure to pick the best rule, e.g., the one with the highest priority.

2. It then updates the byte and packet counting information associated with the rules to log statistics, which can be used to collect the traffic information to detect any abnormal traffic.

3. Once a matching rule is decided, the OpenFlow switch takes the action based on the corresponding flow table entry, e.g., forwarding to a specific port, or dropping.

OpenFlow switch separates the control plane and data plane by virtualizing the network control as a network OS layer. The network controller is considered as the

software engine to deploy the control functions that can be implemented through automatic control algorithms. With these features, dynamic network reconfiguration can be implemented in the cloud virtual networking environment. There are several OFS controllers available following the OpenFlow standard, such as NOX/POX [38]. Both OVS and OFS are OpenFlow protocol enabled switches. For all works mentioned in this report, I choose to use OVS to represent all the OpenFlow switches implemented as a software switch in the cloud environment, where OVS is usually implemented in the privilege domain of a cloud server, e.g., Domain 0 of XenServer [30] and the host domain of KVM [3]; while the OFS represents physical OpenFlow switches.

OpenFlow switch, as a programming network device with high flexibility and scalability, has been largely deployed to enable new services or enhance performance especially for network scenarios [48, 45, 79]. Also, combining the OpenFlow with other opensource packages creates new opportunities. QuaggaFlow [58] integrates the Quagga opensource routing suite with OpenFlow to provide a centralized control over the physical OpenFlow switches and Quagga router in Virtual Machine (VM). However, very few researchers started using OpenFlow as a way for security purpose, especially in cloud environment.

2.5 SDN-enabled Security Solutions

SDN has been well researched to establish monitoring system [70] [18] [80] due to its centralized abstract architecture. These three works all utilize the programmable fabric of OpenFlow monitor the traffic and detect the malicious behavior. However, all three aforementioned studies did not specifically consider the SDN-based detection for DDoS attack and countermeasure for malicious behaviors. OpenSafe[18] is a network monitoring system that allows administrators to easily collect usage statistics

of networking and detect malicious activities by leveraging programmable network fabric. It uses OpenFlow technique to enable some manipulations of traffic, such as selective rules matching and arbitrary flows directing, to achieve its goal. Furthermore, ALARMS is designed as a policy language to articulate paths of switches for easily network management. OpenNetMon [80] is another network monitoring application based on OpenFlow platform and is implemented to monitor per-flow metrics to deliver fine-grained input for traffic engineering. Benefiting from the OpenFlow interfaces that enable statistic query from controller, authors proposed an accurate way to measure per-flow throughput, delay and packet loss metrics. In [70], authors proposed a new framework to address the detection problem by manipulating network flows to security nodes for investigation. This flow-based detection mechanism guarantees all necessary traffic packets are inspected by security nodes. However, all three aforementioned studies did not specifically consider the SDN-based detection for DDoS attack and did not discuss the countermeasure for malicious activities.

SDN technology is also used to detect specific threat, e.g., DDoS, in cloud system. In [23], the authors present a flow-based lightweight method for DDoS attack detection. This solution is implemented on NOX/OpenFlow platform to collect traffic statistic information from flow table by Flow Collector. Important DDoS relevant features will be extracted from feature extractor module and be sent to classifier where classification algorithm comes from Self Organizing Maps (SOM)[47] method. However, this paper does not provide an optimized detection solution for distributed cloud environment and corresponding countermeasures for the DDoS attack.

Besides detection, proposing SDN-enabled prevention solutions is another key direction for SDN based security research. CONA [27] is a content-oriented networking architecture build on NetFPGA-OpenFlow platform. In this design, hosts request contents and agents deliver the requested contents while the hosts can not. Under

Table 2.3: Security Solutions Comparison Table

Project	Cloud	Detection	Prevention	DDoS
CloudWatcher [70]	✓	✓		
OpenSafe [18]	✓	✓		
OpenNetMon [80]	✓	✓		
Avant-Guard [71]	✓		✓	✓
Lightweight DDoS Detection [23]		✓		✓
ORHM [44]	✓		✓	✓
CONA [27]		✓	✓	✓
NICE [29]	✓	✓		
SnortFlow [84]	✓		✓	

the content-aware supervision, system can perform prevention by: (1) collecting suspect flows information from others agents for analysis, and (2) applying rate limit to each of relevant agents to slow down the overwhelming malicious traffic. OpenFlow Random Host Mutation (OFRHM) [44] is another innovative solution by using Moving Target Defense (MTD) to protect the targets from being attacked though changing its identity representation. In this research, they add a transparent virtual IP layer above real IP and make real IP addresses untouchable by unauthorized entities. The system will assign a virtual IP to each host after each mutation interval. However, this work is a proactive solution and does not work when the attack is initiated from malicious insiders who are aware of the real IP address of victims.

Finally, besides enhancing the security level by utilizing the SDN technology, enhancing the security of the SDN itself is also a hot topic. In [71], authors proposed AVANTGUARD, a new framework to advance the security and resilience of openFlow networks with greater involvement from the data-plane layer. It makes SDN security applications more scalable and responsive to dynamic network threats. They introduce actuating triggers that automatically insert flow rules when the network is under duress. The overhead of the proposed extension over SDN was evaluated to be minimal. All SDN-based security solution are summarized in Table. 2.3.

Chapter 3

RESEARCH CHALLENGES AND OBJECTIVES

The research challenges and objectives are classified into the following several perspectives.

3.1 Mobile Cloud Computing Platform

Several MCC system is described in the chapter 2, however, there are more issues brought by such emerging architecture.

Resource Management. When some physical machines in the datacenter are approaching the capacity limit of the system, all devices accessing those machines will be affected. Without efficient resource management approaches, service providers have no choice but to purchase additional physical cloud servers to overcome this issue. To decrease the cost as much as possible, efficient resource management approach that considers resource scheduling, workload balancing, etc., are needed to handle the crisis introduced by hardware resource shortages.

Resource Diversity. Virtual machine is the major format of the resource due to its efficient utilization of hardware resources, although some systems choose to provide physical nodes. All major guest OSes can be supported in a main stream virtualization environment, e.g., Xen [66]. However, current service providers pay more attention to system architecture design but not enough to resources provisioning diversity, which means users have little choice on resources. The resource diversity includes not only the type of guest OS, but also different virtual hardware configurations.

Network Programmability and Sensibility. A static and simple network cannot meet the increasing demand from users nowadays. Network programmabil-

ity usually separates the control and data path of network device, and provides an interface of the control path so that the control function can be easily programmed. Programmable network devices are more flexible in that they make decisions based on not only predefined rules, but also on dynamically generated policies. Sensibility provides a bridge connecting cloud resources and the physical world. Sensibility greatly expands the range of experiments supported in the system.

Extensibility and Scalability. All resource provisioning systems are facing the extension issue when the number of users is increasing. This issue involves two tracks on internal and external one. For the internal issue, the system should consider how to expand the system without changing and affecting the current running system. For example, add more storage servers. For the external issue, the system should consider how others can easily join the system family without much effort. For example, how to make the donated nodes to easily become available resources in system resource pool.

SLA Guarantee. Since cloud resource provisioning is a neonatal service, the performance is not as optimal as others imagine. With a growing number of cloud resource provisioning platforms being developed, how to provide resources within an acceptable time period is another major concern. Besides responding time, SLA also refers to system availability. For some geo-distributed research-oriented platforms, e.g., ProtoGENI, multiple universities or institutions are providing resources collaboratively. It always happens that some of the resources are not available due to node failure or connection failure. Thus, how to enhance the reliability and availability of all resources is also a critical concern for current MCC platform development.

Inter-Domain Communication. Users probably expect a type of resource such as virtual machines (VMs) that connect with each other across domains. Furthermore, in this virtual network resource environment, data and services need to be transferred

from one to another sometimes, which is highly dependent on inter-domain communication. How to optimize the inter-domain communication to establish highly connected resources and transfer the service or data to the destination server without noticed delay is also another challenge for the existing geo-distributed cloud system.

To address challenges and achieve objectives illustrated above, I choose Xen-based virtualization solution which is an efficient and flexible hypervisor and supports a wide range of guest OS. For the backend network of the cloud system, I enable the SDN control in the form of Open vSwitch (OVS) to flexibly control the network traffic and manage the virtual network. Another major concern is how to improve the cloud system from centralized architecture into the geo-distributed architecture, which raises another problem that the interconnection among different clusters should be guaranteed. After enabling the SDN network management as our backend network function, I establish the GRE tunnel among all remote site. Thus, the inter-cluster connection is established in a layer-2 fashion, which VLAN is still supported across different sites. After the mainframe is built, I can further design the higher layer resource provisioning and monitoring function.

3.2 Mobile Cloud Computing Service Model

From the service point of view, current mobile cloud computing service providers and their services customers (i.e., mobile devices) are clearly defined. Most existing computing models are similar to the traditional "client-server" type of service models. Serval issues for the existing MCC services are explained and the expected transition characteristics are also discussed given as below.

Symmetric MCC Service Model: Most current MCC service models are in the asymmetric fashion. As examples presented in Table 2.2, mobile devices are usually considered as clients of cloud services. The service (e.g., computing and storage

services) direction is mainly one directional, i.e., from cloud to mobile devices. With the increasing capability of mobile devices, mobile devices can also collaboratively execute the applications tasks. Moreover, the virtualized environment should provide intelligent feedbacks to physical devices to adjust their behaviors or actions in order to provide better virtualized services. This virtualization-feedback loop model demands a symmetric MCC service model, i.e., both mobile devices and the virtualized cloud are service providers as well as clients.

Personalized Situation Awareness: In the current complicated mobile cloud environment, data sources could be diverse, e.g., mobile device, environment, or social network. Sometimes, single data source is not sufficient for supporting MCC applications in the cloud; moreover, data collected from heterogeneous network might be unstructured or unclassified. For example, in the physical world, there could be multiple networking interfaces and services that are available to a user's device, e.g., wireless sensor network, social network, vehicular network, personal & body area network, etc. cloud should be able to get data from different source network and then cluster them together to make the data structural and readable in the future. Thus, more work is expected to construct situation awareness services that can be personalized according to individual users in the virtual environment.

User-Centric Trust Model: Most current cloud trust model is centralized, i.e., all mobile entities need to trust the cloud service provider. Storing private data in the cloud environment is a big hurdle for most of mobile cloud applications. It is desirable to establish a distributed or decentralized trust management framework within the virtualized cloud system to address the privacy concerns of mobile users. In the physical world, the virtualized resource could be hosted in either public or private clouds that are tailored according to users preference. This requirement demands to transfer current centralized cloud to into a distributed or decentralized fashion. For

example, including mobile users' computing and storage resources into the mobile cloud infrastructure without requiring (or even allowing) administrative privilege can significantly reduce the privacy concerns from mobile users.

3.3 Cloud Security Approach Design

Here, I will describe two major design requirements that should be considered to establish a secure cloud networking environment.

1. Robust Network Architecture Design. Before building the cloud system, a robust security system design is highly desired. The following criteria should be followed when designing the cloud network architecture.

- Network isolation should be provided for multiple purposes. For example, data networks should be separated from the management network because it is not secure to make users have privilege to access the cloud management network. Moreover, different networks needs to be separated from each other physically by using different network devices, e.g., switch, or virtually by deploying the network virtualization technology, such as VLAN, GRE tunnel, etc.

- The system should allocate sufficient resources based on the usage of system components. For example, storage network usually should be allocated with more bandwidth than management network because only control messages are sent over management network while Gigabit-sized virtual machine images may be transmitted over the storage network. Besides network resources, host resources should be also considered. For example, the rabbitMQ, i.e., the message queuing system, should be allocated with better resources due to its higher processing workload than other servers, otherwise, it will become the bottleneck of the whole system.

- They system should be enabled with High Availability (HA), e.g., redundant or backup, to avoid the single point (link) failure. The HA can be reflected in either network or host perspective. It is recommended to enable the HA for the services that especially can be directly accessed by users.

The challenge of building a robust network architecture is that there can not be a perfect system design, which means, system architect or administrator can only design a near perfect system and always have other security solutions to prevent the system from being impact by any possible malicious behavior.

2. Intrusion Detection and Prevention System (IDS/IPS). An intuitive solution to address the cloud security issues is to deploy an Intrusion Detection and Prevention System (IDS/IPS), e.g., Snort [5], Suricata [6], etc. Detecting and alerting natures of IDS solutions demand the human-in-the-loop to inspect the generated security alerts, which cannot respond to attacks in a prompt fashion. Recently, the Software Defined Networking (SDN) technologies provide a programmable networking environment, which enable the Intrusion Prevention System to become a key research area in the cloud automated defensive mechanism. In general, the IPS can be constructed based on IDS. For instance, Snort can be configured as inline mode and work with a common firewall system, e.g., Iptables, to implement the IPS in the cloud networking environment [56]. However, there are several issues in the Snort+Iptables based IPS system, and our presented solutions target at addressing these issues:

- *Latency*: The IPS detection engine usually uses a buffer to queue incoming packets for inspection purpose, and a packet will be dropped when the incoming packets exceeds the buffer's capacity. This mechanism ensure the IPS for packets inspection and possible blocking actions on each network packet. IPS usually consumes more cloud system resources compared to IDS, and it also increases

the packet delivery delay due to the packets inspection procedure.

- *Resource Consumption*: Enabling new services in the system will consume more resources and downgrade the system performance. For the service that is highly interactive with all the network traffic generated in the cloud virtual networking system, resources utilization becomes very critical since the security services availability depends on it. Under the same hardware resources, the one with better processing capability, e.g., detection rate, has better resources consumption performance.

- *Network Reconfigurations*: Programmable virtual networking system in the cloud environment provides the IPS a flexible way to reconfigure the virtual networking system and provide a secure traffic inspection and control. How to incorporate the Deep-Packet Inspection (DPI) with fine-grained traffic control in the cloud virtual networking environment to reduce the intrusiveness to normal traffic is a key research challenge.

- *Service Provisioning*: IDS/IPS are generally provided and administrated by the cloud system administrator to guarantee the security of the system. However, to guarantee the privacy of the users' traffic and efficiency of the security application, IDS/IPS are expected to be provided as a cloud service so that users can claim the IDS/IPS service within their virtual networking domain. The IDS/IPS service is also expected to be elastic, which means that the resources of the IDS/IPS can be dynamically adjusted to meet the demand of traffic.

Chapter 4

MOBICLOUD: A GEO-DISTRIBUTED MOBILE CLOUD COMPUTING PLATFORM

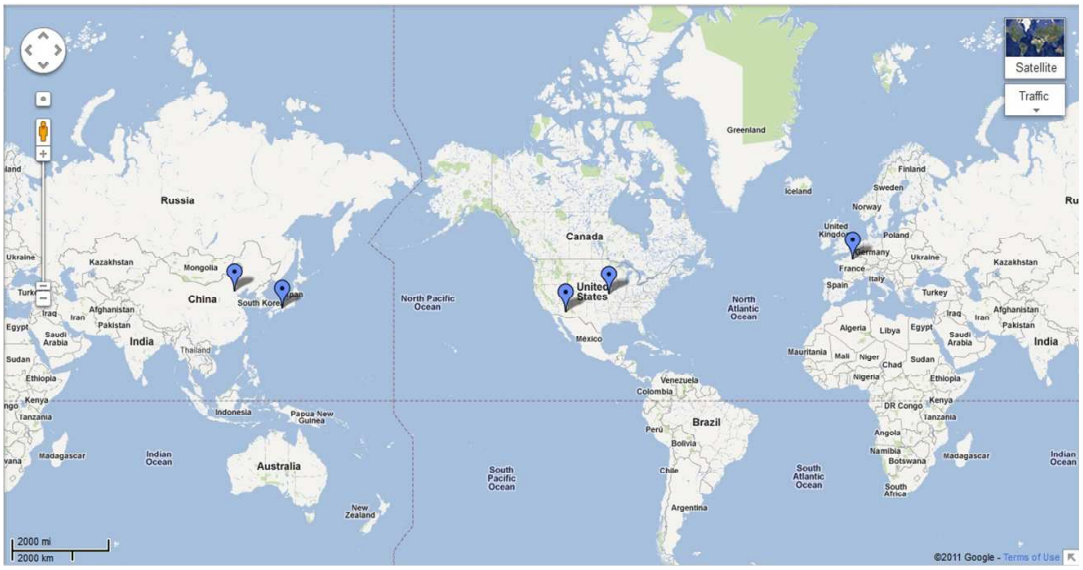


Figure 4.1: MobiCloud Resource Distribution Map

Motivated by the research challenges and objective discussed in chapter 3, it is essential to propose a cloud provisioning platform by considering mobile users and experimenters to solve the problems raised by the emergence of the popular geo-distributed cloud systems. I propose the MobiCloud, a geo-distributed MCC resource provisioning system. The major contributions of this work are summarized as below. I design the MobiCloud framework, define the service model, and propose a novel extension model CaaS. Then, the components are implemented and a geo-distributed infrastructure is established. A concrete example on how to experience the MobiCloud system is given to better understand what MobiCloud can provide. MobiCloud is a geo-distributed MCC service provisioning platform including elastic

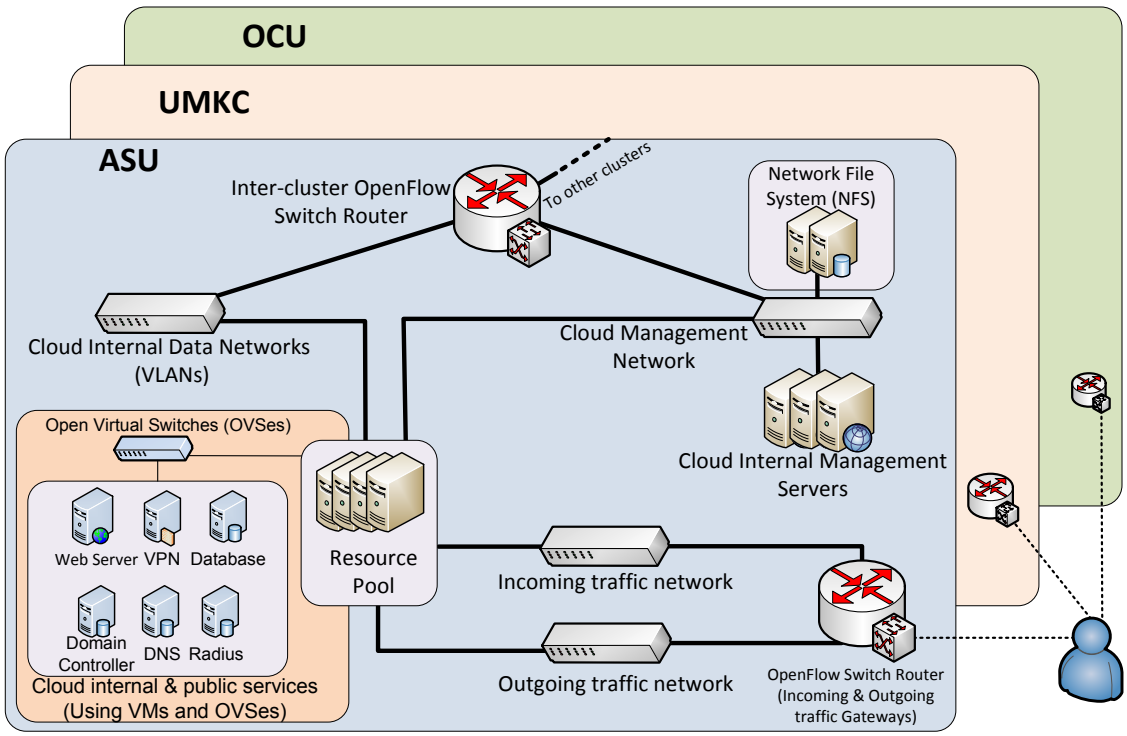


Figure 4.2: MobiCloud Architecture Design

computing, secure storage, and layer-2 and above networking capabilities. This platform is currently connecting three sites located at ASU [1], UMKC [2] in the US and OCU [3] in Japan, respectively. Several sites in Paris and Beijing are under establishment to be joined to the MobiCloud system. A Cloud resource distribution map can be seen in Fig. 4.1.

4.1 System Components

The prototype of the proposed MobiCloud system has been established and can be found in Fig. 4.2. I partition the MobiCloud system into different types of components including computing, storage, administrative and networking ones.

Computing Component. The computing component is the entity providing

[1] Arizona State University

[2] University of Missouri – Kansas City

[3] Osaka City University

the computing resources, i.e., Cloud host. The major difference among different computing components depends on the virtualization technology being adopted. The virtualization in MobiCloud is based on XEN that has an impressive scalability and efficiency. A Cloud system can provide more logically separate resources upon the virtualization layer. Usually, Cloud resources in one domain are grouped into a resource pool that always has at least one physical node known as the *master node*. Other physical nodes that join existing pools are described as *slave nodes*. Only the master node exposes an administration interface and forwards commands to individual slaves as necessary.

Storage Component. Storage stores all the resource images and users' data. Resource is prepared by cloning resource templates that are stored in the storage repository. I choose to establish a remote storage repository, Network File System (NFS), to manage the storage of resources in the Cloud system. An NFS storage server is connected to the computing server via a switch that greatly increases the scalability of storage.

Administrative Component. Dedicated physical servers are for administrating resources and monitoring the network traffic within and across the domains. There is also a set of internal functional servers serving different administrative purposes, i.e., web service, DHCP, DNS, authentication service, database service, VPN, etc.

Networking Component. Control plane and data plane are isolated based on the multi-network design of the MobiCloud. In Fig. 4.2, there are 4 networks in each cluster. The incoming and outgoing traffic switches isolate the control traffic (i.e., resource access, OS update, and package download) coming into or going out of the MobiCloud Gateway. The data network switch is a managed switch with VLAN supported that enables VMs from different physical servers to reside in the same virtual domain. Lastly, the Cloud management network is connecting the internal

management and monitoring server and NFS. Each Cloud server is installed with an Open vSwitch with which the data traffic between two VMs in the same physical server does not need to go through the physical data network switch out of the Cloud host.

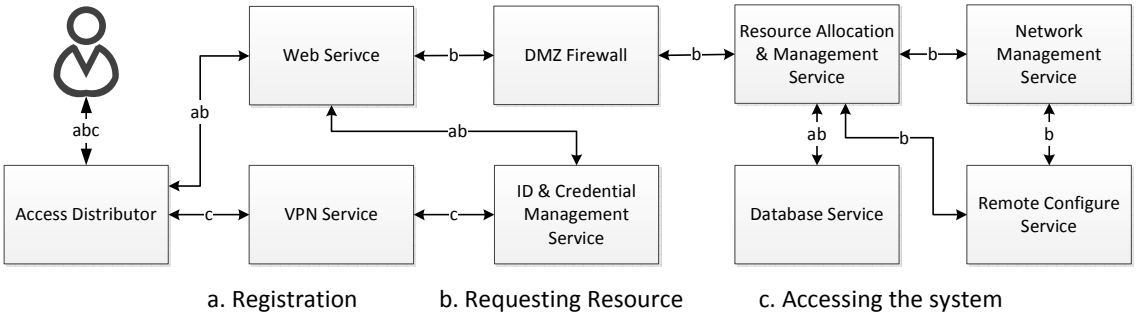


Figure 4.3: Processing flows in MobiCloud

4.2 Implementation Flow

After the framework is designed and the infrastructure is established, I implement the system to provide resource provisioning services to users. The implementation flow is defined in terms of three major aspects, registration, requesting resources and accessing the system. These three procedures can be found in Fig. 4.3.

- **Registration**. Anyone who wants to use the MobiCloud system must have a valid account. Users have to register an account by visiting the MobiCloud portal. After users create an account, the account credential will be automatically stored in the ID and Credential Management Service, which is implemented by the Kerberos based Active Directory, and dedicated Databased Service.

- **Requesting Resource**. After users' credentials are authenticated, users are authorized to request resources at the Resource Allocation & Management Service. Although all services are hosted in the MobiCloud private domain, I still introduce a DMZ to enhance the security of the Cloud resources. The Database

service stores all related information of newly created resources, and the Network Management Service prepares and configures the network attributes of the resources. So far, the network Management Service consists of DHCP and Dynamic DNS. Users are also able to configure the network attributes of resources by themselves, which is the reason why the Remote Configure Service is developed.

- **Accessing Resource**. When users are about to access their allocated resource, they have to connect to the VPN server through the gateway first. The VPN server authenticates the users' accounts at the ID&Credential Management Service where users' account credentials are stored. After connecting to the VPN, users' mobile devices are assigned with the MobiCloud private internal IP address and are free to access their resource by using the corresponding domain name that was assigned for each VM.

4.3 System Extension Model: CaaS (Cluster as a Service)

MobiCloud is not only designed for general mobile users but also for developers and researchers who may want to have more control on the Cloud system. Thus, I propose a novel system extension model called Cluster as a Service (CaaS), with which users can setup their own cluster joining the MobiCloud umbrella and easily turn their roles from service consumers to service providers. You do not have to have all equipment (as shown in Fig. 4.2), but some necessary ones with which everything needed in the cluster can be easily reproducible. I package all functional servers (i.e., DHCP, DNS, Database, etc.) into Virtual Appliances (VAs), so that users just simply import the image files and turn them into functional virtual servers. To configure the network, i.e., VLAN creation and tunnel setup, a set of scripts running on XEN Dom 0 are prepared.

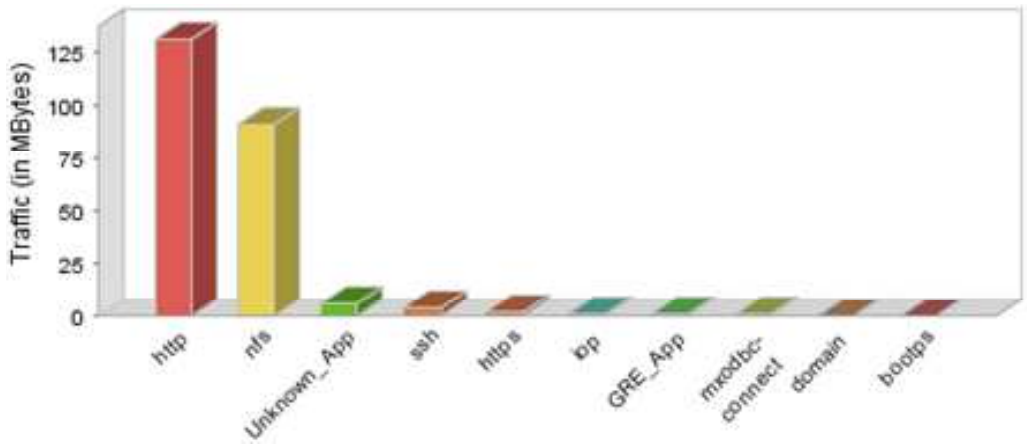


Figure 4.4: NetFlow Monitoring Top Protocol

There are two types of system extension configurations, *full-cluster* and *semi-cluster*. *Full-cluster* exactly duplicates the cluster at ASU. Physical Xen servers, general switches, an OpenFlow switch, and a public IP address are needed. The *semi-cluster* provider does not even need to have the OpenFlow switch and other physical devices, but a single PC supporting Intel-VT or AMD-V with access to the Internet. A dynamic DNS configuration can help the cluster to be accessed from public. To establish the layer-2 connection (GRE tunnel) to other clusters, users just need to use the Open vSwitch residing in the Xen Dom 0. With this CaaS extension model, the MobiCloud system becomes more extensible and scalable.

