9-1-2015

Policy-based Information Sharing using Software-Defined Networking in Cloud Systems

VISWANATH NANDINA

Viswanath Nandina

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Gregory L. Heileman , Chairperson

Wei Winnie Shu

Nasir Ghani

Christopher C. Lamb

Policy-based Information Sharing using Software-Defined Networking in Cloud Systems

by

Viswanath Nandina

B.Tech., Electronics Engineering, Uttar Pradesh Technical University, 2006
M.S., Computer Engineering, University of New Mexico, 2010

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico

Albuquerque, New Mexico

July 2015

Dedication

To my parents, family and friends.

Acknowledgments

It is quite impossible to achieve all that you can and must without the help of others. I am grateful to my parents for their unconditional love and faith in me.

I would like to express my heartfelt gratitude for my longtime advisor and mentor Dr. Gregory Heileman and his constant support, inspiration, encouragement, and guidance throughout my academic career at UNM. He has been sublimely pivotal in my evolution and has motivated me to strive for excellence.

I thank all the members of the Informatics Group including Dr. Chris Lamb, Dr. Pramod Jamkhedkar, Dr. Edward Nava and Dr. Jose Marcio Luna for collaborating in my research endeavors. I would also like to thank my other committee members: Dr. Nasir Ghani and Dr. Winnie Shu. Finally, I would like to thank Dr. Juan Antonia Elices Crespo, Aman Sawhney, Ricardo Piro-Rael and Craig Dubyk, who have provided very helpful reviews and continuous encouragement.

Policy-based Information Sharing using Software-Defined Networking in Cloud Systems

by

Viswanath Nandina

B.Tech., Electronics Engineering, Uttar Pradesh Technical University, 2006

M.S., Computer Engineering, University of New Mexico, 2010

Ph.D., Engineering, University of New Mexico, 2015

Abstract

Cloud Computing is rapidly becoming a ubiquitous technology. It enables an escalation in computing capacity, storage and performance without the need to invest in new infrastructure and the maintenance expenses that follow. Security is among the major concerns of organizations that are still reluctant to adopt this technology: The cloud is dynamic, and with so many different parameters involved, it is a difficult task to regulate it. With an approach that blends Usage Management and Statistical Learning, this research yielded a novel approach to mitigate some of the issues arising due to questionable security, and to regulate performance (utilization of resources).This research also explored how to enforce the policies related to the resources inside a Virtual Machine(VM), apart from providing initial access control. As well, this research compared various encryption schemes and observed their behavior in the cloud. We considerd various components in the cloud to deduce a multicost function, which in turn helps to regulate the cloud.

While guarenteeing security policies in the cloud, it is esential to add security to the network becuase the virtual cloud and SDN tie together. Enforcing network-wide policies has always been a challenging task in the domain of communication networks. Software-defined networking (SDN) enables the use of a central controller to define policies, and to use each network switch to enforce policies. While this presents an attractive operational model, it uses a very low-level framework, and is not suitable for directly implementing high-level policies. Therefore, we present a new framework for defining policies and easily compiling them from a user interface directly into OpenFlow actions and usage management system processes. This demonstrated capability allows cloud administrators to enforce both network and usage polices on the cloud.

Contents

List of Figures

List of Tables

Glossary

AES	Adavanced Encryption Standard
API	Application Program Interface
AS	Autonomous System
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CLI/GUI	Command Line Interface/ Graphical User Interface
DES	Data Encryption Standard
DoS	Denial of Service
DRM	Digital Rights Management
ECB	Electonic Block Chaining
EC2	Elastic Cloud Compute
FSL	Flow-based Security Langauge
HSM	Hardware Security Modules
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service

IDS	Intrusion Detection System
IP	Internet Protocol
ISP	Internet Service Provider
L2	Layer 2/Data Link layer
LDAP	Lightweight Directory Access Protocol
MAC	Media Access control address
MIMO	Multi-Input-Multi-Output
NAT	Network Address Translation
NFV	Network Function Virtualization
NIST	National Institute of Standard Technology
NOS	Network Operating System
OFB	Output Feedback
P2P	Peer-to-peer
PaaS	Platform as a Service
PDF	Probability Distribution Function
REST	Representational State Transfer
TCP	Transmission Control Protocol
TCSEC	Trust Computer System Evaluation Criteria
TLS/SSL	Transport Layer Security / Secure Sockets Layer
S3	Simple Storage Service