4.4 System Resource Monitoring

As system administrators, to better monitor the performance of both network and Cloud hosts in the MobiCloud and track the ongoing and potential issues existing in the system, I introduce a multi-layers monitoring mechanism, that is enabled by sFlow and NetFlow [42]. NetFlow provides limited visibility, focusing on layer-3 network connections, while sFlow provides comprehensive visibility into network and system resources needed to manage performance in virtualized and Cloud environ-

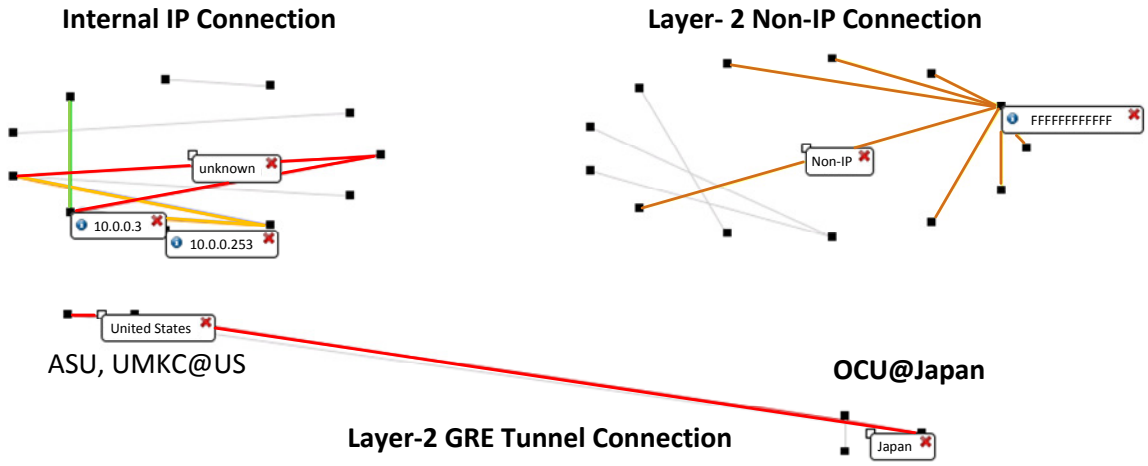


Figure 4.5: sFlow Connection Circle

ments. In each Open vSwitch in Dom 0 of each Cloud server, I enable not only sFlow at the switch level but also at the host level, which means not only can the traffic be monitored, but also the host performance (i.e., CPU utilization, memory usage, virtual disk I/O, etc.) can be monitored.

From the analyzer of both NetFlow ManageEngine and sFlow Flowtrend [8, 7], network parameters, e.g., top conversation, top connections, most popular protocols, etc., can be monitored. Fig. 4.4 is the NetFlow captured top protocol statistic chart for one hour at the ASU site. The chart shows that the top two protocols of HTTP and NFS correspond to two spikes in the website and read activity on the virtual machine. The third top traffic is the unknown application including communication between Open vSwitch and its controller and other internal control messages. The following traffic, which is almost control traffic including ssh, https, GRE tunnel and bootp, etc., does not consume much bandwidth.

Besides layer-3 network monitoring, the sFlow is able to inspect the connection relationship from the following three aspects shown in Fig. 4.5: internal connections, external connections and non-IP connections. The left-top part of Fig. 4.5 shows the internal connections that represent the internal topology at the ASU site where the

connections between the Cloud server master node(10.0.0.3) and the storage server (10.0.0.253) can be seen. The top-right part shows the layer-2 Non-IP connections including unicast and broadcast. The bottom part shows the external connections that indicate the inter-cluster connection among all three sites.

4.5 Experiencing the MobiCloud

In this section, I present the implementation of the MobiCloud through a simple example of its use. Through the developed web interface, users are free to realize all operations (register, request resource, and access resource) I mentioned in the previous section.

Table 4.1: Single VM Creation Specification

VM Host Configuration	
VM name	Define the name of the VM.
OS Type	Specify the Operation System of VM. Since I use Xen Hypervisor, which supports a wide range of guest OS including Windows, Linux, Solaris, and various versions of the BSD operating systems.
Hard Disk Size	Specify the size of the virtual hard disk on each VM. There are several options ranging from 8GB to 128GB.
Ram Size	Specify the size of the memory. The range of memory can be selected from 128MB to 4GB.
VM Network Configuration	
NIC configure	Specify the IP address, netmask and default GW.

4.5.1 Single VM Creation

I design three types of users to be registered. Each type of user has its own privilege. 1. Free users: multiple users share one VM; 2. Basic users: each user is

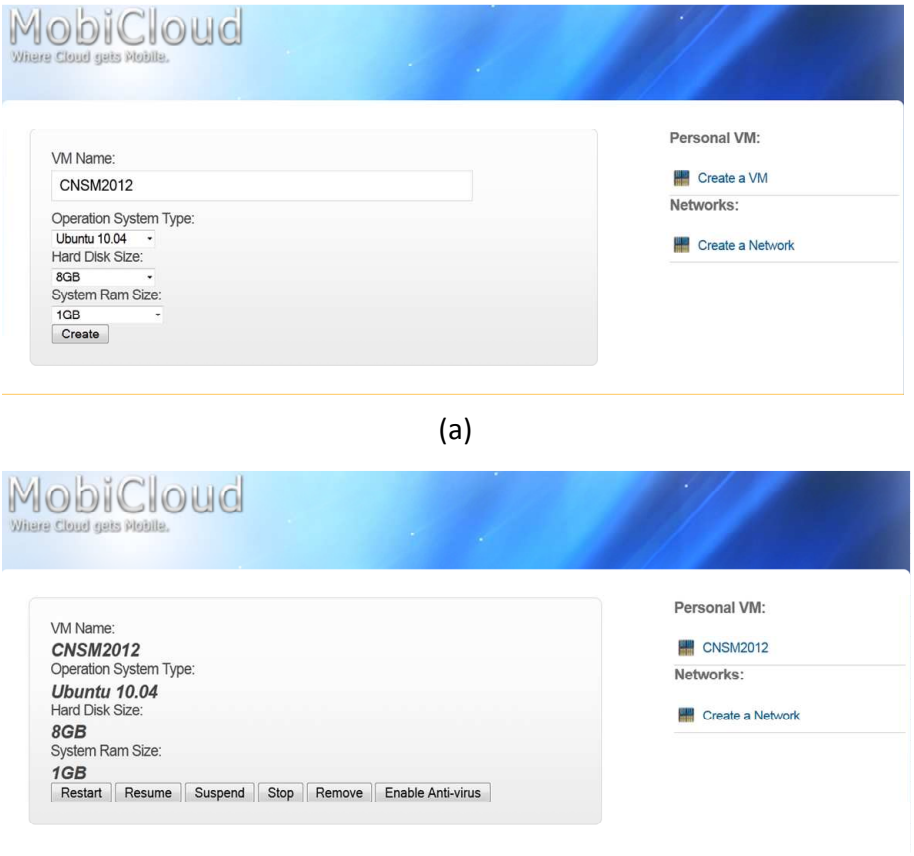


Figure 4.6: Single VM Creation

provided with a dedicated VM with limited resources; 3. Premier users: each user is provided with one or multiple VMs that are configurable. After registering an account at the MobiCloud system, users are free to claim resources from the Cloud servers. Here I use a premier user account, which has full privileges, to demonstrate how a single VM creation is processed. The interface is shown in Fig. 4.6(a). To create the single VM, there are several fields to be configured by users that can be referenced in Table 4.1.

After a user specifies all the parameters in the corresponding fields, VM will be created at the backend side. One of the advantages of MobiCloud over current existing platforms is that the response time of the resource preparation is swift. Because the

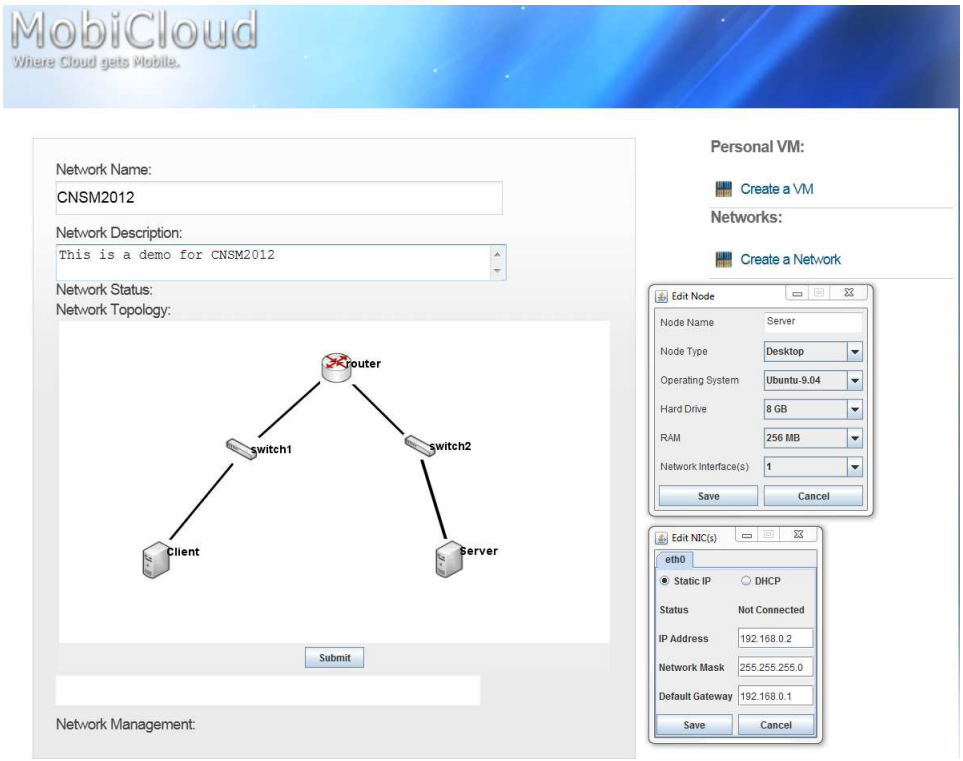


Figure 4.7: Virtual Network Creation

VM template is already prepared, what the system only needs to do involves two steps: 1) fast copy the VM template and 2) configure the VM. When the VM is done being prepared, users can use the management interface, shown in Fig. 4.6(b), to manage their VM. There are several options including restart, resume, suspend, stop, and remove. Each VM will be assigned with a unique domain name, in the format of *username.mobicloud.asu.edu* that is registered in the DNS system. A Single VM can also be set as a proxy of a mobile device to enhance the its capability. For example, users can set their VMs as the proxy with an anti-virus function enabled, which greatly enhances the security of the mobile device.

4.5.2 Virtual Network Creation

Besides single VM creation, users are also allowed to claim multiple VMs connecting as virtual networks. The interface can be seen in Fig. 4.7. Users can simply click,

drag, and drop on the canvas to create their desired network components and connect them. There are three major components in the network creation canvas, desktop, switch, and router. Users can configure the component in a pop-up window. Beside the virtual hardware configuration (i.e., CPU, memory, harddisk, etc.), network information can also be configured, including creating virtual interfaces, IP addresses, netmasks, default gateways, etc. I use a switch component to represent VLAN to partition the network. Users can drag lines to connect two or more components (i.e., desktop or router) through a switch, which means those interfaces are in the same VLAN and isolated from others. The router is the last component with routing functions enabled by a software-based pre-installed router (Vyatta [11]). After user confirms the topology by clicking the submit button, a summary page, indicating all hardware and network information of VMs is presented. When the preparation is done, a resource page is returned with the detail of the resource created. Each VM is assigned a unique domain name that is similar to the single VM creation case. But each user can have multiple VMs, so that I choose the domain naming scheme in the format of *username-networkname-number.mobicloud.asu.edu* to identify the VM. For example, user *cnsm2012* creates a network named *mynetwork* including three VMs, the domain name assigned to the first VM is *cnsm2012-mynetwork-1.mobicloud.asu.edu*.

4.5.3 Accessing the Resources

After the resources are prepared, users are free to access the resources via either fixed terminal (i.e., desktop) or a mobile device (i.e., Smartphone, tablets, etc.). There is a VPN server running behind the MobiCloud firewall to provide access to resources for users. Resources (VMs) are enabled with SSH and VNC service that can be used to access the VM. Besides the fixed device, users are also able to access the resources by using the mobile devices built-in function to access VPN (shown in

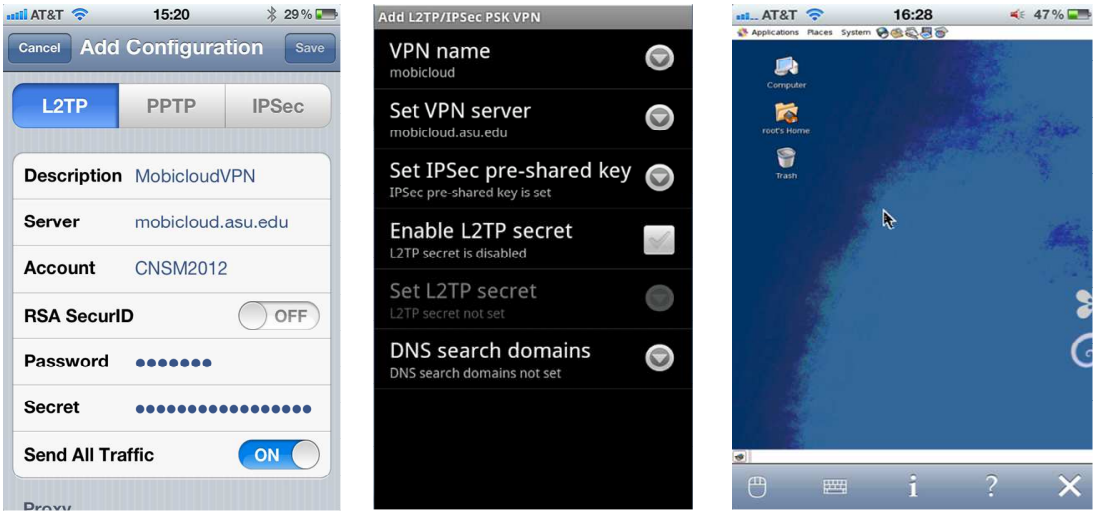


Figure 4.8: Accessing VPN on (a) iOS and (b) android, accessing VM via VNC on iOS

Fig.4.8 (a) (b) for iOS and Android OS). After connecting to the VPN, the device will be assigned with a private IP address that is in the same subnet with resources. Fig. 4.8(c) is an example of accessing the virtual desktop of resources from an iOS smartphone.

4.6 Performance Evaluation

MobiCloud is always aiming to provide resources with expected user experience. Thus, two metrics are mainly considered to benchmark the system performance, resource preparation time and available bandwidth among resources. To make the performance evaluation result approach the real capacity, I test both metrics by using the Cloud server and network device with about a 70% load of the full capacity. Since VM is the major type of resource to be provided in the system, the preparation time of VM is directly related to user satisfaction. Thus, to evaluate the VM preparation performance, I choose the five most popular used guest OS in the system that are CentOS 5.5, Ubuntu 10.04 (with and without desktop GUI), Windows 7, and

Windows XP. The preparation time can be divided into two aspects, creation and reboot. Creation time refers to the time from clicking the create button to returning the confirmation page. The VM preparation includes not only the VM boot procedure, but also some back-end configuration steps (e.g., find and clone the template, domain name registration, and etc.). Besides the first time VM creation, I also evaluate the VM reboot time that indicates how fast the user can refresh the resources. From Fig. 4.9, the VM preparation time varies due to different guest OS and are all in the acceptable even decent range. The reason why creation time is more than reboot time is that creation needs more configuration steps I discussed above.

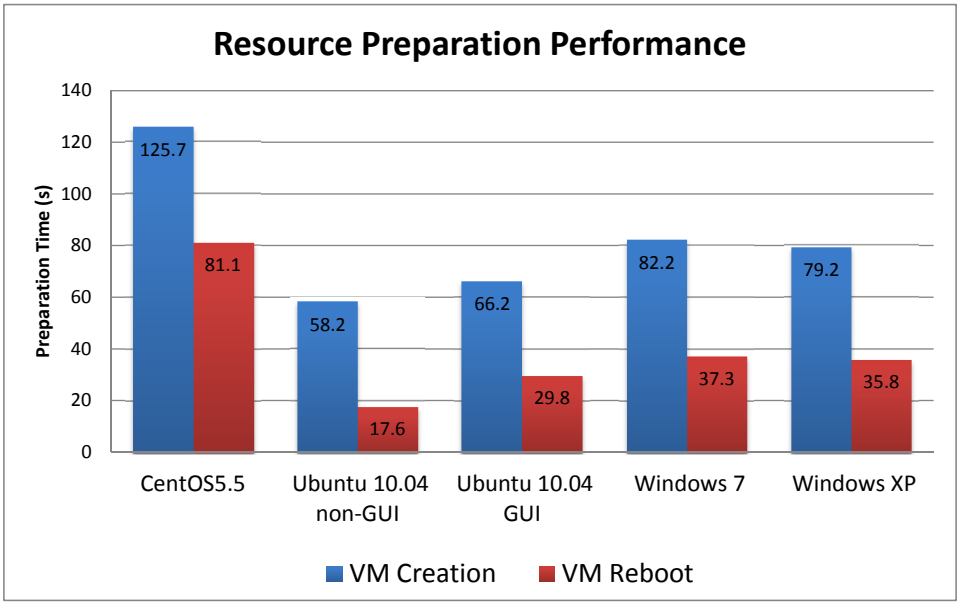


Figure 4.9: VM Preparation Performance

Users also need their resources to be distributed in the system to construct their specific application or service. Thus, I test the bandwidth between resources at different scenarios (*in-cluster mode* and *in-server mode*), to see how freely resources can communicate with each other. Currently, the geo-distributed system does not have a large scale, the bandwidth between different clusters varies due to different link conditions and is not representative. So I am not considering the bandwidth

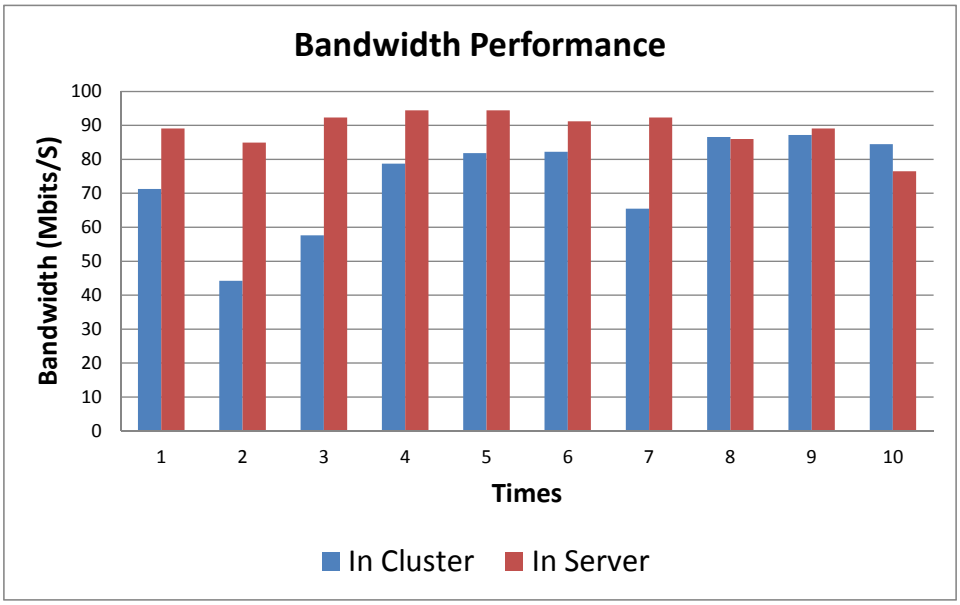


Figure 4.10: Bandwidth Performance

among clusters, but just the two cases mentioned above. *In-cluster mode* tests the bandwidth between two VMs in the same cluster but different physical servers. These two VMs are usually in the layer-2 connection through a physical switch. *In-server mode* tests the bandwidth between two VMs in the same physical server. They are connected internally through the Open vSwitch. I test the TCP bandwidth for 10 times for each mode and the result can be found in Fig. 4.10. From the result, the bandwidth performance between resources in both modes does not show much difference. The average bandwidth for the *in-cluster mode* and *in-server mode* is 89.1 and 74.0 Mbits/s, respectively, which is sufficient for most normal application or experiments.

Chapter 5

CONSTRUCTING VIRTUAL NETWORKS IN A GEO-DISTRIBUTED PROGRAMMABLE LAYER-2 NETWORKING ENVIRONMENT (G-PLANE)

G-Plane is primarily based on MobiCloud system and is more focused on the virtual network manipulation and case study in a geo-distribtued cloud networking environment.

5.1 Virtual Network Construction

Resources in G-Plane are mainly in the form of VMs, connected as a network. The resource network can be created by different configurations due to different requirements: 1) a single physical server, 2) multiple servers with one cluster (servers in the same cluster are connected through a layer-2 physical switch), and 3) multiple servers belong to different clusters.

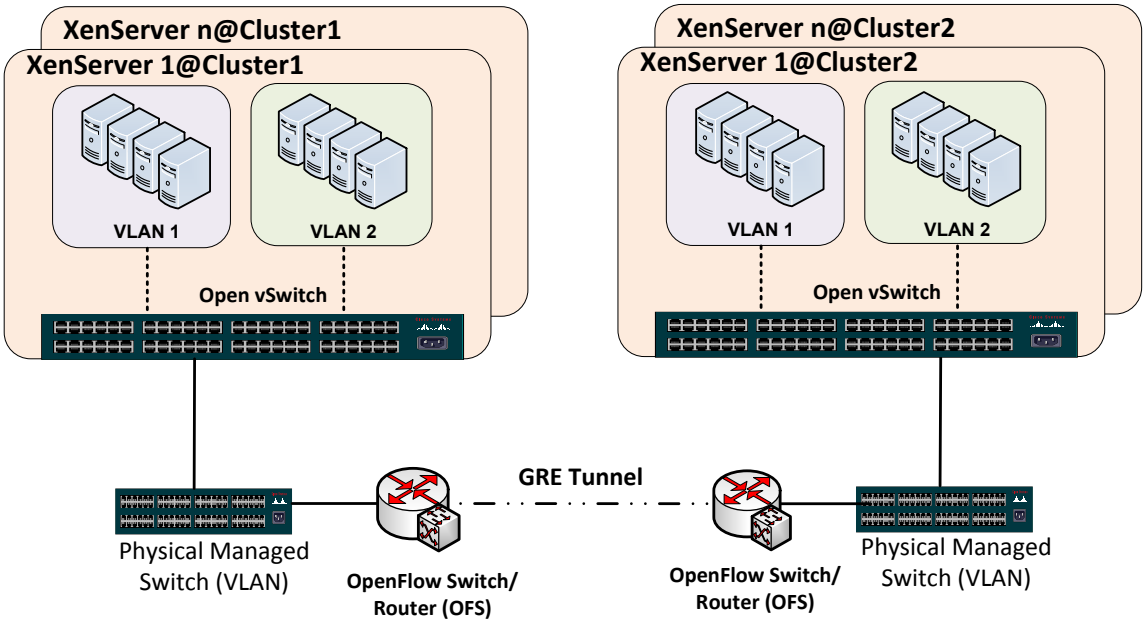


Figure 5.1: Intra- & Inter-Domain Network Architecture

5.1.1 Intra-Cluster Network Creation

Intra-Cluster means that there is always a native layer-2 connection among all resources within the the same cluster. To create a virtual network within the same cluster, VLAN technology is deployed. As I previously mentioned, it is inefficient to forward packets through the managed switch from one VM to another one in the same physical resource provisioning server. Therefore, each XenServer has an internal Open vSwitch enabled to handle traffic inside the physical server as shown in Fig. 5.1. Open vSwitch is designed to enable massive network automation through programmatic extensions, while still supporting standard management interfaces and protocols (e.g. NetFlow, sFlow, RSPAN, ERSPAN, CLI, LACP, 802.1ag, etc.). Open vSwitch can operate as a software-based switch running within the hypervisor (Xen Dom 0), in which many security control functions can be implemented. With Open vSwitch enabled, a packet sent from one VM to another one within the same physical server does not need to be exposed out of the physical box. When a virtual network is created within the same cluster but across different physical servers, a packet sent from one VM to another one on a different server should go through the physically managed switch by enabling trunk ports. The virtual network containing multiple VMs in different physical servers is simply created by assigning the same VLAN ID so that it is virtually isolated from other resources.

5.1.2 Inter-Cluster Network Creation

To enable provisioning of virtual network across clusters in G-PLaNE, I establish a layer-2 GRE tunnel among each site by deploying OpenFlow switch running on top of NetFPGA box. After a layer-2 tunnel is established, VLAN can function well upon a layer-2 tunnel since it is a 2.5 layer technology strictly speaking. Although there are

some options to establish the layer-2 tunnel, I choose the OpenFlow solution since it is user-centric and can be easily extended due to its programmability. OpenFlow is an open standard that enables researchers to run experimental protocols. In a classical router or switch, the fast packet forwarding (data path) and the high level routing decisions (control path) occur on the same device. An OpenFlow switch separates these two functions. The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller, typically a standard server. The OpenFlow switch and controller communicate via the OpenFlow protocol, which defines messages, such as packet-received, send-packet-out, modify-forwarding-table, get-stats, etc. The data path of an OpenFlow switch presents a clean flow table abstraction; each flow table entry contains a set of packet fields to match, and an action (such as send-out-port, modify-field, or drop).

5.2 Enabling Additional Research Capabilities

With the virtual network provisioning service provided, users are able to expand their own development and research work. There are some representative ones listed below:

- **Routing Algorithm & Protocol Design**. Because I provide resources at the IaaS level, users have more flexibility and privilege on their claimed resources. Each virtual machine can be easily turned into a general virtual router or OpenFlow switch by installing a software-based routing suite. Thus, experimenters and researcher can have a real network environment in any possible topology they expect. With the capability of modifying the entire routing protocol or algorithms module, research on routing protocols and other networking layer mechanisms can be investigated, tested and measured in a real networking environment.

- **Distributed Application Development**. G-PLaNE users can use any IP connected device to communicate with their VMs supporting a variety of OS. So it enables developers to develop Cloud based applications on both the Cloud side and the mobile device side. Moreover, after the virtual network resources have been provided, more complicated networking and distributed applications can be developed and tested in a real distributed networking environment.

- **Distributed Security and Privacy Model**. There is some research work [63] deploying a Cloud to help users keep from viruses and enhance security. Cloud based anti-virus engines have been emerging and are being studied. If users are able to control a virtual network rather than a single VM or multiple isolated VMs, then an advanced model can be investigated. For example, different anti-virus engines can be placed onto different VMs and a centralized control VM running some algorithm to coordinate them, is expected to enhance efficiency to some extent. Generally speaking, any security and privacy research issues can be investigated in a real distributed environment that G-PLaNE provides.

5.3 Case Study: SeRViTR

The G-PLaNE system is designed to be able to support research in a virtual networking environment. In this section, I present a case study on how to utilize the G-PLaNE system to generate and manage virtual domains between geo-distributed platforms.

5.3.1 SeRViTR Architecture Overview

A trustworthiness model for future networks called the Virtual Trust Routing and Provisioning Domain (VTRouPD)[40] has recently been proposed. Trustworthiness can be defined in many facets. From the viewpoint of a network, it means

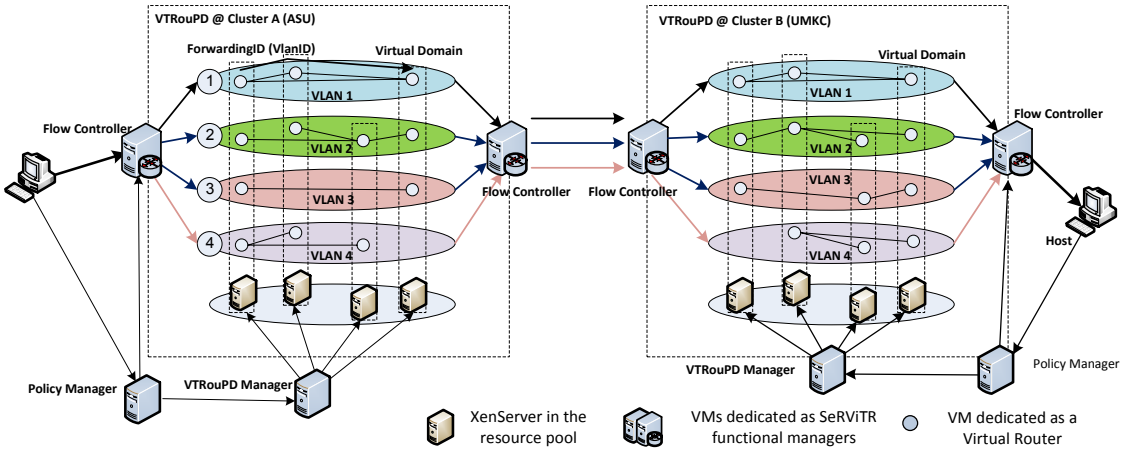


Figure 5.2: SeRViTR Architecture Overview

routing information must be confidential, secured, and protected, whereas from the service provider's perspective, trustworthiness of service assures that the service is safe and exclusive of anonymous users. Due to such variety of trustworthiness, a network should be sliced into multiple virtual domains that are isolated from each other. A VTRouPD is constructed by a collection of networking resources including routers and switches based on virtualization techniques; e.g., constructing virtual managed domains through tunneling and VLAN technologies. Within one or multiple VTRouPDs, I can further create flow or application level virtual routing domains that are denoted as μVTRouPDs[40]. In my recent work [52], the intra- and inter-domain policy and trust management between VTRouPDs within the SeRViTR architecture is discussed.

Fig.5.2 presents an overview of the SeRViTR system architecture. The figure presents two VTRouPDs that are geo-distributed clusters at two places: Arizona State University and the University of Missouri - Kansas City. Each *VTRouPD* is a overall domain with the administrative control, and it may contain multiple *Virtual Domains*. Here, I will only give a very high level description on SeRViTR components

along with their relations; The detailed functions of the SeRViTR system can be found in [52].

Packet flows to *Virtual Domain*s in the SeRViTR architecture are differentiated based on trustworthiness policies, and the *Policy Manager* plays a key role on setting up policies along with establishing *Virtual Domain*s. First, it sends policy rules to the *VTRouPD Manager* to make a request of *Virtual Domain* creation or deletion. Second, it communicates with the *Flow Controller* about flow updating. A *VTRouPD Manager* manages the information of physical routers within the *VTRouPD*, and it is responsible for the creation or deletion of *Virtual Domain* as well as resource management. The *Flow Controller* is placed at the edge of *VTRouPD*s, and it is in charge of forwarding flows to correct *Virtual Domains* based on the policy.

5.3.2 SeRViTR Deployment on G-PLaNE

The core idea behind SeRViTR is constructing the network by setting up policies to assure secured routing between virtual domains across multiple sites. To achieve such a goal, SeRViTR has to be deployed in a virtual networking environment that is well supported by the G-PLaNE system.

The G-PLaNE system allows the cluster to be scalable. Take the SeRViTR clusters at ASU and UMKC for example, presented in Fig. 5.2; here, two *VTRouPD*s are created at different clusters. I have established a layer-2 GRE tunnel between the two *VTRouPD*s through OpenFlow switches. Within the cluster at each site, the G-PLaNE resource pool contains physical XenServers where VMs are created. Recall that all VMs can be created and deployed as any form of functional entities; thus, I can customize VMs as dedicated SeRViTR functional components as well as virtual routers. Particularly, *VTRouPD Manager*s, *Flow Controller*s, and *Policy Manager*s are implemented on VMs created on one XenServer from the resource pool.

The virtual routing domain is a vital constituent part in the SeRViTR architecture design, and it requires good isolation as well as scalability when constructing virtual networks. With the G-PLaNE system data network switch, which is VLAN supported, VMs can be grouped into different virtual domains by tagging VLAN IDs, and the ones which have been used can be queried through the database of the G-PLaNE system. Fig. 5.1 shows a high level virtual domain creation by grouping VMs into the distinct VLANs. Particularly, consider cluster A at ASU in Fig. 5.2, where four XenServers are reserved from the resource pool for creating virtual routers. On each XenServer, I can customize an arbitrary number of VMs as dedicated virtual routers by deploying a routing suit (i.e. OpenFlow switch, Quagga, etc) on it. Now, consider the SeRViTR virtual domains. The *PolicyManager-VM* sends a request along with a trustworthiness policy to announce the creation of a new virtual domain to the *VTRouPDManager-VM*. The *VTRouPDManager-VM*, in turn, communicates with every XenServer that is reserved for creating VMs as virtual routers and sends out an unused VLAN ID. When creating VMs, the XenServer will add this unused VLAN ID through its API. Therefore, VMs (Virtual Routers) created by distinct XenServers can be put into the same VLAN according to the unique VLAN ID. The data network switch will enable the intra- or inter-virtual domain communications.

Chapter 6

MOBILE CLOUD USER-CENTRIC MODEL

After a comprehensive literature review on the existing MCC services in terms of computing, storage, security, context awareness, etc. I firstly summarized and defined three types service model based on the existing MCC services, and proposed a new user-centric MCC service model called Representor as a Service (RaaS).

6.1 Current Mobile Cloud Service Models

Current Internet clouds have been broadly classified in three-type service models: Infras-tructure-as-a-service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). They are classified according to the layers of virtualization. However, due to the involvement of both CPS and CVS, the MCC's service models are more appropriate to be classified according to the roles of computational entities within its service framework, where the classification of MCC service models can use the roles and relations between mobile entities and their invoked cloud-based resource provisioning. Based on this view, existing MCC services can be classified in three major models: Mobile-as-a-Service-Consumer (MaaSC), Mobile-as-a-Service-Provider (MaaSP), and Mobile-as-a-Service-Broker (MaaSB). These MCC service models are illustrated in Fig. 6.1, in which arrows indicate service processing flows from service providers to service recipients.

MaaSC is originated from the traditional client-server model. It was enhanced by introducing virtualization, fine-grained access control, and other cloud-based technologies at the initial stage. Mobile devices can outsource their computation and storage functions onto the cloud in order to achieve better performance, and more

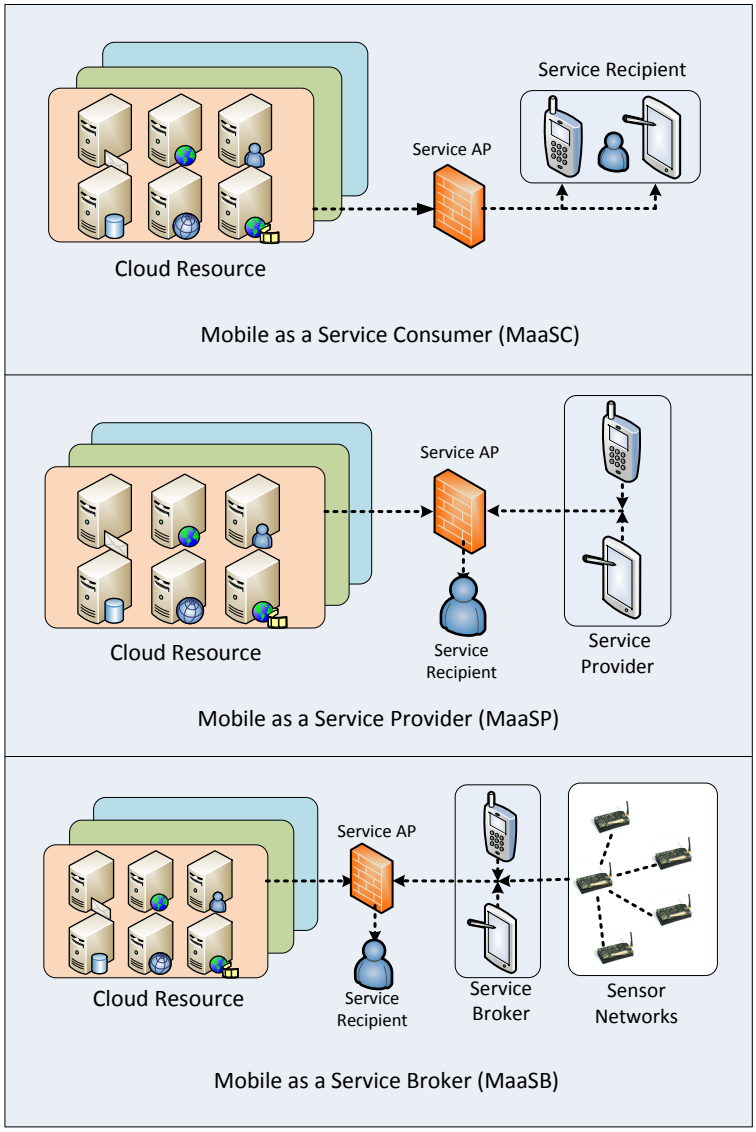


Figure 6.1: Current Service Models of MCC

application capabilities. In this architecture, the service flow is in an one-way fashion. It is from the cloud to mobile devices and mobile devices are service consumers. Most existing MCC services fall into this category.

MaaSP is different from MaaSC in that the role of a mobile device is shifted from a service consumer to a service provider. For example, with on-board sensors, i.e., GPS module, camera, gyroscope, etc., mobile devices are able to sense data from the devices and their neighboring environment, and further provide sensing services

to other mobile devices through the cloud. In Fig. 6.1, consumers receive services provided by both the cloud and mobile devices. The types of services provided by mobile devices are diverse based on their sensing and processing capabilities.

MaaSB can be considered as an extension of MaaSP, where MaaSB provides networking and data forwarding services for other mobile devices or sensing nodes. MaaSB is desired under some circumstances because mobile devices usually have limited sensing capability compared to sensors that are dedicated for specially designed functionalities and sensing locations. For example, mobile phones can be used to collect users' physical activities from Nike Fuelband [61]. MaaSB extends the cloud edges to mobile devices and wireless sensors. Thus, a mobile device can be configured as a gateway or a proxy providing networking services through various communication approaches such as 3/4G, Bluetooth, WiFi, etc. Moreover, the proxy mobile device can also provide security and privacy protections to their interfaced sensors.

6.2 Design Principles of User-centric Mobile Cloud Computing

Future MCC should be re-considered as a new service model, where mobile agents, i.e., both physical and virtual entities, and related resources collectively operate as mobile clouds that enable computing, storage and networking capabilities, context awareness modeling, content discovery, data collection and dissemination. To build the future user-centric mobile cloud computing based on the described concepts and requirements, mobile clouds should be shifted from the traditional Internet cloud by using the following principles:

- *Principle 1: User-centric:* MCC applications should be designed in a way that user can control their own data and activities with strong privacy and security protection. Cloud resources should be collected and allocated according to mobile applications customized for each individual user.

- *Principle 2: Service-oriented application platform:* Due to the symmetric service model, every mobile node can potentially serve as an MCC service provider, and thus service-oriented application platform is the natural choice for MCC.

- *Principle 3: Mobility efficiency:* MCC resources should be dynamically allocated and managed according to the need of mobile cloud applications. Mobility of MCC should be confined through a set of mobile cloud application constrains to maximize the efficiency using a set of system performance evaluation metrics such as availability, computing power, storage, and their spacial-temporal boundaries.

- *Principle 4: Virtual representation:* MCC maintains a trusted, reliable, and accessible virtual representation for each user. The virtualized representation can be considered as an assistant for mobile users and performs actions such as sensing a user's daily activity to build user's behavior and activity profiles, and delegate the user's activities in the virtual environment.

6.3 Mobile-as-a-Representer: A User-Centric Approach

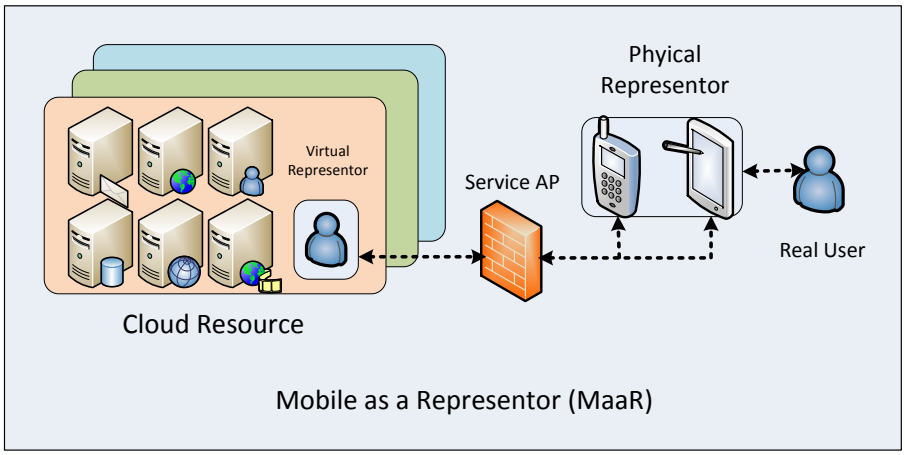


Figure 6.2: Mobile-as-a-Representor (MaaR)

6.3.1 MaaR Model Basic

Besides previously presented service models, i.e., MaaSC, MaaSP and MaaSB, and user-centric MCC design principles, I present a new user-centric MCC service model called Mobile-as-a-Representor (MaaR). The architecture of MaaR can be found in Fig. 6.2. In MaaR, each user can be represented by a virtualized entity in the cloud through his/her physical entity (i.e., mobile device). Users' behaviors and attributes can be collected from the real world (people, environment or mobile devices) in real time and sent to their corresponding virtual entities in the cloud to perform further analysis and processing. Data mining and machine learning algorithms can be used to analyze a mobile users situation and perform actions proactively. MaaR can be regarded as the next-generation MCC service model in that both physical systems and virtual systems are seamlessly integrated through virtualization technologies to provide services. In MaaR, the mobile devices and clouds are highly interactive, and as a result, the service flow can be presented as bidirectional arrows. In addition to help mobile entities execute tasks more efficiently, MaaR is able to accomplish some tasks that are impossible to be realized in current MCC architecture.

MaaR model is presented to support the next-generation user-centric MCC services and applications. A conceptual architecture of MaaR is presented in Fig. 6.3, where both CPS and CVS are integrated as a whole system. In the CPS, heterogeneous networks co-exist and all these networks can be virtualized at the CVS by performing operations including presenting, offloading, abstracting, caching, migration, etc. All data with the spatial, temporal, and correlation information from the CPS will be submitted to the CVS. Among all these three types of information, correlation information is essential in that it helps to fuse different types of data together into a well formatted one so that the CVS can further perform context awareness,

user-centric proactive and security protection tasks. For example, the sensor network carries sensed data while social network collects and generates the social relationship data. The correlation information helps the CVS to generate sensing data with social attributes (e.g., personal data that is only accessible from a specific social group, like people in the users friend list).

In MaaR, the CVS has three main types of provisioning resources: computing resource, storage resource, and networking resource. User's virtual entity is represented by maintaining seamless communication between the CPS and the CVS, which also allows for establishing multiple personalized MCC clouds due to different application purposes. A MCC application is able to control integration of CPS and CVS through a well-defined API and MCC tools. Traditional Internet cloud is one-way operational as users can only submit data from the CPS to the CVS while it is possible to allow the CVS to further control the CPS functions in a highly adaptive and dynamic fashion based on the MaaR model. Besides physical data being virtualized to virtual environment, the CVS can provide feedback and control functions in the CPS.

To enable the service-oriented application running environment, MaaR provides POEM (Personal On-demand execution Environment for Mobile cloud computing) framework [82] to achieve the user-centric MCC service running platform highlighted in Fig. 6.3. POEM is a mobile cloud application execution platform that enables mobile devices to easily discover and compose cloud resources for their applications. For mobile resource providers, they may not even know what applications and who may call their provisioned functions beforehand. In this way, the mobile application design should not be application-oriented; instead, it should be functionality-oriented (or service-oriented). To achieve these features, I can consider those Provisioning Functions (PFs) as the fundamental application components in the MaaR model, which can be composed by mobile cloud service requesters in runtime.

POEM takes a comprehensive approach by incorporating the OSGi-based [65] service-oriented architecture into the mobile cloud computing. It treats the offloading as part of service composition, and as a result, the codes (or computation tasks) are considered as services provided by mobile devices and the cloud. In this way, offloading and migration operations can be multi-directional (i.e., among mobile devices and the cloud) compared to one-directional (i.e., from a mobile device to the cloud) in previous solutions. Moreover, due to the popular Java-based OSGi framework, POEM can greatly improve the adoption of the SoA-based code reuse and composition for mobile cloud computing.

6.4 An Application Scenario based on User-centric MaaR model

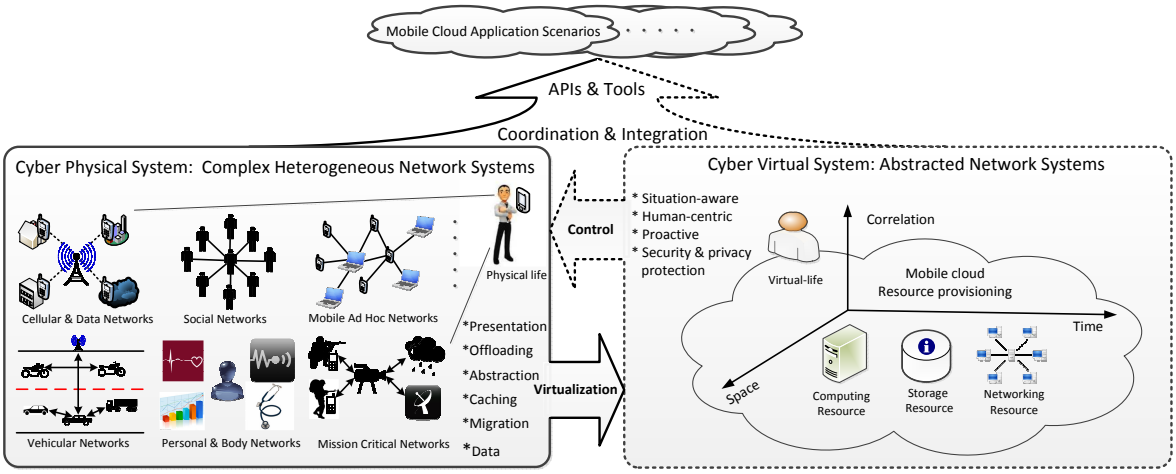


Figure 6.3: MaaR Conceptual Architecture

To better understand the proposed future MaaR model, I introduce a vehicular video sensing and collaboration application as an example. It is assumed that MaaR service modules are already equipped on many users' smartphones. When user Alice is driving, her smartphone uses onboard sensors like camera or GPS to detect her location, driving speed, and image/video captured on the road. Those collected information can be collected and virtualized into the CVS to construct a virtual

representation of the mobile device in the cloud for Alice, which is the essence of MaaR in that the virtual representor represents the real situation of the physical user. Practically, the representor is implemented through a set of software agents (i.e., OSGi bundles) on a dedicated VM allocated for Alice, where Alice has the administrative privilege on the VM to decide what data can be shared and protected (by encryption). The dedicated VM is the application holder for Alice to incorporate various data processing models and functions for security, data mining, and intelligent situation-awareness decision making that are personalized for the uses of Alice. In this model, the VM can be hosted in public or private cloud as the choice of Alice.

User Bob may want to know what is the current traffic status around the bridge 5 miles ahead where Alice is driving through. Users with MaaR services running on their mobile devices near the bridge can provide sensing functions, e.g., GPS, video/camera, which are searchable by Bob so that Bobs display function can call those functions in real-time through either direct P2P connections or a centralized traffic monitoring function provided by a third party. In addition to the presented video capturing usage of the application, MaaR services and applications can also maintain social diagrams for each user. For example, when Bob is driving in the area during the lunchtime, the MaaR service representor of Bob can prepare for suggestions such as good nearby restaurants with high rates by Bobs trusted friends. Other suggestions may relate to Bobs daily activities and job functions according to his current location, and provide promptly when Bob needs them. These personalized suggestions are based on correlating the location and various sensed data by the MaaR service representor.

Chapter 7

SDNIPS: ENABLING A SDN-BASED INTRUSION PREVENTION SYSTEM IN CLOUDS

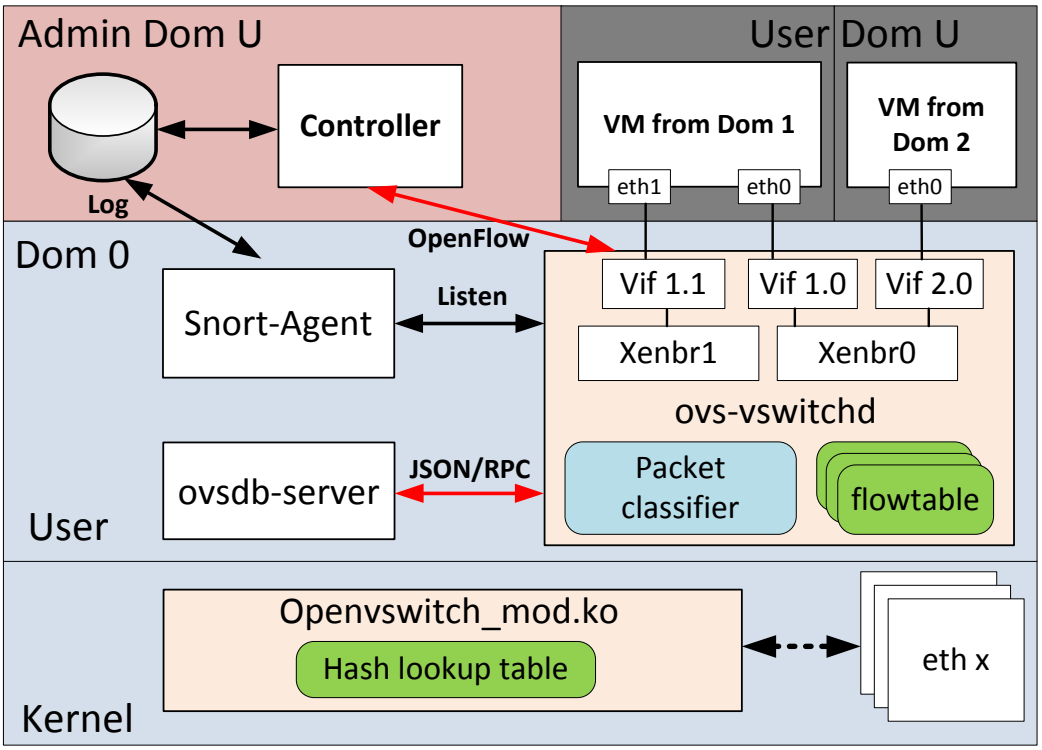


Figure 7.1: The SDNIPS System Architecture.

7.1 SDNIPS: Design and Implementation

In this section, I present the designed architecture including components and the processing flow of the SDNIPS, which is then followed by the Network Reconfiguration (NR). The architecture and components are presented in Fig. 7.1.

7.1.1 Overall Architecture and Components

Cloud Cluster is the major component hosting cloud resources and the environment where the proposed SDNIPS is applied. A cloud cluster can contains one or multiple cloud servers with major cloud-based OS installed. The established system is based on XenServer that is an efficient parallel virtualization solution on top of the bare metal. There are two types of domains in XenServer cloud OS: Dom 0 and Dom U. Dom 0 is the management domain while Dom U is the user domain. I introduce one Dom U dedicated for administrative purpose to place controller and log component, while all other Dom Us are for hosting Virtual Machines (VMs) for users. All Dom U resources are managed by Dom 0 and must go through Dom 0 to access the hardware.

Open vSwitch (OVS) is the pure software implementation of the OpenFlow switch. OVS is usually implemented in the management domain or privilege domain of the cloud system. In my designed prototype, OVS is natively implemented in the Dom 0 of XenServer cloud system. Communications among different VMs in the same physical server only need to be managed and forwarded through OVS without exposing the traffic out of the physical box. Each Dom 0 in Xenserver runs a user-space daemon (flow path) as well as a kernel space module (fast path).

In user-space, there are two modules which are *ovsdb-server* and *ovs-switchd*. The module *ovsdb-server* is the log-based database that holds switch-level configuration; while the module *ovs-switchd* is the core OVS component that supports multiple independent data-paths (bridges). As shown in Fig. 7.1, *ovs-switchd* module is able to communicate with *ovsdb-server* through management protocol, with controller through OpenFlow protocol, and with kernel module through netlink. In the kernel space, kernel module handles packet switching, lookup and forwarding, tunnel

encapsulation and decapsulation. Every Virtual Interface (VIF) on each VM has a corresponding virtual interface/port on OVS, and different virtual interface connecting to the same bridge can be regarded on the same switch. For example, VIF 1.0 (the virtual port of eth0 on VM from Dom 1) has the layer 2 connection with VIF 2.0 (the virtual port of eth0 on VM from Dom 2). OVS forwards packets based on the entries in flowtable as I mentioned in chapter 2.

Snort can be implemented in Dom 0 (privilege domain) or Dom U (unprivileged domain) based on Xen virtualization architecture. In this work, I deploy the Snort in Dom 0, which makes it natively detect the bridge in OVS and has better performance [84]. All the logging information generated from the Snort is output into a CSV file that controller can access in real time.

Controller is the component providing a centralized view and control over the cloud virtual network. The controller contains three major components, SDNIPS daemon, alert interpreter, and rules generator. SDNIPS daemon is mainly for collecting alert data generated from Snort agent in Dom 0 of controlled SDN devices, i.e., OVS. The SDNIPS daemon is implemented in the format of JSON message. The alert information is stored in JSON message and the JSON server is running at the controller side. Alert interpreter takes care of parsing the alert and targets the suspect traffic. Several information is parsed out of the raw alert data, e.g., source IP address, destination IP address, TCP port, etc. Then, the parsed and filtered information is passed to rules generator that generates the rules to be injected to the OpenFlow device to reconfigure the network.