SAML	Security Assertion Markup Langauge
SaaS	Software as a Service
SDN	Software-Defined Networking
SLA	Service Level Agreement
SQL	Structured Query Language
UDP	User Datagram Protocol
UM	Usage Management
UMM	Usage Management Mechanism
VLAN	Virtual Local Area Network
VM	Virtual Machine
XML	Extensible Markup Language

Chapter 1

Introduction

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network. In other words, cloud computing is a service that enables online data storage, so it can be accessed using multiple devices from anywhere in the world. Cloud computing has revolutionized the way we compute and store information, from storing and sharing ski trip pictures, to massive collaboration of employees in a company that spans the globe. The cloud has made our lives more convenient, in that we no longer have to worry about how and where we store our data.

Cloud computing is troubled by infrastructural failures and potential security problems, and over the past few years even major infrastructure providers—from Amazon to Microsoft—have suffered from extensive and unexpected system downtime [1]. This has led to the development of tools like Amazon's CloudWatch, which can monitor infrastructure it provides for failures, and secure additional infrastructure when required.

Many organizations need to share confidential information, and hence require assurance that their data cannot be altered by intentional attacks. Because the current cloud system is unable to provide varying levels of security for various types of data, organizations and users have been hesitant to adopt cloud computing. This has led some of them to invest in their own computing enterprise systems with rigorous security and reliability protections. Thus, a global demand exists for an automatic control system to enforce security rules so that operations can be directed to private or public cloud systems.

The rise in popularity of cloud computing and interest in using leased hardware and software services has resulted in an increased demand for assured information sharing. Infrastructure as a Service (IaaS) cloud services are becoming a lucrative option because they are scalable, offer hardware independence and geographical independence, and have no single point of failure. This eliminates the need to maintain one's own hardware resources and provides easy access for computing needs. In an IaaS cloud, users are limited to security measures that can be incorporated into the user's VM image [2]. No concrete mechanism exists to translate service-level agreements into the cloud allocation chain.

To secure data transmission within the cloud, a solid set of policies are required because encryption and secure protocols are not context-oriented. Since different privacy risks exist in different cloud domains, operational context is very important from a privacy perspective [3]. There is an increased need for privacy mechanisms, but enforcement of the policy within the cloud is difficult because there is no direst access to the hardware [4]. Zissis et al. address some of the security issues in cloud computing: they use a trusted third-party solution that includes the SSL protocol for encryption and uses LDAP for access, cryptographic separation of data, and certificate-based authorization [5]. Takabi et al. describe the various security and privacy challenges in the cloud domain, which describes the problem better [6].

A security mechanism able to control who can access information, how information is used, and how the information is transmitted over a network is in high demand. Since all system components may not be co-located, a policy-based approach that can regulate data access and data transmission and effectively comply with confidentiality and integrity is indispensable. In other words, a comprehensive system that includes both an access control and continuous policy-based enforcement capability is desirable.

The solution proposed in this research is an abstraction that utilizes usage management (UM) and SDN to solve the problems described above. UM provides dynamic security by continuously monitoring policy related to resources, taking into account any change in context, and enforcing appropriate action to ensure that security needs are upheld [7]. SDN helps assure that traffic going in and out of the cloud is secure.

1.1 Software-Defined Networking vs Traditional Networking

SDN enables the deployment, management, and operation of networks by use of a standards-based open architecture and its supporting open-source interfaces. It is an architecture that enables programmability of the network. SDN-based systems have well-defined control as well as a data plane offering flexible management interfaces, as opposed to traditional network management (APIs, scripts and software packages) that is dependent on the mercy of vendor-specific hardware. SDN also allows for better cloud security engineering because of its ability to control the network [8].

The notion behind SDN is to separate the control plane of network hardware from the data plane. In general, the point is to let the software, separate from the data plane, define how the data flows rather than configuring and instructing each and every piece of network hardware. A basic switch forwards all traffic based on "learned" MAC address reachability information, whereas an open flow switch permits selective forwarding of data based on inputs from the controller.