7.1.2 Implementation

I have built up the prototype based on Xen-based cloud technology, Snort IDS and OpenFlow-based SDN. I use Snort as the intrusion detection agent since it is one

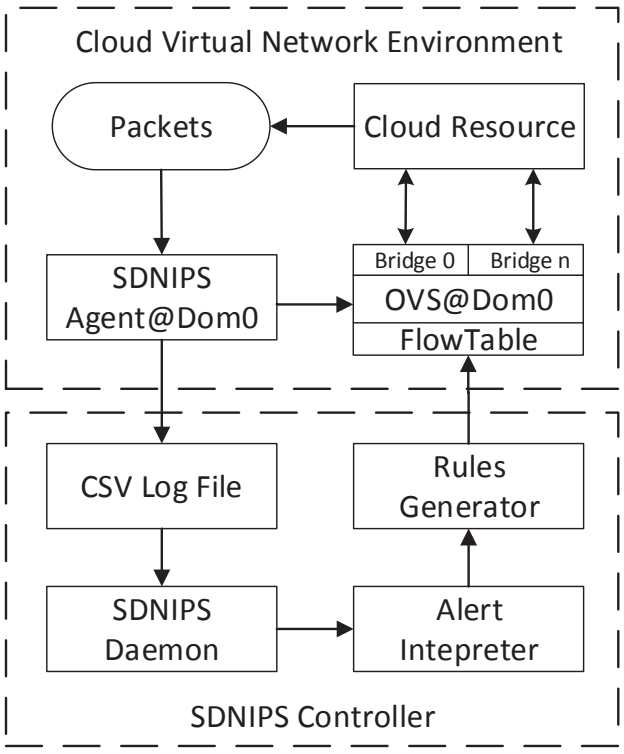


Figure 7.2: The SDNIPS Processing Flow.

of the best and dominating candidates on the market due to its high efficiency and lightweight. Snort at Dom 0 is able to natively monitor all bridges on OVS. Dom U is also possible to realize the Snort agent since the OVS can be configured as mirror based thus Snort agent can keep listening all the traffic on whatever network it can access. Based on the previous evaluation results, I install and properly configure the Snort agent in the Dom 0 in each cloud server as shown in Fig. 7.1. Controller is based on POX [38] and is implemented in a VM in Dom U. All the received Snort alerts is stored in the CSV file that controller is reading in real time. Any alert found in by the controller will be converted into flowtable entry to the OVS to take corresponding actions to prevent the suspect traffic.

7.1.3 SDNIPS Processing Flow

The processing flow of the SDNIPS is illustrated in Fig. 7.2. The network traffic is generated from the cloud resources, i.e., VMs. All network traffic must be generated from the VIFs that are attached to virtual bridges of OVS. The virtual bridge can be regarded as the virtual switch, which means all VIFs connecting to the same bridge are on the same network. Snort agent in Dom 0 has the advantage of directly detecting through the bridge, which is more efficient than sniffing the traffic by utilizing the SPAN technology. The SPAN port mirror technology duplicates all traffic on specified ports and forwards to a dedicated port that the traffic sniffer tool is listening. When any traffic matching the Snort rules is alerted into the log file, The SDNIPS daemon will store the alert information in JSON format and send over to the JSON server at controller side. After that, the alert interpreter will parse the alert information and extract all necessary information, e.g., attack type, source IP, destination IP, TCP port, etc. Finally, the rules generator will generate the OpenFlow rule entries and push them to the OVS to update the flowtable. Therefore, the following suspect traffic matching the newly updated flowtable entries will be swiftly handled with valid countermeasures in the data plane of the OVS with line rate. Currently, I have implemented the system described in Fig. 7.1.

7.2 SDNIPS vs Snort/Iptables IPS

Motivated by the limitation of the traditional IPS, e.g., Snort/Iptables IPS, in chapter 3, SDNIPS is designed to take advantages of SDN to provide more countermeasures to increase the flexibility and efficiency. This section discusses the comparison between the proposed SDNIPS and the Snort/Iptables IPS in terms of IPS working mechanisms and new capabilities.

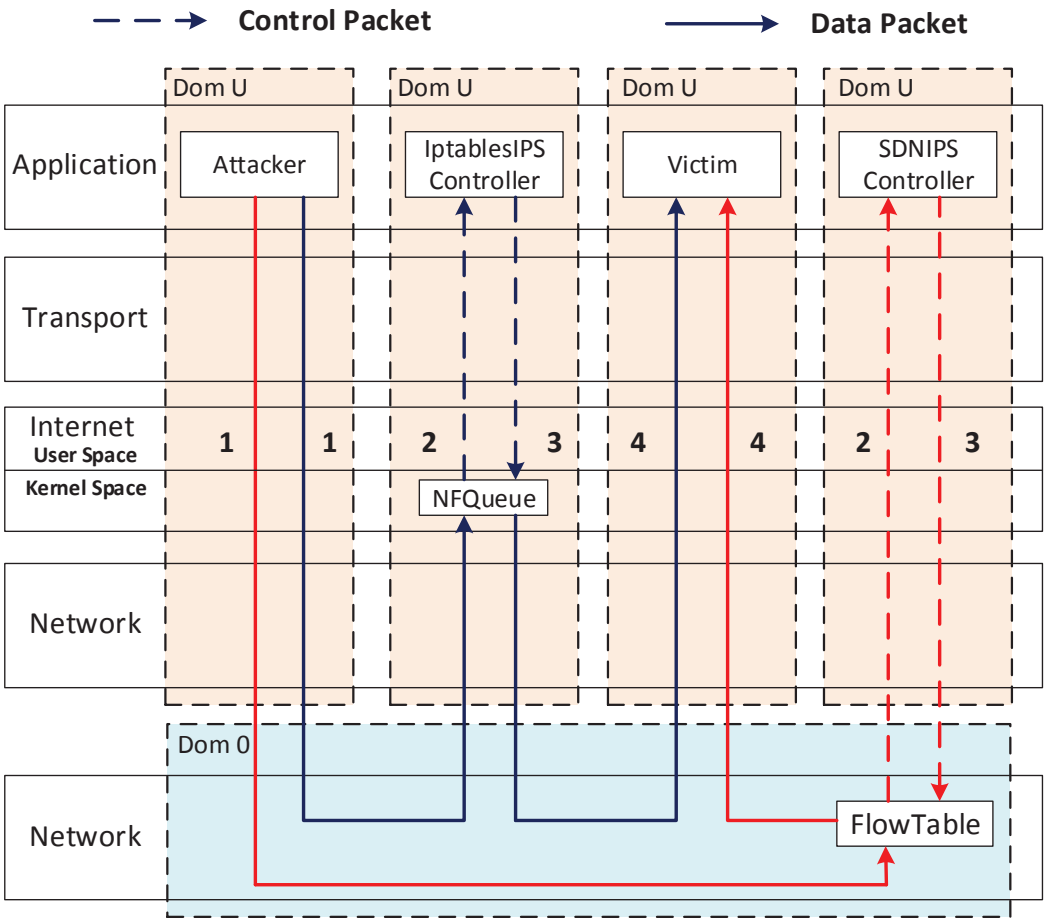


Figure 7.3: SDNIPS and Snort/Iptables IPS Mechanisms.

Traditional IPS system is not specially designed for the cloud virtual networking environment, but for a general network environment. The major difference between the general network environment and the cloud virtual networking environment is that the latter one usually has difference network domains, i.e., management network domain and user network domain. User network is on top of the management network, which means that the management network in the lower layer has better performance. Thus, I design the SDNIPS especially for the cloud virtual networking environment and take advantages of OVS in management domain in order to achieve better performance.

Two IPS solutions are different in terms of the essence, i.e., working mechanism and operation level. Fig. 7.3 indicates the scenario on how the Iptables IPS (blue lines) and SDNIPS (red lines) prevent the attacks. The number besides each line represents the sequence of the packet flow. Solids lines and dot lines represent the data traffic and control traffic respectively. For Snort/Iptables IPS, Snort needs to be configured as inline mode and recompiled with Iptables. Besides detection engines, one of most important components of the Iptables is the NFQUEUE, which is an Iptables and Ip6tables target which delegate the decision on packets to a user-space software. It then must issue a verdict on the packets. Since the Dom 0 has its own packet handling mechanism, it is inefficient and unnecessary to implement the Snort/Iptables IPS in Dom 0. Thus, placing it in Dom U should be the most practical choice.

Additionally, it is noted that OVS in Dom 0 is the same as network stack in OS kernel level, which means that OpenFlow feature is not enabled when the flowtable is empty. As shown in Fig. 7.3, when attacking packets generated from attacker's virtual interface, all the packets need to be passed through Dom 0 before being forwarded to the destination (line 1). When Snort detects any suspect traffic, it needs to inform the NFQUEUE to take the actions defined in the rules. The Iptables IPS needs to consult its brain (controller at application level), which then sends out control messages to issue command (line 2 & 3).

Finally, the suspect packet is handled at Internet level kernel space at Dom U and will be either forwarded to victim or dropped (line 4). Unlike the Snort/Iptables IPS, SDNIPS stands out since both the detection engine and the packet processing are natively deployed in Dom 0, which is dramatically efficient especially handling large amount of traffic. When packets arrive at Dom 0 (line 1), Snort detection engines is able to natively monitoring the bridges, even though the OVS controller is placed at

Dom U, only few traffic between OVS at Dom 0 and controller at Dom U is generated (line 2 & 3).

After the controller update the flowtable, all traffic with the same pattern will be processed at OVS fast path in Dom 0 (line 4). From the Fig. 7.3, it is also obvious that packets in Snort/Iptables IPS scenario need to be in and out the Dom 0 twice while the SDNIPS only needs once to fulfill the same task. Although control message in SDNIPS has further way to go than in Iptables IPS, the control message only updates the flowtable at the first time when traffic is suspected and all the traffic will be only handled by flowtable fast path at Dom 0 without interaction with controller at Dom U. Thus, due to the IPS working mechanism, SDNIPS should significantly outperformance any other Dom U IPS solution especially in cloud virtual networking environment.

Since the accuracy is one of major concerns for detection engines, taking direct actions on suspect traffic is not always an advisable choice since it may kill the healthy and innocent traffics. With the network programmability feature enabled in cloud virtual networking environment. The network for whole cloud environment can be reconfigured to provide security protections. Fig. 7.3 represents the overall working scenario of SDNIPS. The attacker generates malicious packets and sends to target through OVS virtual bridge. When SDNIPS detection agent detects the packets matching any rule and starts suspecting them, there are several possible network reconfiguration actions need to be taken to lower the risk without directly killing them through OpenFlow network programmable capability. All traffic in the cloud system will not be forwarded until OVS allows to do so. There are mainly two ways to handle the malicious traffic through network reconfiguration method, modify the flowtable and rewrite the header of the packet at controller.

Table 7.1: Network Reconfiguration Actions

No.	Countermeasure
1	Traffic Redirection
2	QoS Adjustment
3	Traffic Isolation
4	Filtering
5	Block Port
6	Quarantine

7.3 Network Reconfiguration (NR)

Network reconfiguration is an approach to reconfigure the network characteristics including topology, packet header, QoS parameters, etc. With the SDN concept enabled in the cloud virtual networking environment, network reconfiguration can be applied to construct the IPS system. Major network reconfiguration actions are summarized in Table 7.1:

1. Traffic Redirection (TR) can redirect the traffic to designed destination, such as a secure appliance (e.g. DPI unit, Honeypot) by rewriting the packet header. TR is usually implemented by using MAC/IP address rewriting. Controller can push entry to flowtable, which can take packet header rewriting action on matching packets.

2. QoS Adjustment (QA) is a very efficient way to handle flood type of attacks. OVS is able to adjust the QoS parameters of any attached VIF. After lower the TX/RX rate, suspect attack traffic will generate less impact on the network and hosts nearby. Sometimes, QA can be configured to work with other NR like traffic isolation.

3. Traffic Isolation (TI) is different from the traffic redirection in that TI provides an isolated virtual networking channel separated from others, e.g., separated virtual bridges, isolated ports or GRE tunnel. Malicious traffic will be only impact any host on its isolated virtual channel and will not impact other normal traffic.

4. Filtering is similar with the filter in Iptables, but they are different in that filtering in NR will handled packets at OVS kernel space and will not forwarded to a remote controller. MAC/IP address change is a very straightforward way to prevent the victim from being attacked by the malicious traffic. The default IPS action, i.e., drop, can be also regarded as a filtering rule that drop the matching packets.

5. Some attacks are performed by exploring a certain port, especially a public service port. By blocking those ports, the attack can be prevented as the attacking path is disconnected.

6. Quarantine is a comprehensive mean to do the isolation in cloud virtual networking environment. It works similarly with TI but it isolates the suspect network resources (not just the suspect traffic). Another difference between the normal traffic isolation and quarantine is that more flexible self-defined policies can be applied in quarantine mode. Quarantine can be also regarded as the superset of many network reconfiguration set. For example, you can quarantine suspect network targets, e.g., VMs, with only ingress permission and without egress permission. Thus, such VM can only receive traffic but cannot generate traffic to the network.

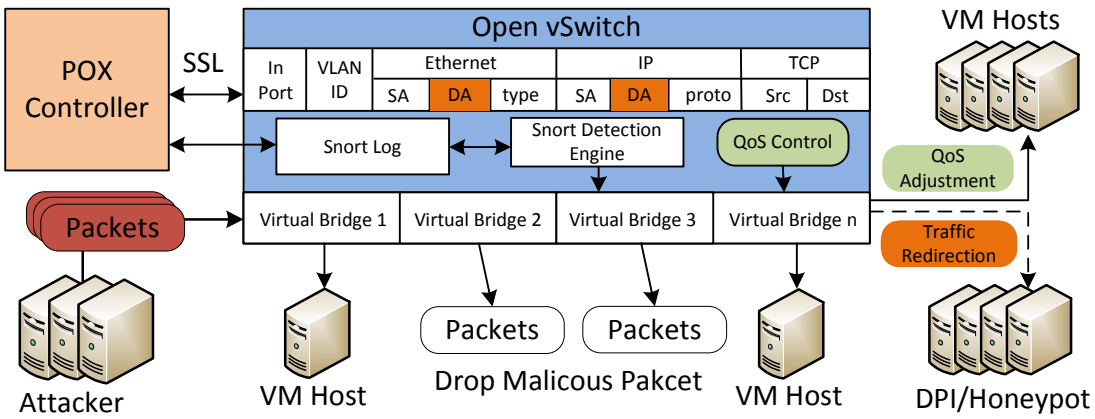


Figure 7.4: The Network Reconfiguration Mechanism.

7.3.1 Representative NR Actions

Before introducing the NR actions, the default action taken by the IPS is blocking or dropping the malicious traffic. Since NIDS may have false positive and false negative when judging the network packets, dropping packets is not always the best strategy when facing the suspect traffic. In this section, I will discuss two representative NR actions besides the default IPS action, traffic redirection (TR) and QoS Adjustment (QA). How dropping and other two NR actions work are displayed in Fig. 7.4.

Traffic Redirection

There are three ways to implement the TR: MAC Address Rewriting, IP Address Rewriting, and OVS Port Rewriting. When any suspect packet is detected, the controller firstly pushes the OpenFlow entry (i.e., matching packet header fields and corresponding actions) to OVS flowtable. When certain packets are matching specific entries, then corresponding actions will be taken for matching packets. Actions can be set as changing on any header field of the flowtable, e.g., source IP, destination IP, source MAC, destination MAC.

Traffic redirection mostly depends on the Destination Address (DA) field. When destination IP or MAC address is changed, the OVS will naturally forward the packet to the changed destination address in the packet header. This NR function is especially useful when dealing with the suspect packets, which cannot be judged as malicious one for sure and are expected not to forward to possible victim but a detection site for further checking, e.g., Deep Packet Inspection (DPI) or honeypot. As shown in Fig. 7.4, the orange colored block represents the corresponding flowtable fields that TR may change. Moreover, IP and MAC field rewriting can be combined with other NR function to implement many network function, e.g., Network Address Translation (NAT).

Beside the MAC and IP address change, there is another way to realize the TR, port rewriting. This method is also natively enabled by OVS architecture. As shown in Fig. 7.1, each bridge created in OVS can be regarded as a virtual switch. All VM VIFs are connected to virtual bridges through virtual port (i.e., virtual interface). Thus, by forwarding any packet to the virtual port, the VM VIF that connects to that virtual port will be able to receive the forwarding packets. Through this mechanism, SDNIPS is also able to set any virtual port as the output port of any packet to implement the TR function without changing the packet header.

One benefit of using port rewriting is that any packet header will not be changed while traffic redirection is being realized, which is very efficient and useful to some components (e.g., security appliance) when collecting original network data for further learning. To sum up, traffic can be redirected by OVS through multiple ways in cloud environment.

QoS Adjustment

QoS Adjustment (QA) is a desired feature when dealing with flood type of attack, e.g., DoS and DDoS. When one or multiple victims are under stress from receiving a huge amount of traffic from one or multiple sources, which cannot be confidently determined as attackers, it is always expected to slow down the current extremely fast flow and to determine if the traffic is malicious or not after further inspection. Thus, QoS adjustment is an ideal way to handle such situation.

There are two ways to implement the QA, reset the QoS parameters on either VIF or port on OVS. Setting the QoS limitation on VIF or OVS port has different applied scenarios. When setting the QoS limitation on VIF, it is necessary to firstly locate the packet source, e.g., the attacker. Thus, the number of suspect attackers would better not be large. On the other hand, when there is a DDoS attack on the network, the attack source may be a large set, which means it is infeasible and impractical to locate all zombie attackers and adjust their VIFs. To solve this issue, I introduce a smart way to implement the QoS for such situation, by limiting the incoming port on OVS. When any packet arrived at OVS, it must have an event port also called in port, as shown on the flowtable in Fig. 7.4.

The QoS of that incoming port can be limited so that any arriving packet exceeding the QoS limit will be dropped without further process, which greatly enhances the performance of QA. Also, attacker may deploy IP spoofing technology to modify the source IP address of attacking packets to avoid being traced back. Thus, the best way is to modify QoS parameters. I will not discuss the the QoS model based on the network but just explore the capability provided by SDNIPS, because QoS issue can be discussed as an individual research topic in its own area.

7.3.2 NR Selection Policy

Table 7.2: SDNIPS Actions Selection Guidance

Major Actions	DoC	Cost	Preferred Scenario
Drop	High	Low	Any determined malicious traffic
TR	Medium	Medium	Attacking traffic requiring further inspection
QA	Low	Medium	Attacks with overwhelmed traffic, e.g., DoS

In Fig. 7.2, there is a component in SDNIPS controller called rules generator. Rules generator is for choosing NR and generating the corresponding OpenFlow rules based on the detection engine output. The rules can be generated based on different algorithms that are not the focus of this work. Based on two representative NRs I mentioned above and the default drop action, I am summarizing the IPS action selection policy in Table 7.2.

Degree of Confidence (DoC) represents the degree of how confident the detection engine believes the traffic is a malicious one. Since one of the biggest challenges of NIDS is to reduce the false negative and false positive, it is impossible for a detection engine to work with a 100% correctness. Several types of evidence, e.g., detection engine signature, history behavior, machine learning result, etc., can lead to the confirmation of judging the real attack. Thus, when the traffic is suspected and the detection engine cannot draw the conclusion that it is definitely the malicious traffic, it is wise to choose the appropriate NR to keep the traffic alive and lower its possible dangerousness. Cost means the resources consumption in both the system and network level when taking corresponding actions. Different actions consumes different amount of resources due to the frequency of OpenFlow operation, e.g., updating

flowtable, taking OpenFlow actions. In this way, it is possible to enumerate preferred scenarios of attacks that can be counted by using certain NR setup.

Drop is the default NR and it has the lowest cost since when the packet match the entry in flowtable, it will be simply dropped without any further actions. This action is the best candidate to prevent the determined attacks with high DoC by terminating the malicious packets.

Traffic Redirection (TR) is appropriate for the traffic with medium DoC. When SDNIPS detects possible attacking traffic with a medium DoC, the traffic can be redirected to a secure appliance, e.g., DPI proxy or Honeypot for further inspection and learning. After the secure appliance further inspect the traffic, the traffic can be possibly forwarded to the original destination or take other actions, e.g., drop. Through the TR, the suspect traffic will not be forwarded to the original destination until a further process is done. Since TR needs to use packet header or OVS port rewriting technology for every single suspect packet matching the flowtable entry, it costs more resource than simply drop action.

QoS Adjustment has two ways as I mentioned above, VIF based QA and virtual port based QA. I am mainly focusing on the virtual port-based QA since it can be applied to broader range of scenarios. QA is preferred for traffic with low DoC than TR since the malicious packets with lower DoC can be sent to the original destination. Packets with low DoC cannot be determined as the malicious one. Thus, such traffic does not need to be dropped or redirected. QA costs similar amount of resource with TR since all the packets need to be handled by OVS. QA is preferred for flood type of attack with either few or large set of sources.

7.4 Evaluation

7.4.1 Evaluation Environment

I establish the SDNIPS prototype by using one cloud server with OVS installed and properly configured in Dom 0 which has 4 virtual CPUs. The detection engine in Dom 0 can directly access the virtual bridges in OVS to monitor all tenant networks while Snort/Iptables-based IPS agent in Dom U can only monitor the tenant network where it is in. The OVS controller is implemented based on the POX controller [38]. All other components are placed in VMs.

Traditional Snort/Iptables IPS is implemented in a VM (Ubuntu 12.04 Server edition, 4 virtual CPUs and 2048 MB Memory) at Dom U. All VMs at Dom U are configured with VIF with 10GbE maximum capacity whose actual bandwidth is around 8 Gbits/s based on our testing in the real XenServer virtual network environment. I evaluate three major performance scenarios:

1. Performance comparison between placing detection engine in management domain and user domain;

2. Performance comparison between our proposed SDNIPS and traditional Snort/Iptables IPS in terms of the attack prevention capability (I will simply use the term IPS in the following to represent the Snort/Iptables one); and

3. Performance of the designed and developed NR functions to validate their feasibility and efficiency.

In the evaluation, the traffic are generated by hacking tools and packet generators such as [1, 4, 51, 2, 59] to mimic the real attack scenario in the cloud virtual networking environment.

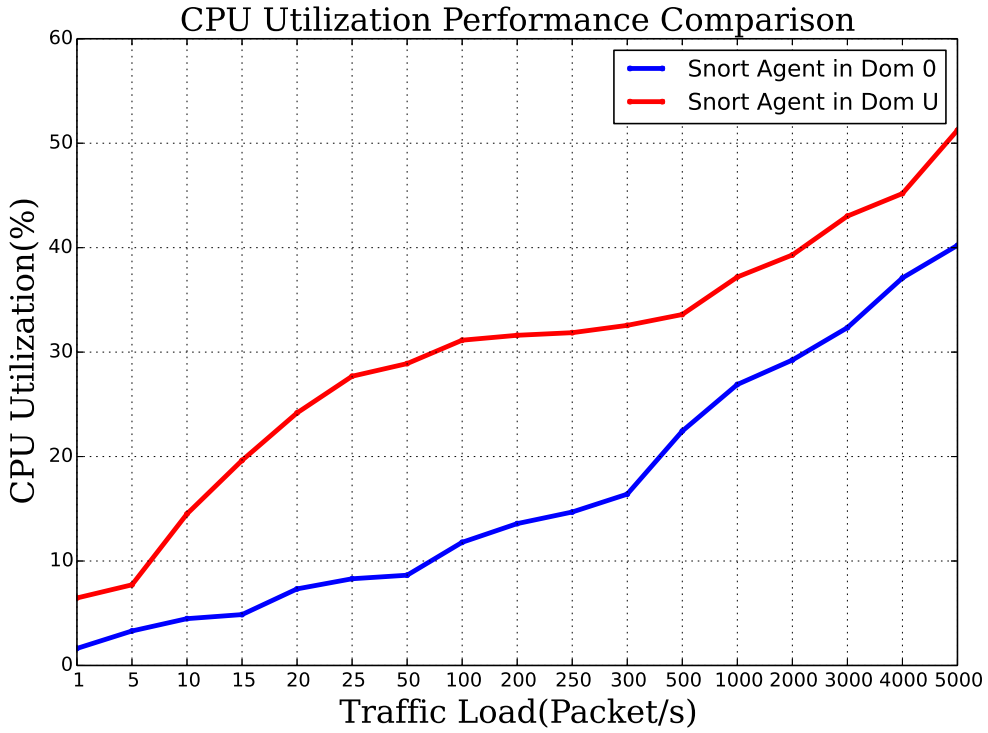


Figure 7.5: CPU Utilization Performance Comparisons.

NIDS Performance

Fig. 7.5 illustrates the performance comparison between Snort in Dom 0 and in Dom U, measured by CPU Utilization. The traffic load is represented by the packets sending speed from 1 to 5000 packets per second (pps). Both Dom 0 and Dom U are configured with 4 virtual CPUs. The detection engine running in Dom U also consumes computation resources in Dom 0 since Dom 0 is the management domain. Thus, the CPU utilization is measured by the summation of CPU utilization of Dom 0 and Dom U. In sum, the performance of running the detection engine at Dom 0 consumes less CPU resources than in Dom U because Snort is able to natively access all the network traffic through OVS bridge in Dom 0. Thus, based on the

evaluation in the Xen-based cloud environment, the conclusion can be drew that deploying the Snort detection agent in Dom 0 has better performance than in Dom U, which validates the SDNIPS design in Fig. 7.1. The main reason is that all the traffics of Dom U need to go through OVS in Dom 0, and virtual resources in each Dom U need to be routed to access the hardware resources through Dom 0. Thus directly deploying the Snort agent in Dom 0 is more efficient in terms of both host and network resource utilization.

OpenFlow-based IPS vs Iptables-based IPS

In this part, I compare the performance of proposed SDNIPS and one traditional IPS candidate, i.e., Snort/Iptables IPS (mentioned as IPS as following). SDNIPS only deploys the default NR action (drop) to make the comparison fair, since Snort/Iptables IPS does not have extra NR capability besides drop.

Fig. 7.6 shows the health traffic forwarding capability under overwhelmed workload for both systems. To evaluate the performance, I set the IPS and SDNIPS as the proxy of two virtual end hosts to forward ordinary traffic packets. I use hacking tools [1] to initiate the DoS attack toward the IPS target at different attacking rate as the interference source. For demonstration purpose, I choose two major DoS attacks as candidates, which are Ping of Death (PoD) and SYN flood Attack. And then, one VM sends normal health packets to another VM via proxy at the rate of average daily traffic rate, to measure the IPS health traffic forwarding capability under attack. In traditional IPS solution, DoS packets are captured by the IPS detection engine, which further matches the rules and takes drop action on the inspecting packets.

In SDNIPS solution, the OVS fulfills the same task as Iptables does but it handles packets by a different and more efficient mechanism. After Snort detecting packets matching any signature, controller is able to be aware of the current threats in real

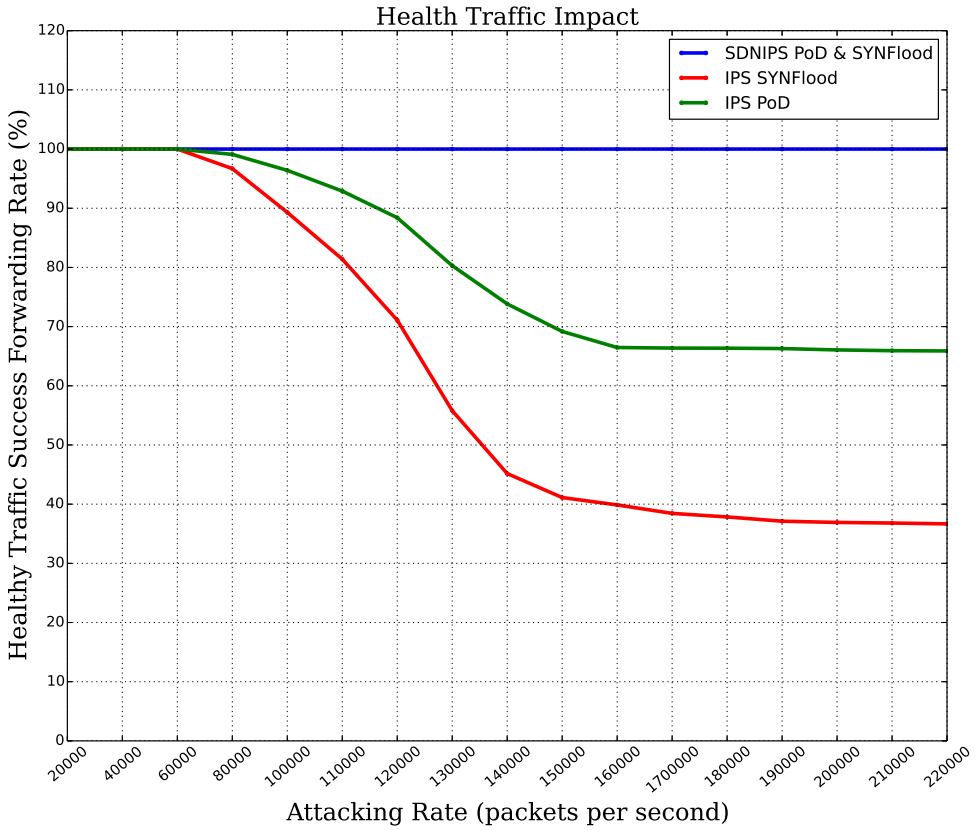


Figure 7.6: The Impact of Health Traffic.

time by parsing CSV log file and then pushes corresponding flow entries into the flowtable. After flowtable is updated, the malicious traffic can be handled by using OVS fast path approach in line-rate, which can dramatically increase the system performance and avoid the overwhelming traffic to interrupt normal traffic forwarding.

In Fig. 7.6, SDNIPS under both type of attacks has almost 100% forwarding rate, which means that all normal traffic can be properly forwarded even the SDNIPS is under the significant stress. For Snort/Iptables IPS, it has decreased around 68% and 38% success forwarding rate under PoD attack and SYN flood respectively after the attacking increase to 150,000 packets per second. The reason why health traffic has higher success forwarding rate of health traffic under IPS PoD attack is because

the SYNFlood attack consumes more bandwidth and network resources compared to IPS PoD attack.

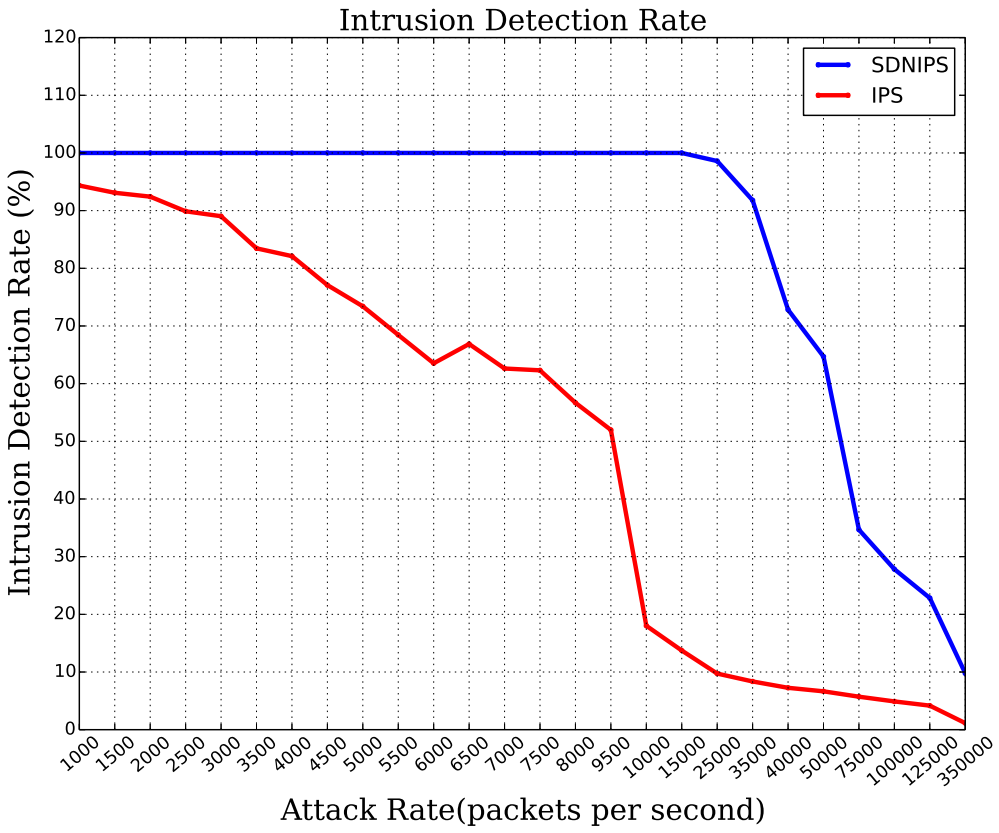


Figure 7.7: Evaluation of Intrusion Detection Rate.

In Fig. 7.7, I evaluate the alert generation capacity of both IPS and SDNIPS under flood interference. This metrics also states how IPS can process the attacking packets from security perspective. To evaluate this performance, I generate two different types of attacks, which are DoS flooding attack acting as the stressful background traffic and ICMP flood attack acting as an potential threat to be tested. This evaluation mainly indicates whether IPS and SDNIPS can generate alert under high workload stress. The figure shows the successful alert generation rate of ICMP attack under DoS

attack interference. It suggests that alert generation of traditional IPS is impacted by DoS interference and most resources of IPS system are used to handle DoS attack therefore the performance of alert generation rate decreases as the ICMP attack speed increases.

When the speed of the ICMP attack reaches to 15000 packets per second, IPS can only generate 13.72% alerts of the ICMP attack. On the other hand, SDNIPS is able to efficiently avoid interference from DoS flooding attack due to OVS capability, so it can successfully alert all the threats that are sent at the speed of 15000 packets per second. When the speed of the ICMP attack reaches to 30000 packets per second, the performance of SDNIPS start decreasing, and when the speed of ICMP attack increases to 300000 packets per second, Snort agent in SDNIPS is not able to capture packets and launch alerts because the snort detection engine itself almost reached its threshold.

Thus, the evaluation comparing between proposed SDNIPS and traditional IPS validates the analysis mentioned in section 7.2, which is that the SDNIPS has better network and security performance in cloud virtual networking environment.

Network Reconfiguration

After the SDNIPS and traditional IPS are comparatively evaluated, I evaluate the performance of SDNIPS NR alone since traditional IPS does not have the NR capability. I mainly evaluate two NR actions mentioned above, Traffic Redirection (TR) and QoS Adjustment (QA), in the aforementioned cloud environment.

Fig. 7.8 shows the performance of resources consumption in term of the CPU utilization of Dom 0 for all discussed NRs. I am using packet generator [51] to generate the packets captured by Snort and processed by flowtable in order to test the resources utilization change in the cloud system. In each NR approach, TR is

implemented by using destination IP & MAC rewriting; while TR with spoofing reply is implemented by rewriting not only destination IP & MAC address but also source IP & MAC of victims. Thus, the attacking traffic can be redirected to a security appliance that is able to spoof the attacker by replying the packet with victims' IP & MAC address as source address. TR with spoofing feature consumes a little more resources than the pure TR since OVS modifies more packet fields to enable the spoofing feature.

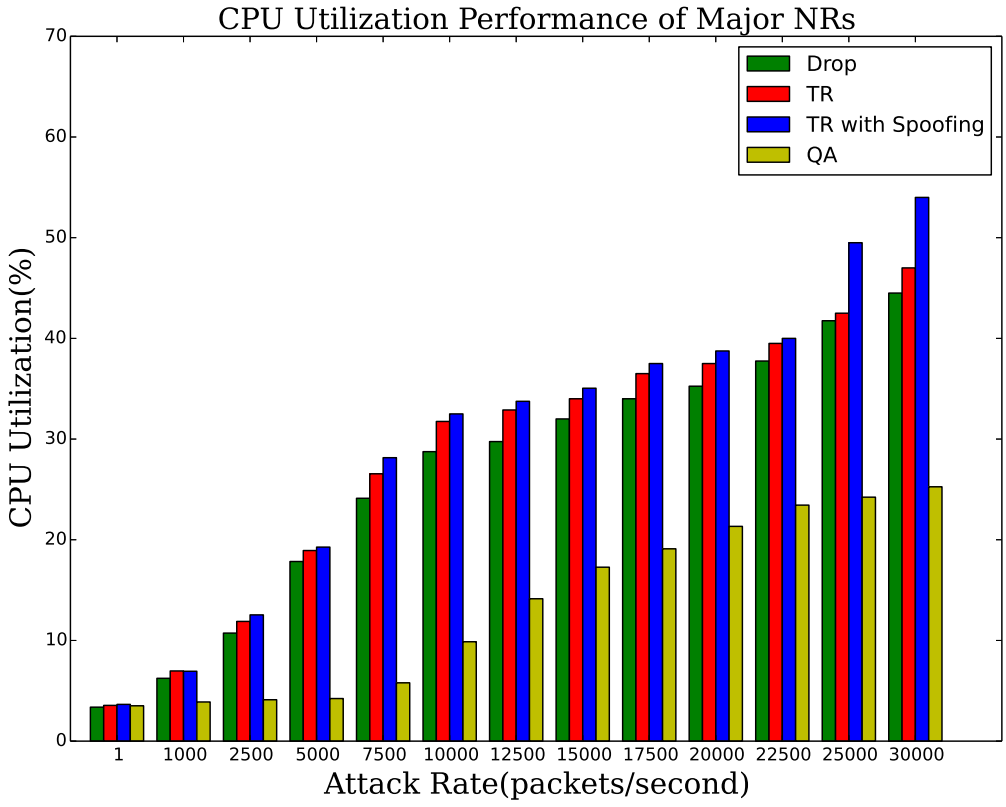


Figure 7.8: CPU Utilization Performance of Major NRs.

The default NR, i.e., drop packets, consumes less system resource because the OVS does not modify the matching flow and just simply drop them (output to a non-existing virtual port in POX controller implementation). In the QA scenario, it

has the best performance among all NRs because the rate limiting action is performed based on OVS native mechanism, which means excess packets will be discarded and OVS does not have to inspect and match the packet with all kinds of fields.

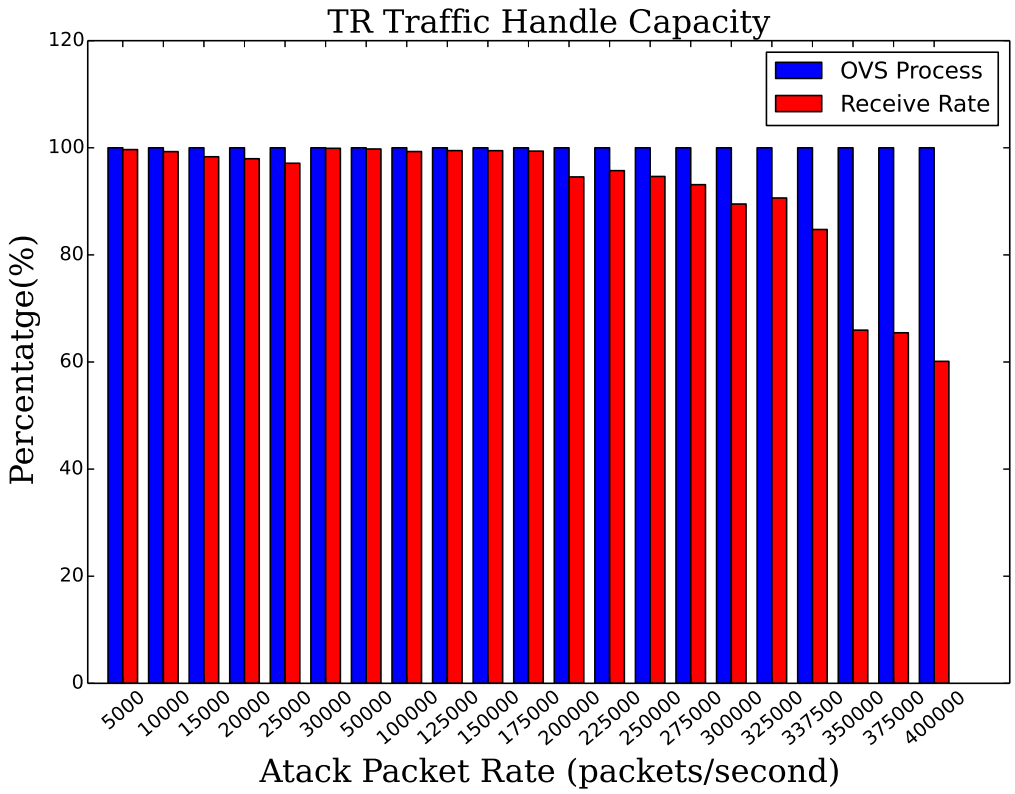


Figure 7.9: TR Traffic Handle Capacity.

Fig. 7.9 provides the performance on the capacity of OVS and secure appliance VM, e.g., DPI checker. In this scenario, I use TR (port rewriting) to redirect all the attacking traffic to a DPI checker VM (4 vCPUs, 2,048 MB memory, Ubuntu 12.04 server OS). When the attacking packet rate is increased from 1,000 packets per second up to 400,000 packets per second, the OVS can handle all the traffic without any packet loss, which validates that the packet process capacity of OVS is expected. The receive packet means the percentile of OVS redirected traffic has been received by the DPI checker's VIF. When the packet rate reaches the 400,000 packets per

second (I assume that there is no more than 400,000 packets generated in a single physical server per second), the OVS can still handle 100% of the packet and security appliance can receive 60% of them due to its VIF capacity.

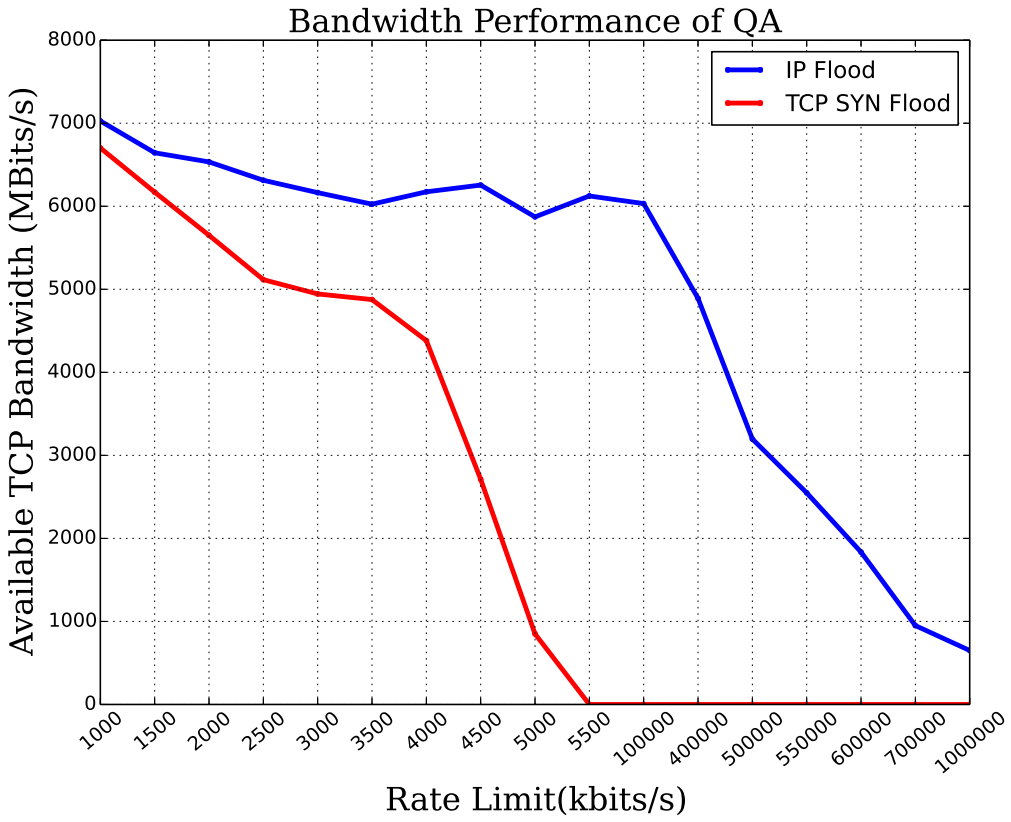


Figure 7.10: Bandwidth Performance of QA.

Fig. 7.10 evaluates the TCP bandwidth performance of OVS-based QA under different speed limit rate. I use either IP flood or TCP SYN flood as the background attack traffic and try to evaluate the available bandwidth on the same network. Both IP and TCP SYN flood attack send flood traffic at their maximum rate (about 300k packets per second). I limit the source port on OVS thus the attacking traffic arriving at OVS is limited before being processed. The default speed is 10Gbits/s without

any rate limit in XenServer cloud environment. The range of the speed limit is from 1 Mbits/s to 10Gbits/s.

For IP flooding attack, when the attack packet rate raises up to 5 Mbits/s, the available bandwidth is still almost full (around 8 Gbits/s in XenServer environment), which means when I limit the attacking traffic below 5 Mbits/s, it does not impact the network where the victim is. With the increase of the speed limit, more bandwidth is occupied by the attack traffic and thus less bandwidth is available for the victim. For the TCP SYN flood attack, when the speed limit reaches to 4 Mbits/s, the TCP bandwidth starts degrading because TCP SYN flood needs to establish the TCP connection and this will definitely impact the TCP bandwidth performance. When the speed limit is above 5.5 Mbits/s, almost no available bandwidth is available for the victim. This evaluation result provides a valuable reference for any system when deploying the QA NR as a security solution.

7.5 Network Security as a Service (NSaaS)

7.5.1 NSaaS System Architecture

After the SDNIPS is proposed, implemented, and evaluated, next research focus goes to how to convert this system into a service provisioning system so that each cloud tenant can claim such security service to use. Fig. 7.11 illustrates the system architecture of the proposed Network-Security-as-a-Service (NSaaS). It shows its framework in a virtual networking environment including cloud tenants running within the data centers programmable networks and edge cloud (i.e., customer resources such as computers or mobile devices). This setup is to demonstrate a common programmable networking environment where the IT infrastructure can be established based on public and private clouds running within one or multiple data centers and edge clouds

containing end users computer/devices. For this NSaaS, I focus on network security and various network security appliances that can be componentized as security appliances to provide Network-Security-as-a-Service (NSaaS). NSaaS is controlled by the cloud security service model including several major components to deploy NSaaS. In the following research description, I will focus the presentation on how to construct NSaaS.

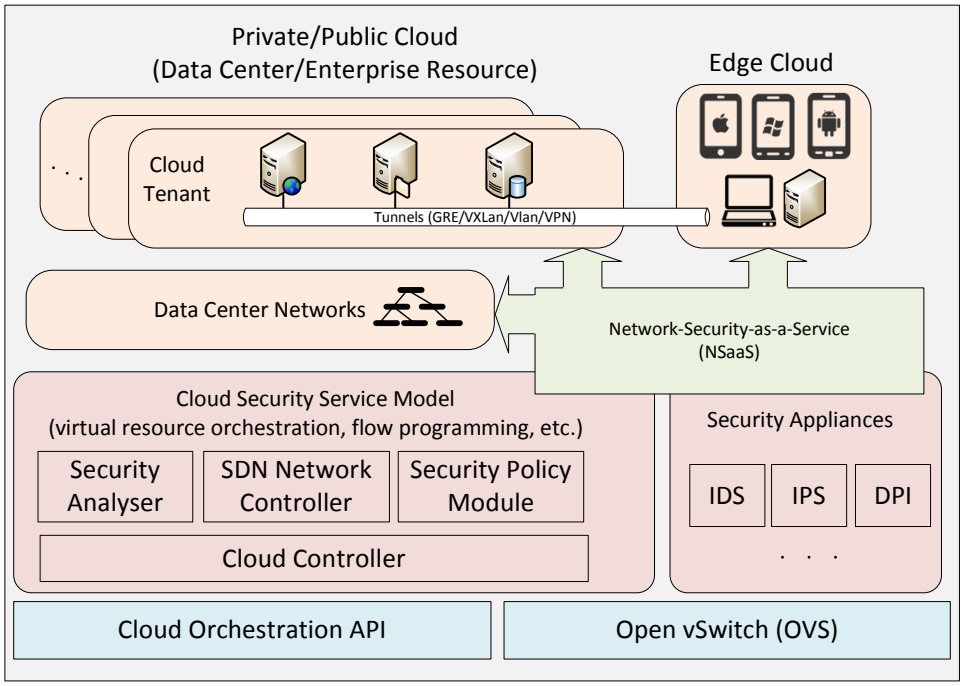


Figure 7.11: SDNIPS as a Service System Architecture

Network-Security-as-a-Service (NSaaS) is a security appliance allocation model according to the security requirements in the cloud environment. Similar work has been presented in the Internet environment [35], where the authors proposed a mechanism called FireCol that is a collaborative system detecting flooding DDoS attacks on the Internet Service Provider (ISP) network. FireCol takes the hop-count as the evaluation metric and deploys IPS in the ISP networks. In contrast, NSaaS is an SDN-enabled data center environment manipulating both traditional cloud resources and the mobile edge cloud, and thus the IPS resource management must take a com-

prehensive approach that can be easily incorporated into the cloud cost evaluation framework. In general, the NSaaS is deployed based on the cloud system resource management according to the utilization of vCPU, memory, virtual hard disk, network bandwidth, etc.

As shown in Fig. 7.11, the NSaaS are able to access and control all of three domains, which are datacenter programmable networks, private/public cloud resources, and edge cloud. Edge cloud is consist of mobile devices that may be acting as either service resource consumer and provider. It connects to the private/public cloud domain through layer-2 network virtualization technology, e.g., GRE tunnel, VXLAN, VPN, etc. Both edge and central cloud are connected by and sitting on top of the SDN programmable networks. SDN programmable networks can be in the form of either physical OpenFlow swtich or virtual Open vSwitch, and can provide flexible and scalable NFV. The NSaaS is also a hierarchical security provisioning service, which means that IDPS in terms of virtual appliances can be deployed in different tenant.

7.5.2 NSaaS Elasticity Model

Another critical research issue of NSaaS is that how to efficiently deploy the NSaaS service in terms of IDPS virtual appliance in the datacenter environment. In the FireCol [35], the IPS is deployed right on the routers of the ISP network and the major metric is based on the detection range (number of hop) of IPS. In contract, the NSaaS is deployed based on a more elastic policy where system resource (vCPU, memory, virtual harddisk, etc.) and network resource (amount of traffic) are all considered.

In the data center environment, the resource is virtually isolated in forms of tenants. Resources in each tenant may cross multiple cloud clusters (each is denoted by C_l) and may include edge cloud devices. NSaaS allocates security appliances for each

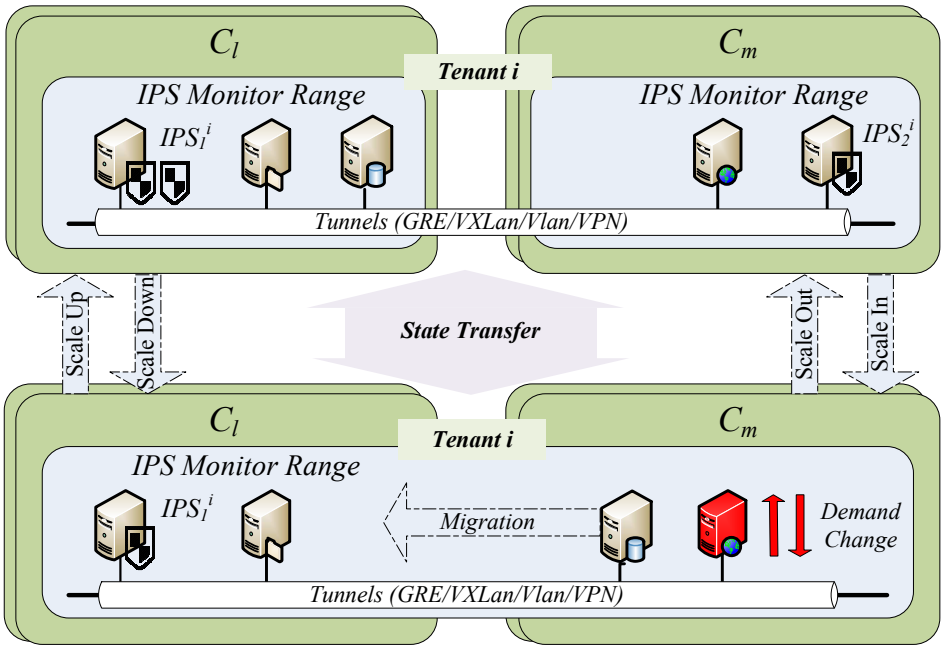


Figure 7.12: NSaaS Elastic Service Model for One Tenant.

tenant or for the entire data center virtual network. To simplify the demonstration of the elastic NSaaS elastic service model, I choose the IPS appliance as the example and define the following notations that will be used in Fig. 7.12: the j^{th} IPS in tenant i is represented as IPS_j^i, where each IPS has its own monitor and control range that does not overlap with others. In this illustrative example, I define four elastic strategies for IPS service deployments that are summarized in Table 7.3. Scale Up/Down is to change the cloud resource configuration, and scale in/out is to change the number of resources in terms of virtual machines.

The IPS with both VM configuration and network configuration are considered. The VM configuration includes the number of vCPUs, virtual memory and the virtual hard disk, while the network configuration includes different QoS requirements. For example, an IPS hosting a VM can have x vCPU, where $x = 1, 2, ...n$. The number of allocated vCPU is decided according to the security performance need. All configurations are pre-defined by the system administrator in to a pool. For instance of

Table 7.3: NSaaS Elastic Operations.

Title	Actions
Scale-Up	Increase the resources config
Scale-Down	Decrease the resource config
Scale-Out	Create a new IPS virtual appliance
Scale-In	Shutdown a existing IPS virtual appliance
Migration	Migrate tenant resource to optimized cluster

vCPU, the pool is pre-defined as follows: {1, 2, 3}, which means the number vCPU can be configured as 1, 2, or 3. In the example setting, 1 denotes the bottom flavor of vCPU and 3 denotes the top flavor. Scale up means increase flavor and scale down means decrease the flavor.

There are two thresholds can be established, say *overutilized* and *underutilized*, to decide adding or removing vCPUs. In general, the *overutilized* status triggers the scale-up and the *underutilized* status triggers the scale-down action. Moreover, there are two special cases: when an IPS hosting a VM only has one vCPU (bottom flavor) and it is in the *underutilized* status, the system can enforce the scale-in policy; and when an IPS hosting VM has n vCPU (which is the top flavor) and it is in the *overutilized* status, it can scale-out. Particularly, the scale-in action has four sub-actions: (1) evaluate the expected networking resources needed by the tenant monitored by the current IPS; (2) check if any other IPS is available in the same tenant with sufficient resources; (3) migrate the VMs from the current tenant into the monitor range of available IPS from step 2; (4) shutdown the current IPS. Scale-out is the reverse action of scale-in but with simpler sub-actions: (1) evaluate the expected resources needed by the tenant under the current IPS; (2) create a new IPS with proper number of vCPUs; (3) migrate the VMs from the current tenant to the

monitor range of a newly created IPS. The example of the scaling function can be illustrated in Fig. 7.13.

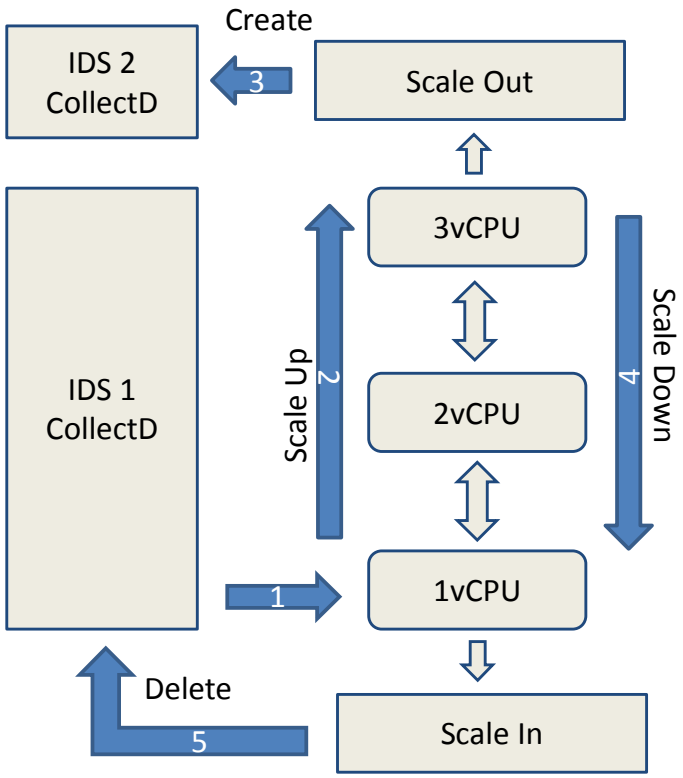


Figure 7.13: NsaaS Elasticity Example

Chapter 8

CLOUDARMOUR: CONSTRUCTING A SDN-BASED DEFENSIVE SYSTEM IN THE VIRTUAL NETWORKING ENVIRONMENT

Motivated by the aforementioned research challenges, I am aiming to propose a detection&prevention system with complete lifecycle to defend the harmful and complicated attacks in the distributed cloud environment. The complete lifecycle can be divided into three stages as shown in Fig. 8.1: *Traffic Sniff*, *Attack Analysis* and *Actions Execution*. In the traffic sniff stage, attacker are detected by using IDS or other monitoring agents. In the analysis stage, it should analyze the attack behavior that helps further determine the malicious traffic. Finally, prevention actions should be optimally executed to kill the malicious traffic without impacting other normal traffic.

In this research, I propose "CloudArmour", a new SDN-based cloud network security framework, providing attack traffic monitoring and detection, attack analysis, and mitigation deployment. Compared to existing solutions, CloudArmour provides the following unique features:

- I design a Traffic Statistics Aggregate Module (TSAM) to monitor network traffic patterns and detect anomaly behaviors. Moreover, TSAM can also in-

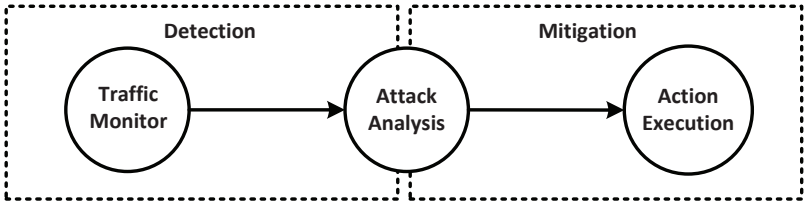


Figure 8.1: CloudArmour Lifecycle

spect real packets to perform signature-based attacks detection. TSAM's implementation is efficient in that it is implemented based on OpenFlow protocol; moreover, TSAM's implementation is based on component-based design that provides common interfaces (e.g., data access and operational functions) to facilitate other CloudArmour modules. For examples, TSAM provides interfaces to support various existing detection modules such as anomaly and signature based detection modules.

- I develop an Attack Source Tracer Module (ASTM) to track attack sources at the SDN physical layer that prevents attackers from spoofing attack source addresses even using dynamic address assigning approaches, e.g., no fixed IP is assigned to a host such as using DHCP-based IP address assignments. ASTM provides strong security protection against stealthy attack sources. This is because attackers need to compromise the hypervisor of the cloud system in order to fake their addresses.

- I devise a Mitigation Executor Module (MEM) to systematically implement various network reconfiguration approaches such as packets filtering, traffic redirection, traffic shaping (i.e., QoS-based adjustment), etc. I provide a comprehensive performance evaluation to address the pros and cons of various mitigation approaches.

- I implement 20 Python-based APIs for OpenFlow Controllers to interface traffic monitoring, attack detection, traffic control and analysis modules. I illustrate how to use the presented CloudArmour APIs by providing a comprehensive tutorial on constructing a DDoS defense system in a cloud environment.

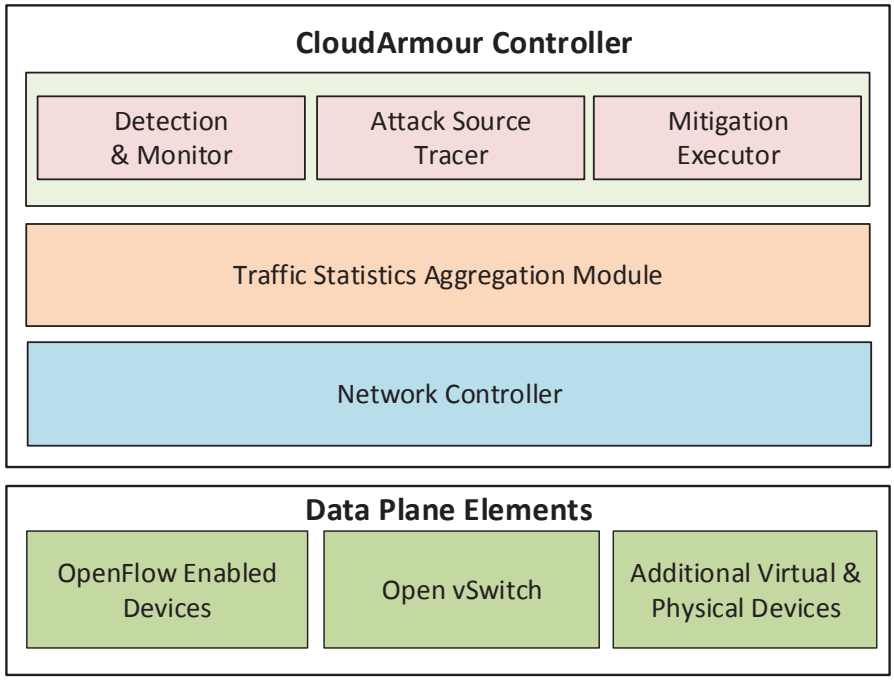


Figure 8.2: CloudArmour System Overall Architecture.

8.1 CloudArmour Overall Design

The proposed CloudArmour system architecture is described in Fig. 8.2. All SDN-enabled data plane elements including OpenFlow switch, Open vSwitch, and other additional virutal&physical devices are connected and controlled by a centralized CloudArmour controller that is implemented based on POX controller [57]. Once the centralized controller is connected with data plane elements, it has a cloud system view including network topology, network connectivity status, packets statistics information, etc. The proposed Traffic Statistics Aggregation Module (TSAM) is designed on top of the SDN controller. It provides several traffic related interfaces (e.g., data access and operational functions). For example, TSAM APIs can be called to expand flow table rules to collect more detailed traffic information such as IP/Mac address, port number, etc.

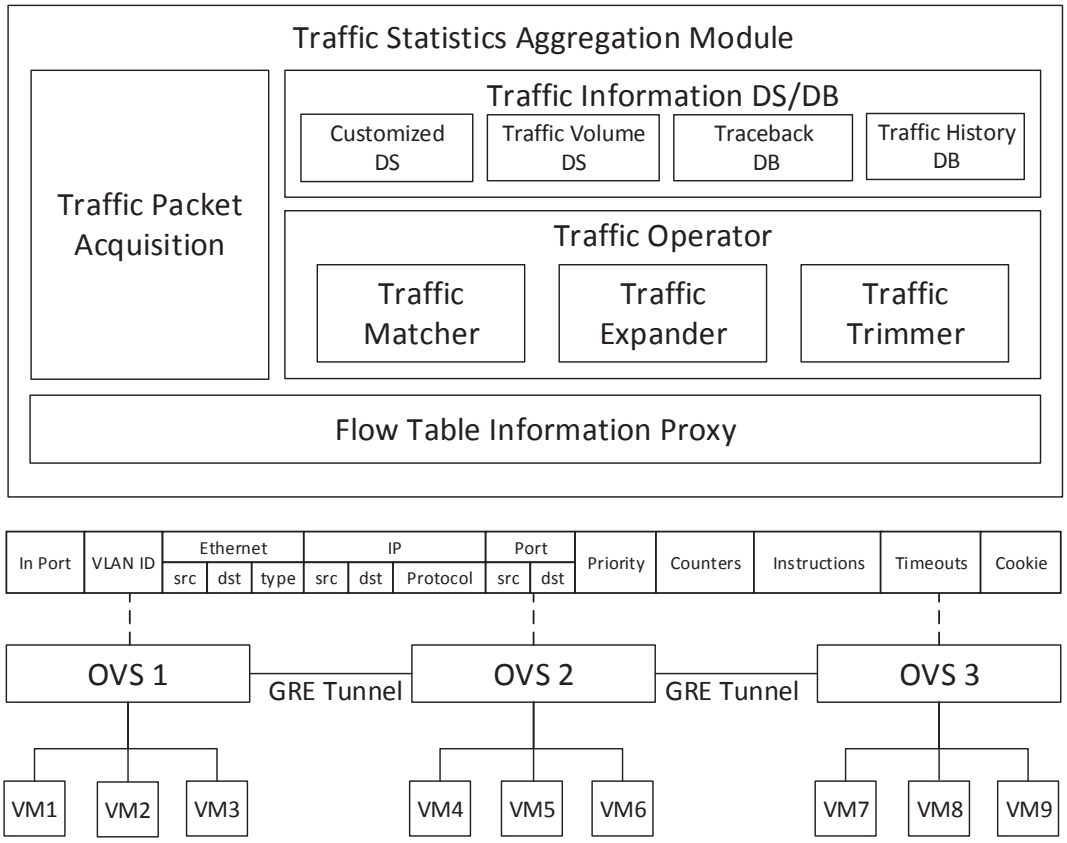


Figure 8.3: Traffic Statistic Aggregation Module

Above the TSAM, there are three highly integrated application modules: Detection&Monitor Module, Attack Source Tracer Module , and Mitigation Executor Module. The Detection&Monitor Module is able to read the traffic statistics information or fetch the real packets from all connected OpenFlow devices. It can also set up the connection between OpenFlow devices and existing security application such as intrusion detection engine. Furthermore, the Attack Source Tracer Module (ASTM) traces back from the victim to identify the location of attack sources. Finally, the Mitigation Executor Module (MEM) performs various network reconfiguration approaches such as traffic redirection, drop, QoS adjustment, etc.

8.1.1 *Security Application Interface Architecture*

There are total 20 module APIs designed in the CloudArmour system. The CloudArmour system provides API functions at different layers. The TSAM module provides several API function calls to retrieve the traffic statistics information in different format. On top of the TSAM, several highly integrated functions are constructed to provide integrated services including attack source trace, mitigation execution, etc. Users have sufficient flexibility to construct their security solution by organizing several integrated modules together or develop their own service module such as the detection engine based on our lower level module, e.g., traffic packet acquisition module that will be discussed in section 8.2. Section 8.4 will give a concrete example on how to construct a full life cycle security solution to defend the DDoS attack based on the CloudArmour. All APIs are listed in section 8.5.

8.2 Traffic Statistics Aggregation Module (TSAM)

TSAM is designed to collect traffic statistics information and provide flexible data access and operational functions. It is able to retrieve traffic statistics information and packet content in a more efficient and flexible fashion because OpenFlow can records all traffic flow information and corresponding statistics information in the flow table of all its controlled devices.

8.2.1 *Traffic Statistics Aggregation Architecture*

As part of the infrastructure of the CloudArmour, the TSAM shown in Fig. 8.3 can be divided into several components, which are Flow Table Information Proxy, Traffic Operator, Traffic Information Databases (DB)/Dataset(DS), Traffic Packet Acquisition.

The Flow Table Information Proxy retrieves flow table information periodically and it can also configure flow table entries onto OpenFlow switches. The Traffic Operator includes three modules, which are traffic matcher, traffic expander, and traffic trimmer. These traffic operator modules get inputs from flow table information proxy and output to several databases(DB) and datasets(DS). Data access and operational function APIs supports flexible traffic statistics information retrieval. Existing OpenFlow implementation does not support packet content inspection. To address this issue, Traffic Packet Acquisition extends the limited flow level information to packet content level that can be used for deep packet investigation. The developed TSAM APIs can be found in Table. 8.1 in section 8.5.

Normally, network related security applications or modules need to retrieve the traffic data information for further analysis. There are many ways to retrieve the traffic data, for example, configuring the SPAN to forward the traffic to certain appliances, retrieving the traffic data based on some monitoring protocols, e.g., sFlow, NetFlow, etc. There are pros and cons for all aforementioned solutions. Therefore, by utilizing this characteristic, Traffic Statistics Aggregation Module (TSAM) is able to retrieve traffic statistics information and then further provide the traffic data and packet acquisition services to the upper layer security applications, which is shown in Fig. 8.3. As part of the infrastructure of the CloudArmour system,

8.2.2 Flow Table Information Proxy

In order to collect the flow statistical information, the flow table information proxy module maintains the connections with all OpenFlow devices. It periodically checks the OpenFlow connections and grabs the traffic statistics information from all connected OpenFlow switches. All the information collected by flow table information proxy will be processed by the traffic operator. It also performs control function by

injecting flow table rules into any specified OpenFlow switch. Basically, it acts as a proxy between OpenFlow devices and CloudArmour modules by implementing read and write functions.

8.2.3 Traffic Operator

To provide customized flow statistic data for CloudArmour APIs that can be called by seccurity analysis modules, e.g., anomaly intrustion detection module, the traffic operator is designed including three modules: traffic matcher, traffic trimmer, and traffic expander. Their traffic operational functions are described as below:

Traffic Matcher is a module that implements matching function. Due to the native feature of OpenFlow protocol, packets or flows can be matched through any single field or combinations. Many APIs are implemented by utilizing the matcher function.

Traffic Trimmer is the module trimming raw statistics data into a desired or customized format. This module works with other modules such as traffic matcher and traffic expander to generate the following statistics DB/DS, i.e., *Traffic Volume DS*, *Traceback DB*, *History Traffic DB*. Users can also use these modules to build their own customized data set, i.e., *Customized DS*, to fulfill their own design purposes.

Traffic Expander enables users to deeply investigate flow detail information. It is possible that the flow table forwards packets based on very simple flow information, which means that the flow statistics information may include insufficient information. To investigate the traffic behaviors, more detailed traffic information are expected. At this point, I implement the traffic expander function to expand the simple flow entry into more detailed ones.

By default, the OpenFlow switch in datacenter can forward packets based on layer-2 or layer-3 headers. Thus, the statistics information only maintains layer 2 or

3 header information rather than upper layer information that is important metric for determining attack signatures and behaviors. Once the traffic expander function is triggered, the matched flow entry will be expanded with more detailed header information specified by users. After detailed flow entries are injected into flow tables, the traffic information proxy is able to collect more detailed statistics information of the targeted traffic.

An example is introduced in the Fig. 8.4. The packets coming from node A are forwarded to node B only based on IP addresses and corresponding flow entry is in the format of {A-B, forward}. However, this information is insufficient for identifying the malicious traffic type, e.g. HTTP traffic (TCP, 80). The traffic expander module is able to dynamically update detailed rules by learning the expected header information from packets. The developers can use traffic expander modules API to specify the flow table fields that need to be expanded, such as IP protocol. So the corresponding expanding rules, {A-B, TCP, forward} and {A-B, UDP, forward}, will be injected to flow table. When more detail information is expected, such as destination port, flow table entry {A-B, UDP, 53, forward}, {A-B, UDP, 2049, forward},{A-B, TCP, 22, forward} and {A-B, TCP, 80, forward} are inserted to the flow table. The port numbers in the updating flow entries are automatically learned from the corresponding headers of packets. We can increase the priority of the flow entry each time it is expanded. Thus, due to the different priority, the TCP traffic with port 80 from A to B only matches the highest priority rule, which is {A-B, TCP, 80, forward}. In this way, it will not create duplicate counters on other higher level matching flow table entries, i.e., it can be guaranteed that only the leaf node rules are matched.

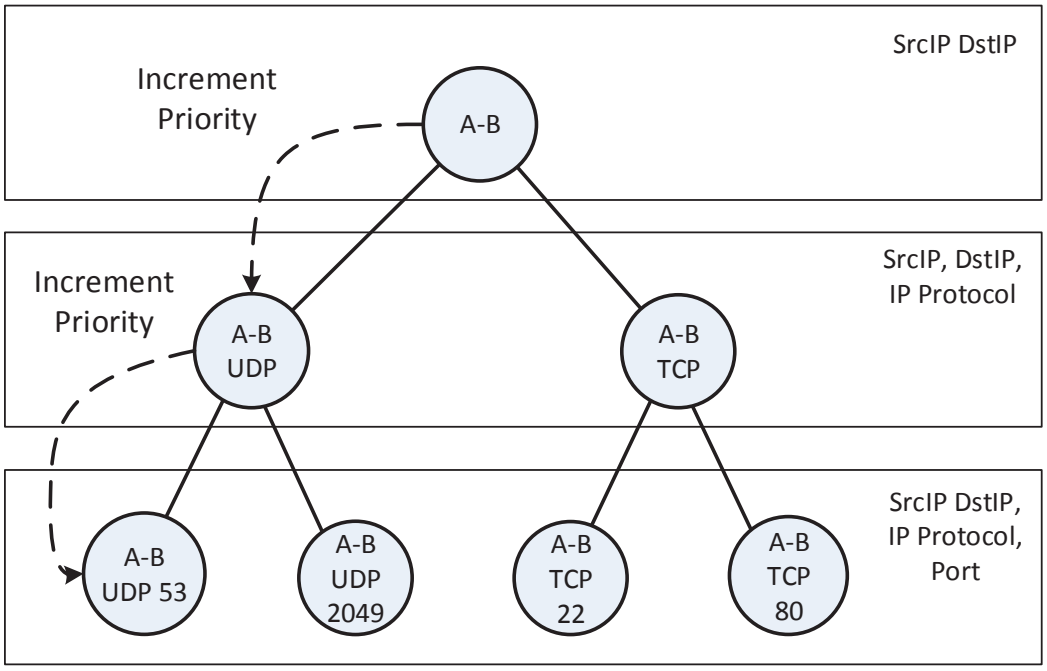


Figure 8.4: Tree-based Traffic Expanding Model.

8.2.4 Traffic Statistics Database/Dataset(DB/DS)

To maintain the history traffic information and realtime traffic information, I design DB and DS, respectively. In particular, two DSes and two DBs are implemented to store real time traffic information and history traffic information. Using these DBs and DSes, developers can acquire more customized traffic information based on the functions provided by traffic operator.

Customized DS is an empty dataset for user to develop their own realtime dataset based on their requirements. Traffic operator provides several basic methods for developers to transfer the OpenFlow statistics information to their own data structure and sequence. Those basic methods could be found in Table. 8.1 in Appendix. Developers can customize user-defined realtime data set to support their security application development.

For example, authors in [67] present a method to proactively detect the DDoS attack by using source IP address monitoring. To plug-in this detection method, an IP address monitoring data set is required. Users could use get_active_switch() function to find all the connected OVS and use get_raw_data (dpid, start_time_stamp, end_time_stamp) to query the history traffic information. IP Address Database can be generated from these history knowledge. Then, based on the detection method mentioned in this work, it can apply the customized_monitor (match, start_time_stamp, time_period, recurring = true) function to watch all the real time traffic information. I set the $\Delta_n (n = 1, 2, 3, ...)$ as the detection resolution parameter in the function, then collect IP address information during each time slot Δ_n and formulate a real time system IP address dataset. This dataset supports upper layer detection engines and algorithms to analyze the system status and report the potential DDoS attacks.

Traffic Volume DS can be used for supporting traffic volume-based detection engines or monitors to identify the anomaly behaviors in the system. Most volume-based detection engines need the traffic volume sequence to monitor the system status and detect the anomaly traffic behaviors. To fulfill this requirement, the flow table information proxy will request each OVS to return their current ingress traffic volume of each port attached by active VM every ΔT time period. Then the Traffic Information Trimmer will trim the raw data as follows formation: {*Destination IP, Source VM IP sets, Traffic Volume, Time Stamp*}.

For each active VM i in the system, let $R_i(t)$ be the summation of traffic volume record from all the source IP addresses to the destination VM i at timestamp t, where t is the discrete time index. So at timestamp t, the system overall ingress traffic volume record $V(t)$ could be calculated by $V(t) = \sum_{i=1}^{k} R_i(t)$, k is the total number of current active VMs in the system. After having the system traffic volume record at each timestamp t, to generate the system traffic volume data sequence

$\{S_t, t = 0, 1, ..., n\}$, we have $S(t) = V(t + \Delta t) - V(t)$. The request time period Δt is selected by corresponding detection engine and detection resolution.

Traceback DB is used to help Attack Source Tracer (AST) to traceback malicious insiders in the system. In traditional cloud computing system, when attack is performed by malicious insider with spoofed source IP addresses or MAC addresses, it is difficult to trace them since there is no an effective mechanism to bining IP addresses and connected physical port. Using CloudArmour, the decoupled data and control plane enable us to track the attacking source by looking up the traffic ingress port and egress port of OVS flow table, which can be tampered only if the attackers can compromise the underlying virtualization layer, e.g., the hypervisor, which is considered safe in this work.

At the back end of AST, the traceback database provides the *Source VM* and *Destination VM* pair for each flow, which means even the source IP address of traffic packet is spoofed, the system still can traceback the source virtual machine by looking up this database. To generate the source and destination pair for a flow, traffic information proxy acquires the statistics data from flow table of each OVS and traffic operator will trim the data into following structure: {*OVS ID, Ingress Port, Interface type, Destination IP, Egress port,*}. This information is stored in each OVS and its flow table. OVS ID indicates which OVS this flow belongs to. Ingress Port and Egress Port represent which port a flow comes through and to which port this flow is delivered. Interface Type shows the type of the ingress port of this flow, which could be Controller, OVS or Virtual Machine. This table tells us the abstract topology of current network connection and the true location of each OVS with its attached active VMs. It also shows where each traffic flow is exactly delivered. The system can have an accurate information of source & destination pairs from this database and the AST will trace the malicious insider based on it.

Traffic History DB can be used for upper layer security modules to develop machine learning-based or data mining-based detection and monitor applications. This database logs all the history traffic information for developers to perform deep investigation of traffic patterns and features. Some functions provided in Table. 8.1 in Appendix can be called to retrieve the historical information easily. The flow table information proxy requests all the active switches to return their flow table statistics information with a configurable time interval and store them in following structure: *Switch DPID, Request Time, Raw Data Statistics*. get_raw_data(dpid, start_time_stamp, end_time_stamp) could retrieve the historical raw traffic data based on switch *dpid* and timestamp. It will return the parsed raw data information by a python dictionary with *dpid* and time sequence as the retrieving key.

8.2.5 Traffic Packet Acquisition

Besides collecting and operating on the traffic statistics information, some security applications such as detection engine or DPI need to be deployed to inspect packet content. Thus, I introduce a modular called Traffic Packet Acquisition (TPA) to poll the real packets instead of traffic statistics information to the controller that further forward to the security applications. Users are able to specify what traffic need to be polled to the controller for inspection. The aforementioned module, traffic matcher, enables developers or users specify what packets they are looking for. After users and developers determine the packets they are interested in, new flow table rules will be injected into connected OpenFlow devices so that following matching packets will be forward to the traffic packet acquisition module.

8.3 CloudArmour Service Module Design and Implementation

This section discusses the integrated service modules that are constructed based on the TSAM. All the Python-based APIs of service module can be found in Table. 8.2 in section 8.5.

8.3.1 Traffic Monitor and Detection

In a comprehensive security solution including detection, analyzer and mitigation, traffic monitor and detection is usually the first step. The intrusion detection system can be divided into two types, signature-based and anomaly-based. Signature detection searches network traffic for a series of bytes or packet sequences known to be malicious. For example, users might use a signature that looks for particular strings within an exploit payload to detect attacks that are attempting to exploit a particular buffer-overflow vulnerability. The anomaly detection technique centers on the concept of a baseline for network behavior. The anomaly detection can perform the detection based on different aspect, such as IP address changes, destination port changes, volume changes, protocol anomalies, etc. In this traffic monitor and detection module, there are two types of information query approaches: packet-based and statistics-based. An example of how to utilize the monitor and detection module API will be given in section 8.4.

8.3.2 Attack Source Tracer Module (ASTM)

The attack behaviors need to be further analyzed after it is detected. One of the most important tasks is to determine the location of the attack sources. A common way to identify the location of attack sources is to use the trace back techniques that is based on several metrics such as IP address, MAC address. Traditional solutions,

such as packet marking technique [87, 73] for IP traceback, always need to modify the packets, which causes overhead. However, it is impossible to correctly trace back all the attack sources based on IP or MAC addresses, when the attacks perform with source address spoofing techniques. Based on the unique feature of OpenFlow that flow table history records the historical flow information, users can utilize the ingress and egress port characteristic to perform the trace back function. Since the ingress and egress ports are the native property of the OpenFlow devices, thus they can not ba spoofed if it is assumed that the OpenFlow itself is not compromised. Thus, any IP or MAC spoofing technique does not work when performing trace back functions at the port level.

Thus, I propose an IP spoofing prevented attack source trace back mechanism called Attack Source Tracer Module (ASTM). I will discuss an example to illustrate how the ASTM works in the OVS scenario (this algorithm can be also simply applied to physical OpenFlow switch scenario). The benefit of considering the OVS port over the traditional network switch port is that it is possible to know if a dedicated OVS port is connecting to a VM directly or another OVS's tunnel port. As shown in Fig. 2.1, each flow entry has a field called in_port. Through this information, the CloudArmour is able to trace back attacker VMs from the OVS that any victim directly connects. The connectivity of all OVS can be modeled as a graph. I first define a function called traceback as below:

trace_cur_OVS(cur_OVS, dst_IP, IP_prototol, port_number) = {OVS/VM}

The trace_cur_OVS function returns all entities (VM or OVS) that sending packets to the current OVS (cur_OVS) with destination IP = dst_IP, IP protocol = IP_prototol, port number = port_number. Thus I propose a trace back algorithm based on depth-first search, which is described in Algorithm 1. The input of this algorithm is the IP address of the victim, attack traffic type, e.g., TCP port 80 traffic,

and the OVS that the victim is directly connecting to. The algorithm returns a list of all attacker VMs (OVS ports that all attack VMs are attached). The Python-based API of this attack source traceback algorithm can be also found in Table. 8.2.

Algorithm 1 Attack Source Traceback Algorithm

Input: IP_proto, $port\_number$ VM_{victim}, OVS_{victim}

Output: {Set of Attacker VMs}

1: $v \leftarrow OVS_{victim}$
2: **TraceRecursion** (OVS v, Set AttackVM)
3: **if** v = VM **then**
4: ZVM.add(v)
5: return
6: **end if**
7: **for** each $x \in trace\_cur\_OVS(cur\_OVS, dst\_IP, IP\_proto, port\_number)$ **do**
8: **if** $x.visited = FALSE$ **then**
9: **TraceRecursion**$(x, list)$
10: **end if**
11: **end for**
12: **RETURN** *list*

8.3.3 Mitigation Executor Module(MEM)

The Mitigation Executor Module(MEM) is proposed to provide the mitigation actions for users. In the SDN-based mitigation, when controller insert a mitigation flow rule to the OpenFlow device, no matched overwhelming malicious traffic will be sent to the controller, and the drop action is executed either in the kernel level of the OVS or the hardware level of the physical OpenFlow switch. This mechanism can guarantee the performance of mitigation as well as the detection engine. Thus, the

mitigation capability of SDN-based one can be expected more efficient than traditional mitigation approach, e.g., Iptables NFQueue. The representative mitigation actions are the same as what I discussed in section 7.3. The performance between SDN-based mitigation and traditional mitigation approach is already evaluated in section 7.4.2.

8.4 Tutorial, Implementation and Evaluation

In this section, I provide a use case to present how cloud security solution researcher and developer can leverage the benefits of CloudArmour to construct flexible and efficient network security applications. I construct and implement a DDoS defensive system based on CloudArmour framework. At last, I evaluate the CloudArmour in the DDoS scenario. All the modules are implemented in the mininet [55] environment and the evaluation is conducted in the real Xen-based virtual networking environment.

8.4.1 A Tutorial: A DDoS Defensive Solution

DDoS Attack Model

According to the definition from WWW Security FAQ [75] on Distributed Denial of Service (DDoS) attacks: "A DDoS attack uses many computers to launch a coordinated DoS attack against one or more targets. Using client/server technology, the perpetrator is able to multiply the effectiveness of the DoS significantly by harnessing the resources of multiple unwitting accomplice computers, which serve as attack platform". The DDoS is distinguished from other attacks by its ability to deploy its weapons in a "distributed" way over the network and to aggregate these forces to create lethal traffic. DDoS can be simply divided into two categories, the bandwidth depletion attack and the resource depletion attack. The bandwidth depletion attack is designed to flood the victim's network with overwhelming traffic that prevents legit-

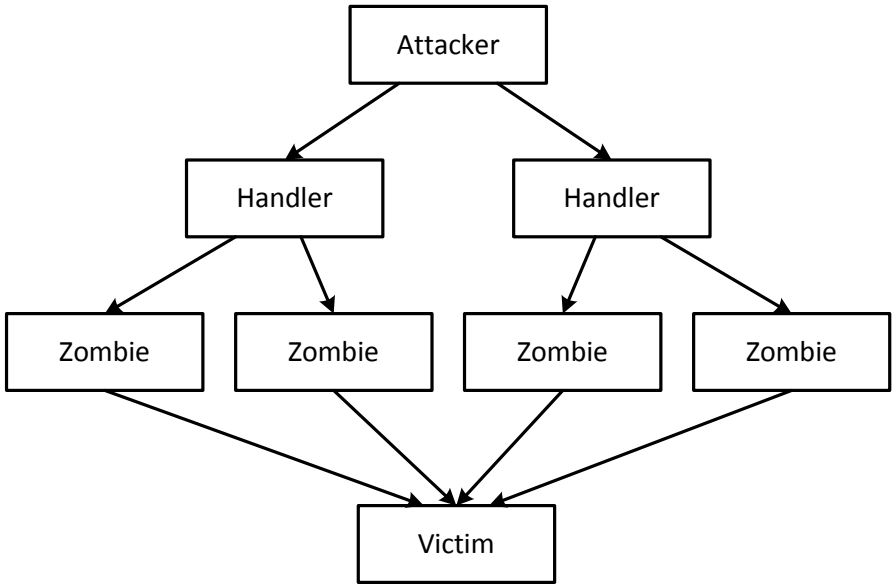


Figure 8.5: DDoS Attack Model

imate traffic from reaching the victim, while the resource depletion attack is trying to break the victim's system by either breaking the protocol running on victim's system or crashing the victim's system by sending incorrect data packets.

The attack scenario is modeled in Fig. 8.5. The DDoS attack is the most advanced form of DoS attacks, which has been regarded as the most noticeable threat to Internet security by both Kaspersky Labs and Symantec. A representative IDS/IPS system Snort [5] claims that the detection and prevention for DDoS is not as effective as for other attacks. The DDoS is distinguished from other attacks by its ability to deploy its weapons in a "distributed" way over the network and to aggregate these forces to create lethal traffic. DDoS can be simply divided into two categories, the bandwidth depletion attack and the resource depletion attack. The bandwidth depletion attack is designed to flood the victim's network with overwhelming traffic that prevents legitimate traffic from reaching the victim, while the resource depletion attack is trying to break the victim's system by either breaking the protocol running on victim's system or crashing the victim's system by sending incorrect data packets. Although

DDoS has different types of attack in terms of the way of performing attack, they still have common in nature, which is that a great number of packets from multiple zombie hosts go to the victim at the same time. This work is also based on the assumption that all DDoS traffic initiated from multiple zombies belongs to the same type.

Module Composition Flow

To construct the DDoS defensive solution based on CloudArmour modules, I first divide the defensive solution into four phases: DDoS behavior detection, attack analysis, zombie trace, and mitigation execution. These four modules are executed in an assembly line fashion. The processing phases of the DDoS defensive solution based on CloudArmour are illustrated in Fig. 8.6.

The object of DDoS detection engine is to detect the DDoS attack behavior. The detection engine can be implemented as either a signature-based approach or an anomaly-based one. The anomaly-based detection engine can be simply constructed by calling monitor_volume(match, start_time, time_period,recurring). Users can also plug-in an existing signature-based detection engine such as Snort, by calling ids_connector(match,detection_engine_ip) function to make traffic flows and detection engine connected. After the DDoS behavior is detected, further inspection needs to be performed to determine: (1) identifying the victim; (2) identifying the type of DDoS traffic initiated from zombies. Flow_table_expander(in_port, out_port, ethernet_layer, ip_layer, transport_layer) can be used to generate more elaborated traffic statistics information, which can be further analyzed to determined aforementioned information. When DDoS behavior, DDoS traffic type, and victim are all confirmed by the previous modules, attack_source_traceback (victim_ip, victim_ovs, attack_traffic_type) function is triggered to identify all DDoS traffic sources in zombie trace phase. After zombie machines are all identified by traceback function, mitigation will be performed

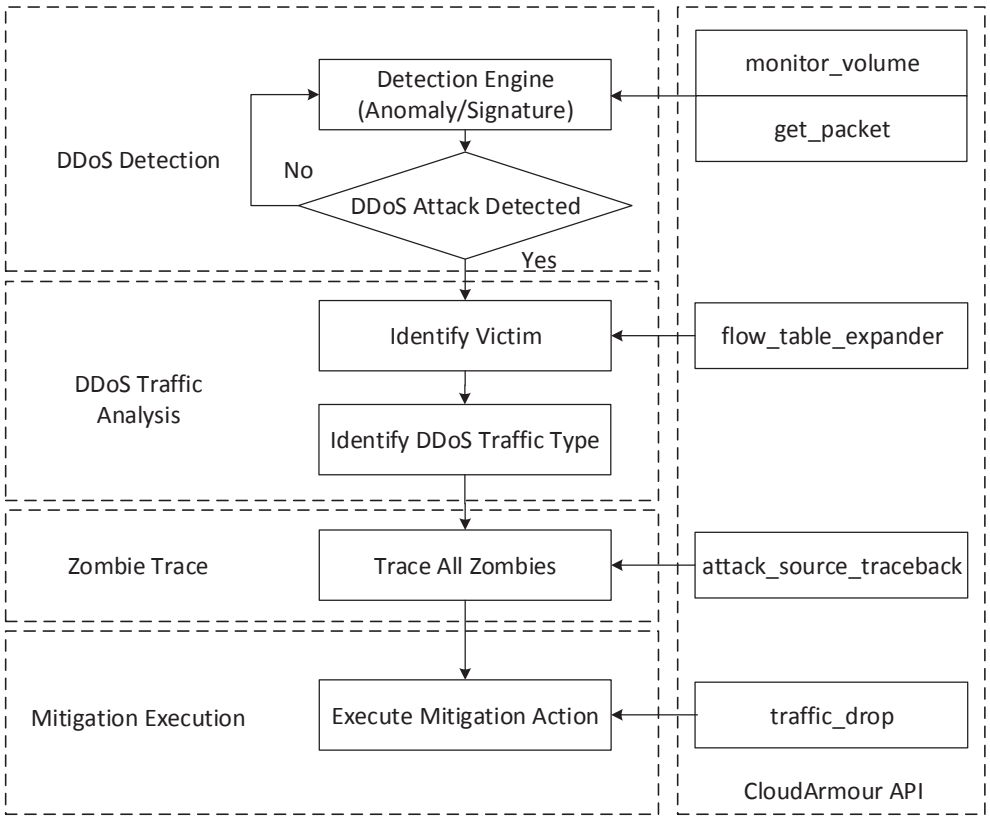


Figure 8.6: DDoS Defensive Solution Construction Flow

to stop the DDoS attacking traffic at all port entries of the OpenFlow devices.

8.4.2 Implementation and Evaluation

I build a comprehensive DDoS defensive system based on our CloudArmour framework and three major components, DDoS Detection Engine, DDoS Traffic Analyzer, Zombie Tracer and Mitigation Executor are implemented by CloudArmour APIs. Implementation is demonstrated by screenshots and evaluation validates the feasibility and flexibility of the CloudArmour framework.

Volume-based Anomaly Detection Engine

A lightweight DDoS detection engine is designed to detect the anomaly overwhelming traffic in time. This detection engine choose to apply the non-parametric Cumulative Sum (CUSUM) [24] algorithm to detect the occurrence of abrupt change of the system overall traffic volume. To simplify the detection engine and reduce the overhead, this method only focuses on the system traffic volume and alerts the potential DDoS attack activities. Similar methods could be found in [81, 67], where [81] watches the occurrence of large difference between the number of SYNs and FINs and [67] detects the anomaly increase of new source IP addresses ratio.

I use TSAM API to setup a monitor to watch the system overall traffic volume and formulate a traffic volume data set $X(t) = \{x_1, x_2, .., x_n\}$, where x_n is the whole system ingress traffic volume in the nth time interval. To make the method more general, I use the Exponential Moving Average (EMA) method to normalize the traffic volume data set. Thus, given the normalized data set $X(t)$, non-parametric CUSUM could be simplified by following equation based on [24]:

$$y_t = (y_{t-1} + X(t))^+, y_0 = 0, \tag{8.1}$$

To determine a potential flooded attack to VM i, the decision function can be presented as follows:

$$d_N(y_t) = \begin{cases} 0 & \text{if } y_t \leq N, \\ 1 & \text{if } y_t > N. \end{cases} \tag{8.2}$$

N is the flood threshold and $y_t > N$ indicates the occurrence of flood traffic in the system at time stamp t. This threshold can be calculated based on the degree of detection sensitive, which was explained in [24] and will not be elaborated here.

Seagull[39] is used to generate background traffic at a normal speed and Pktgen[51] is used to emulate DDoS traffic due to its ability of generating packets at extremely

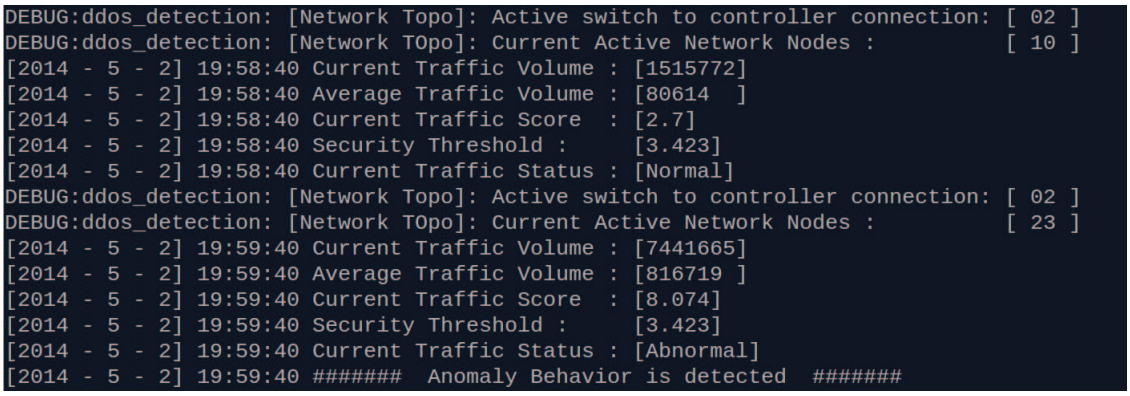


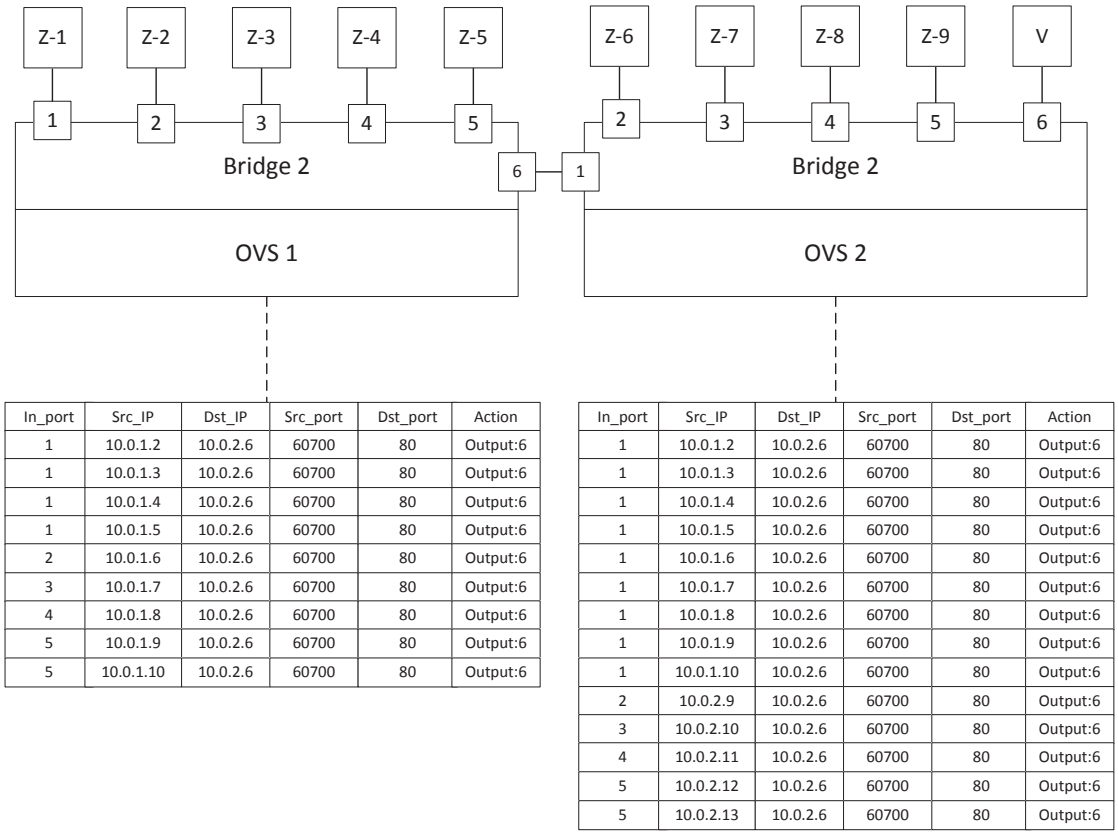
Figure 8.7: DDoS Detection Engine Implementation Screenshot

high speed in the kernel. Implementation scenario can be found in Fig. 8.8. 9 zombie machines perform DDoS attack by flooding traffic packets to the same victim at the same time.

Fig. 8.7 is the screenshot of our implemented anomaly-based detection module. I choose the detection resolution ΔT equals 60 seconds so that the DDoS detection engine queries the traffic volume, average traffic volume, and CUSUM statistics score every 60 seconds. When the CUSUM statistics score exceeds the threshold, Traffic Analyzer and Mitigation Executor will be trigged to protect the system.

IP Spoofing Attack Traceback

Attack Source Traceback module can track all the packets even they are intentionally modified to spoof security agent and other virtual machines in the cloud environment. Based on different traceback type, it can target the attackers who are sending malicious packets with specific attacking signature, for example, the Ping of Death attack is performed by changing the IP fragments information of packets headers and try to overflow receiver's memory buffer. It also can trace the distributed attackers, e.g. zombie machines in DDoS attack, which they always send huge amount of attacking traffic to victim at the same time and forge the security agent with spoofed IP ad-



In_port	Src_IP	Dst_IP	Src_port	Dst_port	Action
1	10.0.1.2	10.0.2.6	60700	80	Output:6
1	10.0.1.3	10.0.2.6	60700	80	Output:6
1	10.0.1.4	10.0.2.6	60700	80	Output:6
1	10.0.1.5	10.0.2.6	60700	80	Output:6
2	10.0.1.6	10.0.2.6	60700	80	Output:6
3	10.0.1.7	10.0.2.6	60700	80	Output:6
4	10.0.1.8	10.0.2.6	60700	80	Output:6
5	10.0.1.9	10.0.2.6	60700	80	Output:6
5	10.0.1.10	10.0.2.6	60700	80	Output:6

In_port	Src_IP	Dst_IP	Src_port	Dst_port	Action
1	10.0.1.2	10.0.2.6	60700	80	Output:6
1	10.0.1.3	10.0.2.6	60700	80	Output:6
1	10.0.1.4	10.0.2.6	60700	80	Output:6
1	10.0.1.5	10.0.2.6	60700	80	Output:6
1	10.0.1.6	10.0.2.6	60700	80	Output:6
1	10.0.1.7	10.0.2.6	60700	80	Output:6
1	10.0.1.8	10.0.2.6	60700	80	Output:6
1	10.0.1.9	10.0.2.6	60700	80	Output:6
1	10.0.1.10	10.0.2.6	60700	80	Output:6
2	10.0.2.9	10.0.2.6	60700	80	Output:6
3	10.0.2.10	10.0.2.6	60700	80	Output:6
4	10.0.2.11	10.0.2.6	60700	80	Output:6
5	10.0.2.12	10.0.2.6	60700	80	Output:6
5	10.0.2.13	10.0.2.6	60700	80	Output:6

Figure 8.8: DDoS Attack Scenario and flow table Entries

dresses. In this scenario, I develop a zombie trace application based on CloudArmour APIs to test the feasibility of our back end framework. I implement this component fitting in the scenario described in Fig 8.8.

In Fig. 8.8, it can be seen that all the VMs in the system are connected to the OVS bridge through virtual interface (VIF) and each network interface has attached to one switch port. It can be found that the number of source IP addresses is more than current active VMs, which means the IP spoofing attack is performed. This is based on the assumption that each port can be only connected with one VIF with a single IP address. Attack zombie host 1 inserts 4 flow entries in the flow table and it is sending 4 kinds of traffic to victim VM with forged source IP addresses. Zombie

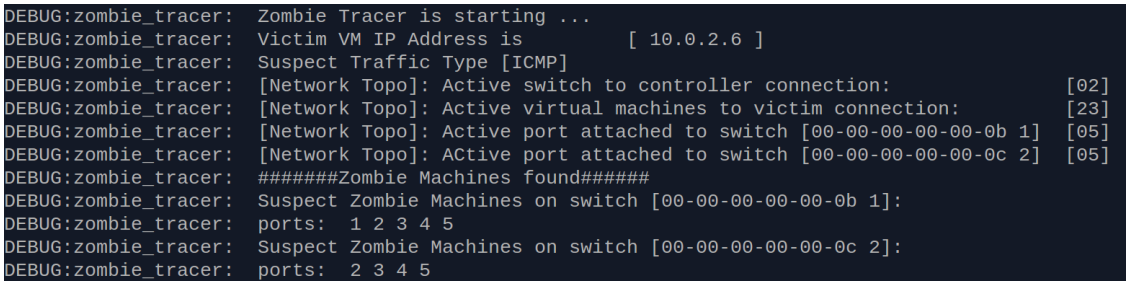


Figure 8.9: Screenshot of ASTM in DDoS Attack Scenario

machine 5 and 9 also apply IP Spoofing technique to challenge the traceback of zombie machines. In this situation, it can not track the zombie machines based on IP address because they are spoofed and changed intentionally. However, due to the assumption that OpenFlow port can not be forged, from the flow table of OVS 1, it can be seen that the source IP address {10.0.1.2, 10.0.1.3, 10.0.1.4, 10.0.1.5} come from the same ingress port and this ingress port is connected to only one active VM. So when users traceback the zombie machines, users only call the function attack_source_traceback (victim_ip,victim_ovs,ip_proto, port_number) in CloudArmour. I set the victim_ip as 10.0.2.6 and the detected ip_proto as ICMP (I leave the
port_number field blank because ICMP does not have port number), and then set the victim_ovs as the dpid of the OVS that victim attaches. This function returns all the ports of zombie machines.

Fig. 8.9 is the screenshot of the ASTM. it can be seen that the victim IP address is 10.0.2.6 and attack traffic type is ICMP. It also reports the number of current active IP addresses, i.e., 23, and the number of active OVS ports, i.e., 5, and then it tracebacks all the ports of suspect zombie machines in each OVS. From the scrrenshot, ports 1,2,3,4,5 on OVS1 and ports 2,3,4,5 on OVS2 are determined as the ports that the zombie machines are connected.

CloudAmoure Mitigation Executor

In this section, I evaluate the performance of OVS-based mitigation compared to the traditional IPS mitigation that is implemented based on the Iptable and Snort. I setup our test environment based on Fig.8.8. OpenFlow-based solution and IPS-based solution are deployed separately in two different experiment. Zombie machines are located at different servers and it can be assumed that the DDoS behavior has been detected previously. In the OpenFlow-based mitigation, Open vSwitch is planted in each physical servers and corresponding prevention rules are inserted to the flow table to tackle the DDoS malicious traffic and protect the normal traffic delivery. In the IPS solution, an IPS device, e.g. Snort agent is installed in each physical server and it is configured to drop malicious overwhelming traffic. Among all mitigation options, I choose drop action as the countermeasure approach to conduct the comparison evaluation. In this scenario, all the zombie machines are performing DDoS flooding attack and sending normal traffic at the same time. I evaluate the success rate of processing normal traffic to measure the mitigation capability of different approach when they are both under significant stress.

The evaluation result is shown in Fig. 8.10. All the packets are 100% processed by Open vSwitch even the attack rate reaches to 350,000 packets per second. This is because Open vSwitch can block all the already known malicious packets before they are forwarded. However, traditional IPS mitigation can not fully prevent the interruption of flooding traffic due to the mechanism of Iptables and Snort, traffic packets are always congested into Iptables Queue. The capability of Snort is not enough to handle all the traffic when the attack rate higher than 17,550 packets per seconds. In OpenFlow case, the detection kernel is decoupled from mitigation module, the mitigation is performed in the data plane of the OVS in line-rate, which guarantees

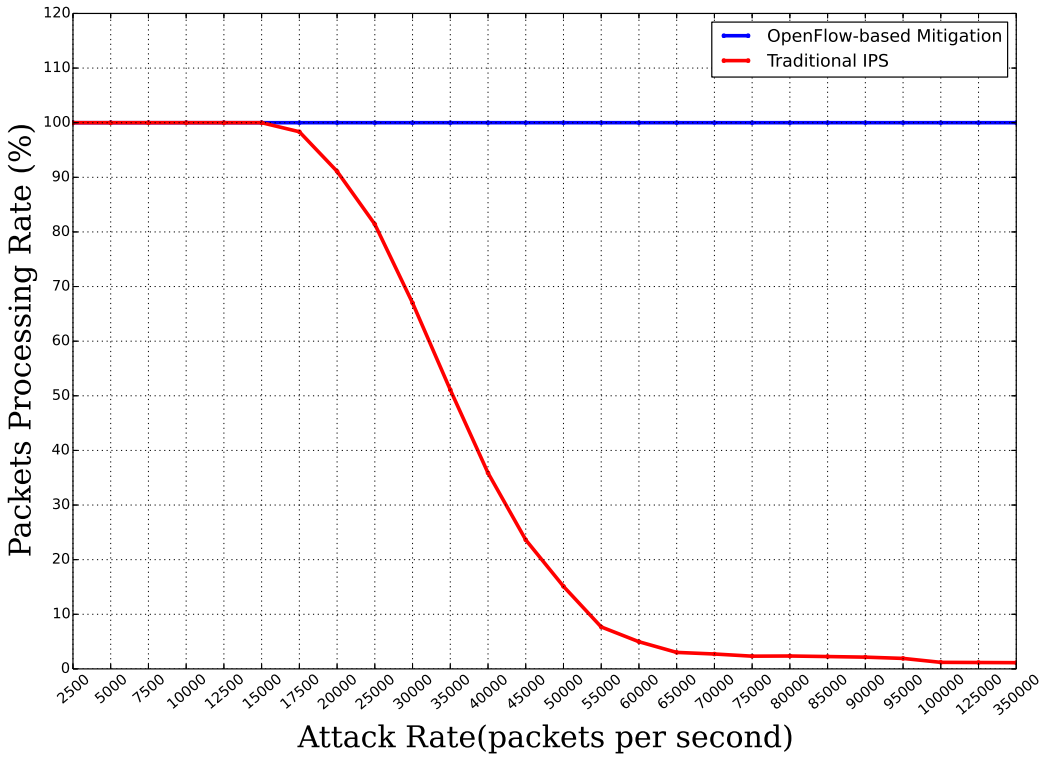


Figure 8.10: Comparison Evaluation between OpenFlow-based and Traditional Mitigation

the packet processing still can work properly even system is under overwhelming flooding attack.

Fig. 8.11 shows the performance of various mitigation options in terms of the CPU utilization. Packet generator [51] is used to generate the packets mitigated by MEM. In all mitigation approaches, TR is implemented by using destination IP & MAC rewriting; while TR with spoofing reply is implemented by rewriting not only destination IP & MAC address but also source IP & MAC of victims. Thus, the attacking traffic can be redirected to a security appliance that is able to spoof the attacker by replying the packet with victims' IP & MAC address as source address.

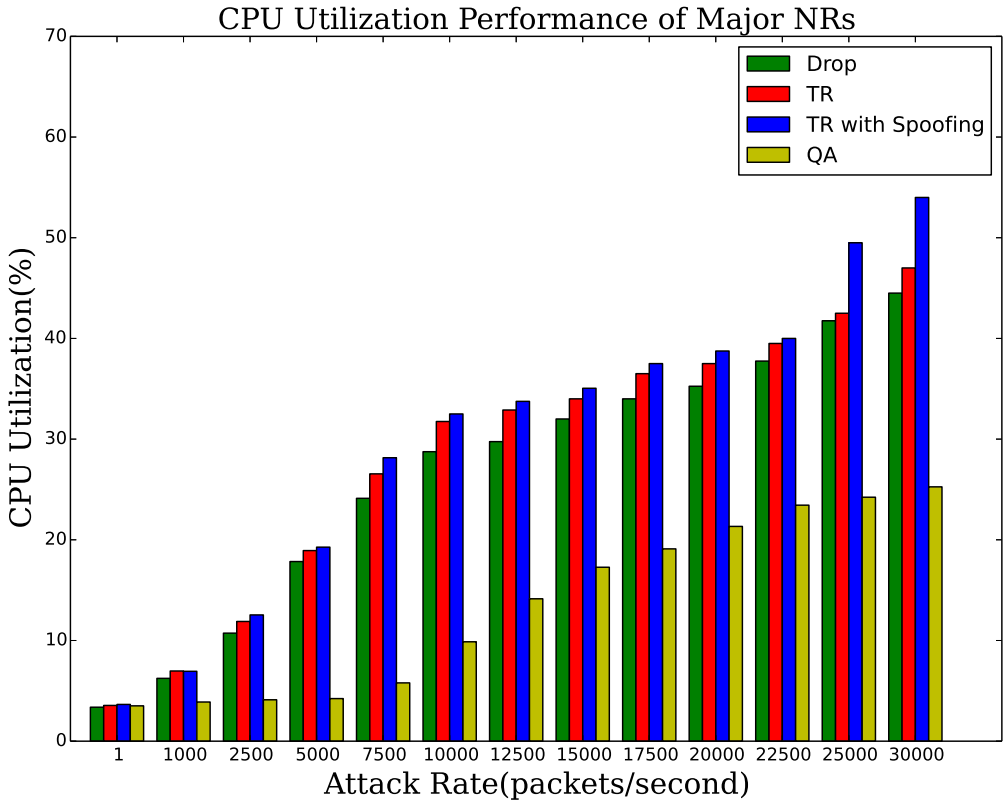


Figure 8.11: Major Mitigation Options Evaluation

TR with spoofing feature consumes a little more resources than the pure TR since OVS modifies more packet fields to enable the spoofing feature. The default NR, i.e., drop packets, consumes less system resource because the OVS does not modify the matching flow and just simply drop them (output to a non-existing virtual port in POX controller implementation). In the QA scenario, it has the best performance among all mitigations because the rate limiting action is performed based on OVS native mechanism, which means excess packets will be discarded and OVS does not have to inspect and match the packet with all kinds of fields. Since the mitigation is same as the NR I discussed in section 7.3, thus the evaluation result of the mitigation is also the same as the NR evaluation shown in Fig. 7.8.

Overall Performance Evaluation

The performance of our implemented DDoS defensive solution is showed in Fig. 8.12. X axis represent the time after starting the experiment and the Y axis represent the CUSUM statistics value y_t, which could be considered as the score of current traffic. The blue line shows the curve of CUSUM statistics score during the test and the red line shows the threshold which is computed based on the method in [19]. From the results, it can be seen that the detection engine will not generate alert for the normal traffic even there are some busty increasing and decreasing traffic

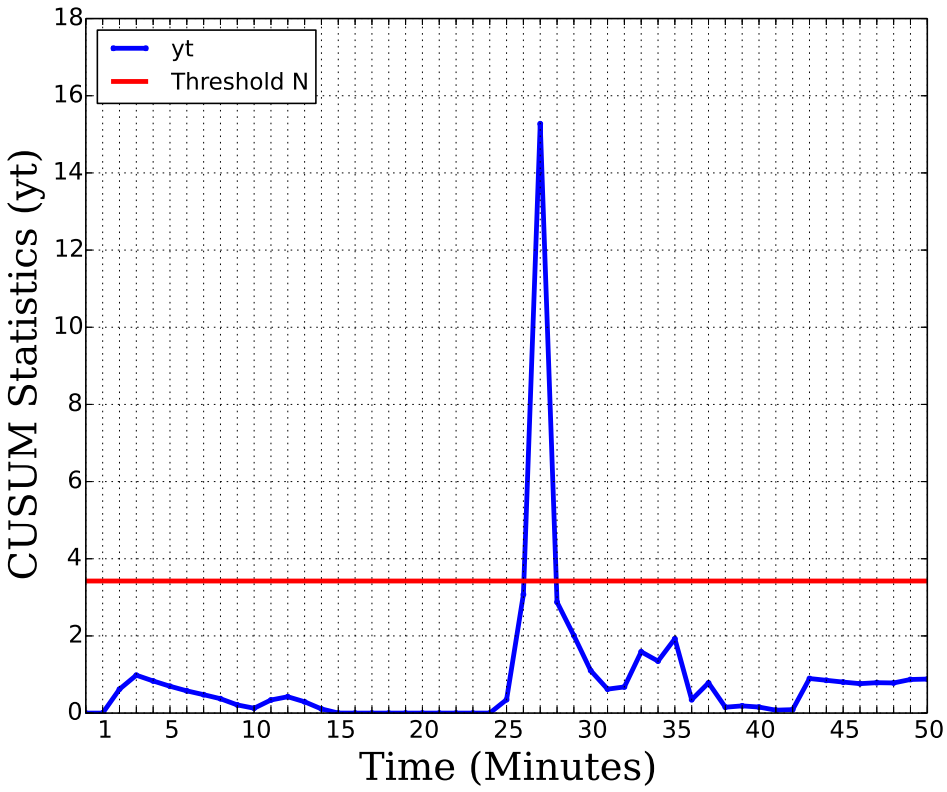


Figure 8.12: Comprehensive DDoS Defensive Solution Performance Evaluation

generated by Seagull. However, when performing DDoS attack at the 26th minutes, our detection engine swiftly detects the anomaly ingress traffic volume and trigger the ASTM and MEM for further investigation and mitigation. At the 28th minutes, CUSUM statistics score goes down under the threshold, which the DDoS attack has been successfully mitigated. Since the detection resolution is set to 60 seconds, 2 minutes processing time proves the feasibility and efficiency of the CloudArmour framework on constructing comprehensive defensive solution in virtual networking environment.

8.5 CloudArmour APIs

Table. 8.1 illustrates all Python-based APIs for TSAM modules. Table. 8.2 illustrates all Python-based APIs for service modules including detection engine, ASTM, and MEM.

Table 8.1: TSAM API Summarization

Function	Parameters	Explanation
get_active_switch		return all active connections
get_active_port	dpid	return all active ports of specified OpenFlow switch
get_port_volume	dpid, port, start_time, end_time, ingress	get history port traffic volume from start_time to end_time, time format shoud be python standard time format, ingress value is true by default and set it to false to watch the egress traffic volume of the port
monitor_port_volume	dpid, port, start_time, time_period, recurring, ingress	monitor port ingress or egress traffic volume from start_time every time_period seconds the method monitor ingress traffic by default, when ingress=false, it will monitor the egress traffic volume
monitor_volume	match, start_time, time_period, recurring	monitor traffic volume from start_time every time_period time, user needs to set the attribute of match to define the flow to be monitored and the function will return real time data set
get_volume	match, start_time, end_time, time_period, ingress	return the history traffic volume information for the matched flow from start_time to end_time, by default ingress value is true and set it to false to monitor the egress traffic volume
flow_table_expander	in_port, out_port, ethernet_layer, ip_layer, transport_layer	expand the raw flow table rules to detail flow rules automatically, set the ethernet_layer, ip_layer, transport_layer parameter to false to disable the expand for corresponding layer
get_packet	match,controller_port	send all matched packets to controller
traceback_IP	destination_IP, traffic_type	this function return all the source ip_addresses who are sending specific traffic type to destination_ip
customized_monitor	match, start_time, time_period, recurring	set up a monitor, match flow and start_time, return the match flows information every time_period
check_customized_counter	match, start_time, counter	check if the number of matched packets exceed the counter value
get_raw_data	dpid, start_time, end_tim	get raw traffic statistics information of selected switch from start_time to end_time. It returns a python dictionary with dpid and record time as the key

Table 8.2: CloudArmour Application Module API Summarization

Function	Parameters	Explanation
ids_connector	match, detection_engine_ip	get the matched packet from the OVS and forward it to the third party detection engine
traffic_volume_monitor	ovs, port, threshold	set up a monitor to watch the system traffic volume, when the traffic exceed the threshold, return a notification
customized_counter	match, start_time_stamp, counter	set up a counter for matched flow, when the number of matched flows reach the threshold, it will return a notification
attack_source_traceback	victim_ip, victim_ovs, ip_proto, port_number	return the list of port and the attached OVS attack_traffic_type is needed to traceback the source attackers from victim_ovs
traffic_drop	match	drop all the traffic based on the match fields
traffic_redirect	match, destination_ip	redirect all the matched traffic to new destination_ip
traffic_redirect_spoof	match, destination_ip, spoof_ip	redirect all the matched traffic to new destination_ip and change the source ip to spoof_ip
traffic_qos	dpid, port, qos_rate, ingress, egress	set the port of openvswitch ingress and egress traffic rate ingress and egress traffic rate is limited by default, set ingress or egress to false to disable the rate limit

Chapter 9

CONCLUSION

9.1 MobiCloud: a Geo-distributed Mobile Cloud Computing Platform

The proposed MobiCloud system is able to provide resources in terms of computing, storage, and networking that greatly enhances the capability of mobile devices. It combines network-based storage, Xen virtualization, and OpenFlow based network management solutions into one smart system, and that has not been seen previously to our best knowledge. Also, an example of system experience is given to better state the capabilities of MobiCloud. Finally, the performance evaluation of the MobiCloud shows expected results.

9.2 Constructing Virtual Networks in a Geo-Distributed Programmable Layer-2 Networking Environment (G-PLaNE)

After the MobiCloud system is design and implemented, I introduce a geo-distributed cloud resource provisioning system called G-PLaNE that is discussed in terms of system components, network architectures, and etc. Virtual network creation, as a major service provided by G-PLaNE, is explained from two perspectives, intra-domain virtual network and an inter-domain virtual network. A concrete example, SeRViTR, is given to illustrate how virtual network construction can help in the current research. G-PLaNE posts an efficient and flexible way to construct a virtual network in a geo-distributed cloud environment to enable more research capabilities.

9.3 Mobile Cloud User-Centric Model

After both MobiCloud and G-PLaNE are established, I investigate the service model of such MCC system. I first explore the motivation of emerging MCC and summarize three current MCC service models. Then, the transformation from traditional Internet cloud to mobile cloud is discussed by comprehensively defining the MCC, which is not merely a combination of cloud computing and mobile computing. Representative works are also summarized according to service models I classified. Finally, the future MCC design principle as well as a conceptual Human-centric MaaR approach including service model and a easy to understand example are proposed.

9.4 SDNIPS: Enabling a SDN-Based Intrusion Prevention System in Clouds

I propose an SDN-based Intrusion Prevention System called SDNIPS in the cloud virtual networking environment. It inherits the intrusion detection capability from Snort and flexible network reconfiguration from SDN. SDNIPS is firstly compared with traditional IPS from principle perspective and the real world evaluation. NR actions are also designed and developed based on OVS and POX controller. The evaluation such as NIDS performance, performance comparison between the proposed SDNIPS and Iptables/Snort IPS, validates the superior of proposed solution. I also evaluate the representative NR actions to prove its feasibility and efficiency. Finally, NSaaS are proposed to enable the security application as a service with elasticity and scalability considered.

9.5 CloudArmour: Constructing a SDN-based Defensive System in the Virtual Networking Environment

CloudArmour is a new SDN-based module composition framework to comprehensively construct security defensive solutions in virtual networking environment. It is to simply deliver the SDN-based security solution by providing efficient service modules. The proposed TSAM enables users access the customized traffic statistics information and real traffic sequence so that traffic monitor and detection capability can be delivered. Several service modules are design and implemented to provide detection, attack sources traceback, and mitigation services. A DDoS defensive system is given as an example to illustrate how to construct the comprehensive security solution based on the CloudArmour. The implementation and evaluation show the feasibility and functionality of the proposed system.

9.6 Limitations and Future Works

The limitations and corresponding future works of the proposed research are listed as follows:

- **OAuth 2.0 Integration** The OAuth 2.0 [62] authorization protocol enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. It helps manage MobiCloud users accessing both inside and outside services. Besides that, OAuth could work together with an existing Kerberos based ID management system, such as Active Directory, which provides more flexible and scalable capability for MobiCloud ID management.

- **Dynamic VM Migration.** Based on a mobile users' geographic location in MobiCloud and G-PLaNE system, its VM can be migrated from its home cluster to a guest cluster for better performance. The VM migrations are operated using the inter-cluster networking system. In this way, VMs' migrations are protected during the migration procedure, in which the VMs will not be exposed to the public domain.

- **Detection Synchronization.** Current SDNIPS does not provide the multiple detection engines synchronization capability, which means that multiple detection engines are able to detect one attack behavior. Current IDS, even multiple IDS distributed at different cluster, can not provide the collaborative IDS capability.

- **RESTful API.** Current CloudArmour implementation is based on the POX in python. All implemented APIs are python-based. To increase the availability of the proposed system, it is expected to provide the RESTful API so that all languages supporting the normal HTTP method are compatible with CloudArmour system.

REFERENCES

[1] Back Track Linux. http://www.backtrack-linux.org.

[2] hping3. http://linux.die.net/man/8/hping3.

[3] KVM. http://www.linux-kvm.org.

[4] Mausezahn User's Guide. http://www.perihel.at/sec/mz/mzguide.htmg.

[5] SourceFire Inc. http://www.snort.org.

[6] Suricata Inc. http://suricata-ids.org.

[7] inMon Flowtrend. http://www.inmon.com/products/sFlowTrend.php.

[8] ManageEngine. http://www.manageengine.com/.

[9] ProtoGENI . http://www.protogeni.net/trac/protogeni.

[10] The Global Environment for Network Innovations (GENI). http://groups.geni.net.

[11] Vyatta. http://www.vyatta.com/.

[12] Zscaler Inc. http://www.zscaler.com/.

[13] S. Akim, A. Jones, X. Yu, J. Busch, G. Fyffe, P. Graham, and P. Debevec. A cell phone based platform for facial performance capture. In *ACM SIGGRAPH 2012 Posters*, page 35. ACM, 2012.

[14] CSA Cloud Security Alliance. Top threats to cloud computing v1.0. In *White Paper*, 2010.

[15] Amazon Inc. http://http://aws.amazon.com/ec2/.

[16] Anish Arora, Emre Ertin, Rajiv Ramnath, Mikahil Nesterenko, and William Leal. Kansei: A high-fidelity sensing testbed. *In Proceedings of the DETER Community Workshop on Cyber-Security and Test*, 10:35–47, 2006.

[17] L. Aymerich-Franch, C. Karutz, and J.N. Bailenson. Effects of facial and voice similarity on presence in a public speaking virtual environment.

[18] Jeffrey R. Ballard, Ian Rae, and Aditya Akella. Extensible and Scalable Network Monitoring Using OpenSAFE. In *INM/WREN*, 2010.

[19] Michle Basseville and Igor V. Nikiforov. Detection of abrupt changes: Theory and application, 1993.

[20] Andrey Belenky and Nirwan Ansari. Ip traceback with deterministic packet marking. *IEEE Communications Letters*, 7(4):162–164, 2003.

[21] Terry Benzel, Robert Braden, Dongho Kim, and Clifford Neuman. Design, deployment, and use of the deter testbed. In *In Proceedings of the DETER Community Workshop on Cyber-Security and Test*, August 2007.

[22] M. Billinghurst and H. Kato. Collaborative augmented reality. *Communications of the ACM*, 45(7):64–70, 2002.

[23] R. Braga, E. Mota, and A. Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408–415, Oct 2010.

[24] E. Brodsky and B.S. Darkhovsky. *Nonparametric Methods in Change Point Problems*. Mathematics and Its Applications. Springer, 1993.

[25] Roy Campbell, Indranil Gupta, Mike Heath, Steve Ko, Michael Kozuch, Marcel Kunze, Thomas Kwan, Kevin Lai, Hing Yan Lee, Martha Lyons, Dejan Milojicic, David OHallaron, and Yeng Chai Soh. Open cirrus: Cloud computing testbed: Federated data centers for open source systems and services research. In *Proceedings of the USENIX Hotcloud*, June 2009.

[26] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: A platform for educational cloud computing. In *The 40th Technical Symposium of the ACM Special Interest Group for Computer Science Education (SIGCSE)*, 2009.

[27] Yanghee Choi. Implementation of content-oriented networking architecture (cona): A focus on DDoS countermeasure.

[28] B.G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[29] Chun-Jen Chung, Pankaj Khatkar, Tianyi Xing, Jeongkeun Lee, and Dijiang Huang. Nice: Network intrusion detection and countermeasure selection in virtual network systems. In *IEEE Transactions on Dependable and Secure Computing (TDSC), Special Issue on Cloud Computing Assessment*, 2013.

[30] Citrix System Inc. `http://www.citrix.com/`.

[31] Adrian Covert. Google Drive, iCloud, Dropbox and More Compared: Whats the Best Cloud Option? In *Technical Review*, 2012.

[32] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. Maui: Making smartphones last longer with code offload. In *MobiSys*, 2010.

[33] Christos Douligeris and Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643 – 666, 2004.

[34] Rasool Fakoor, Mayank Raj, Azade Nazi, Mario Di Francesco, and Sajal K. Das. An Integrated Cloud-based Framework for Mobile Phone Sensing. In *Proceedings of the ACM SIGCOMM MCC workshop*, 2012.

[35] Jrme Franois, Issam Aib, and Raouf Boutaba. Firecol: a collaborative protection network for the detection of flooding ddos attacks. *IEEE/ACM Trans. Netw.*, 20(6):1828–1841, 2012.

[36] Michael T. Goodrich. Probabilistic packet marking for large-scale ip traceback. *IEEE/ACM Trans. Netw.*, 16(1):15–24, February 2008.

[37] Google Inc. `http://www.google.com/wallet`.

[38] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenkes. Nox: Towards an operating system for networks. In *ACM SIGCOMM Computer Communication Review*, July 2008.

[39] HP Opencall Software. `http://gull.sourceforge.net/`.

[40] D. Huang, S. Ata, and D Medhi. Establishing secure virtual trust routing and provisioning domains for future internet. In *Proc. of IEEE Globecom 2010 Conference (Next Generation Networking Symposium)*, 2010.

[41] Dijiang Huang, Tianyi Xing, and Huijun Wu. Mobile cloud computing service models: A user-centric approach. In *IEEE networks*, 2013.

[42] Network Instruments. Extending network visibility by leveraging NetFlow and sFlow technologies, February 2011. White Paper.

[43] David Irwin, Navin Sharma, Prashant Shenoy, and Michael Zink. Towards a virtualized sensing environment. In *Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom)*, 2010.

[44] Jafar Haadi Jafarian, Ehab AI-Shaer, and Qi Duan. Openflow random host mutation: Transparent moving target defense using software defined networking. In *HotSDN*, 2012.

[45] Rishi Kappor, George Porter, Malveeka Tewari, Geoffrey M. Voelker, and Amin Vahdat. Chronos: Predictable low latency for data center applications. In *SOCC*, 2012.

[46] Kevin Kell. Ec2 security revisited. In *Online Blog*, 2013.

[47] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, Sep 1990.

[48] Boris Koldehofe, Frank Durr, Muhammad Adnan Tariq, and Kurt Rothermel. The power of software-defined networking: Line-rate content-based routing using openflow. In *MW4NG*, 2011.

[49] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. ThinkAir: Dynamic resource alloation and parallel execution in cloud for mobile code offloading. In *Proceedings of The 31st Annual IEEE International Conference on Computer Communications (IEEE INFOCOM)*, 2012.

[50] Dijiang Huang Le Xu, Vijayakrishnan Nagarajan and Wei-Tek Tsai. Secure Web Referral Services for Mobile Cloud Computing. In *IEEE International Symposium on Mobile Cloud, Computing, and Service Engineering (IEEE MobileCloud)*, 2013.

[51] Linux Foundation. `http://www.linuxfoundation.org/collaborate/workgroups/networking/pktgen`.

[52] Xuan Liu, Akira Wada, Tianyi Xing, Parikshit Juluri, Yasuhiro Sato, Shingo Ata, Dijiang Huang, and Deep Medhi. SeRViTR: A framework for trust and policy management for a secure Internet and its proof-of-concept implementation. In *Proc. of 4th IEEE/IFIP International Workshop on Management of the Future Internet (ManFI'2012)*, Maui, Hawaii, April 2012.

[53] Mark Walshy. Gartner: Mobile to outpace desktop web by 2013, January.

[54] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. In *ACM SIGCOMM Computer Communication Review*, April 2008.

[55] Mininet. `http://mininet.org/`.

[56] Winston Morton. Intrusion prevention straitegies for cloud computing. 2011.

[57] Murphy McCauley. `http://www.noxrepo.org/pox/`.

[58] Marcelo Ribeiro Nascimento, Christian Esteve Rothenberg, Marcos Rogeris Salvador, and Mauricio Ferreira Magalhaes. Quagflow: Partnering quagga with openflow. In *SIGCOMM Poster*, 2010.

[59] NetPerf. `http://www.netperf.org`.

[60] Dirk Neumann, Christian Bodenstein, Omer F. Rana, and Ruby Krishnaswamy. Stacee:enhancing storage clouds using edge devices. In *Proceedings of the 1st ACM/IEEE workshop on Autonomic computing in economics*, 2010.

[61] Nike Inc. `http://www.nike.com`.

[62] OAuth Community. `http://oauth.net`.

[63] Jon Oberheide, Evan Cooke, and Farnam Jahanian. CloudAV: N-Version Antivirus in the Network Cloud. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, July 2008.

[64] OpenFlow. Openflow switch specification 1.4. In *Open Networking Fundation*, 2013.

[65] OSGi Alliance. `http://www.osgi.org/`.

[66] P.Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfieldh. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP)*, 2003.

[67] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Proactively detecting distributed denial of service attacks using source ip address monitoring. In *In Proceedings of the Third International IFIP-TC6 Networking Conference (Networking 2004*, pages 771–782. Springer, 2004.

[68] Larry Peterson, Andy Bavier, Marc Fiuczynski, and Steve Muir. Experiences building planetlab. In *Proceedings of the Seventh Symposium on Operating System Design and Implementation (OSDI)*, November 2006.

[69] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2005.

[70] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Proceedings of the 2012 20th IEEE International Conference on Network Protocols (ICNP)*, ICNP '12, pages 1–6, Washington, DC, USA, 2012. IEEE Computer Society.

[71] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avantguard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 413–424, 2013.

[72] G. Singh, J.E. Swan, J.A. Jones, and S.R. Ellis. Depth judgments by reaching and matching in near-field augmented reality. In *Virtual Reality Workshops (VR), 2012 IEEE*, pages 165–166. IEEE, 2012.

[73] Dawn Xiaoding Song and A. Perrig. Advanced and authenticated marking schemes for ip traceback. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 878–886 vol.2, 2001.

[74] Hamed Soroush, Nilanjan Banerjee, Aruna Balasubramanian, Mark D. Corner, Brian Neil Levine, and Brian Lynn. Dome: A diverse outdoor mobile testbed. In *Workshop on Hot Topics of Planet-Scale Mobility Measurements (HotPlanet)*, June 2009.

[75] Lincoln D. Stein and John N. Stewart. In *The World Wide Web Security FAQ, Version 3.1.2*, Feb 2002.

[76] J.P.G. Sterbenz, D. Medhi, B. Ramamurthy, C. Scoglio, D. Hutchison, B. Plattner, T. Anjali, A. Scott, C. Buffington, G.E. Monaco, D. Gruenbacher, R. McMullen, J.P. Rohrer, J. Sherrell, P. Angu, R. Cherukuri, H. Qian, and N. Tare. The Great Plains Environment for Network Innovation (GpENI): A programmable testbed for future Internet architecture research. In *Proc. of 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, pages 428–441, Berlin, Germany, May 2010.

[77] Patrick Stued, Iqbal Mohomed, and Doug Terry. Wherestore: location-based data storage for mobile devices interacting with the cloud. In *MCS '10 Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, 2010.

[78] Ming-Yang Su, Gwo-Jong Yu, and Chun-Yuen Lin. A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach. In *Computers & Security*, pages 301–309, Feb 2009.

[79] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. Network architecture for joint failure recovery and traffic engineering. In *SIGMETRICS*, 2011.

[80] Niels LM van Adrichem, Christian Doerr, and Fernando A Kuipers. Opennetmon: Network monitoring in openflow software-defined networks.

[81] Haining Wang, Danlu Zhang, and K.G. Shin. Detecting syn flooding attacks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1530–1539, June 2002.

[82] Huijun Wu and Dijiang Huang. Personal on-demand execution environment for mobile cloud computing. In *http://poem.mobicloud.asu.edu*, 2013.

[83] Tianyi Xing, Dijiang Huang, Deep Medhi, and Shingo Ata. Mobicloud: a geo-distributed mobile cloud computing platform. In *Proceedings of 8th International Conference on Network and Service Management*, 2012.

[84] Tianyi Xing, Dijiang Huang, Le Xu, Chun-Jen Chung, and Pankaj Khatkar. Snortflow: A openflow-based system in cloud environment. In *GENI Research and Educational Experiment Workshop, GREE*, 2013.

[85] Tianyi Xing, Xuan Liu, Chun jen Chung, Akira Wada, Shingo Ata, Dijiang Huang, and Deep Medhi. Constructing virtual networking environment in a geo-distributed programmable layer-2 networking environment (g-plane. In *in IEEE 5th International Workshop on the Network of the Future (FutureNet-V*, 2012.

[86] Tianyi Xing, Zhengyang Xiong, Dijiang Huang, and Deep Medhi. Sdnips: Enabling software-defined networking based intrusion prevention system in clouds. In *Proceedings of 8th International Conference on Network and Service Management*, 2014.

[87] A. Yaar, A. Perrig, and D. Song. Stackpi: New packet marking and filtering mechanisms for ddos and ip spoofing defense. *Selected Areas in Communications, IEEE Journal on*, 24(10):1853–1863, Oct 2006.

[88] Guanhua Yan, Ritchie Lee, Alex Kent, and David Wolpert. Towards a bayesian network game framework for evaluating DDoS attacks and defense. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 553–566, New York, NY, USA, 2012. ACM.

[89] S. Yu, Y. Tian, S. Guo, and D. Wu. Can we beat DDoS attacks in clouds?, 2013.

[90] Shui Yu, Wanlei Zhou, Robin Doss, and Weijia Jia. Traceback of ddos attacks using entropy variations. *Parallel and Distributed Systems, IEEE Transactions on*, 22(3):412–425, 2011.