Today, network programmability is highly dependent on vendor-specific implementations connected to their hardware. Therefore, it is difficult to provide an end-to-end SDN solution. Users desire to break the shackles of proprietary vendor-specific interfaces to fully utilize the SDN capability. One of the most popular SDN specifications used by many developers is OpenFlow [9]. OpenFlow was initially proposed as an academic research project, led by Nick McKeown (Stanford University) with the goal of enabling scientists to run network experiments in real-world campus networks.

As depicted in Figure 1.1, OpenFlow was developed to standardize communication between the control plane and the data plane (Southbound interface). Users can access OpenFlow-enabled switches on the open interface (Northbound interface) without having to deal with the internal data plane.

Figure 1.2 depicts entries in a flow table [9]. Each OpenFlow-enabled switch implements a flow table, and the OpenFlow protocol is used to access the flow table. Each entry in the table defines rules to match a packet (based on IP address, MAC address, TCP/UDP port, VLAN, and switch port) and the action to be taken upon a match (drop the packet, forward to the controller, forward the packet to particular switch port(s) and/or send to normal processing pipeline). This allows the network administrator to implement

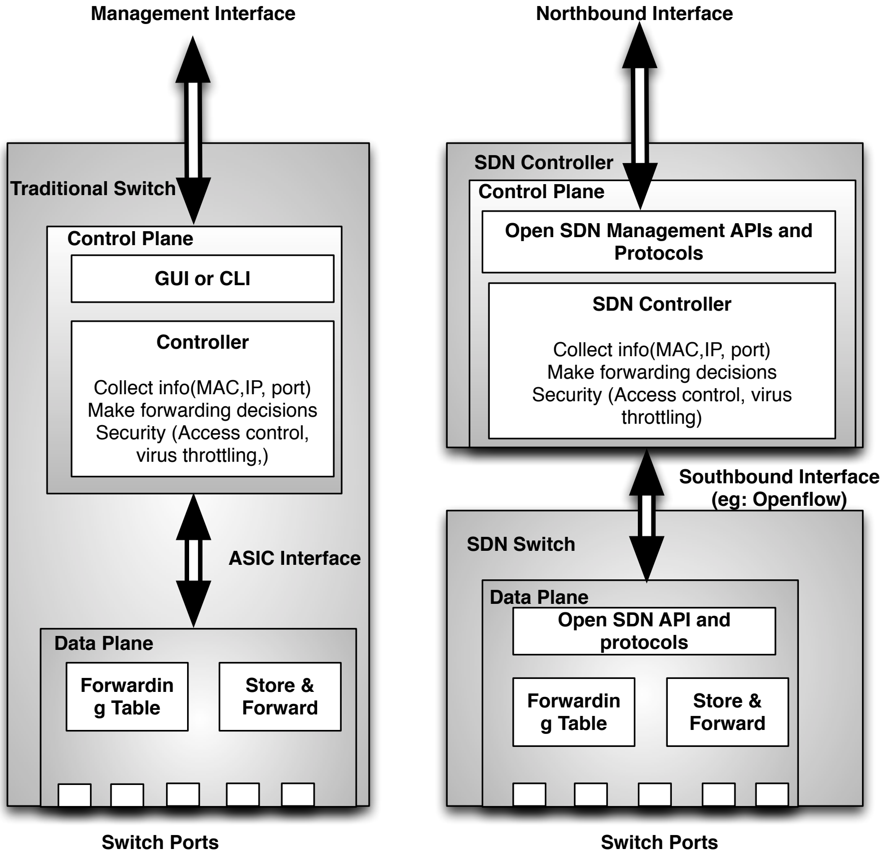


Figure 1.1: Architectural differences between regular and an OpenFlow switch.

a load balancer, a hub, or a firewall, all operating at line rate performance.

Currently, users are segregated from each other using VLAN, but this requires a reconfiguration of the network each time a VM is created or shut down. VLAN is a network of computers grouped together based on a common set of requirements even though the computers are physically on separate LANs. It is very difficult to configure each switch manually in a dynamic cloud environment, but by using SDN this can be automated by programming the controller via the Northbound interface API. With the use of CloudStack or OpenStack, VLAN user isolation is achieved, which can be managed more easily by the use of SDN.

The majority of SDN deployments are in a private LAN setting, where different dynamic policies, which require the network to be reconfigured, must be implemented. SDN is also attractive for larger networks

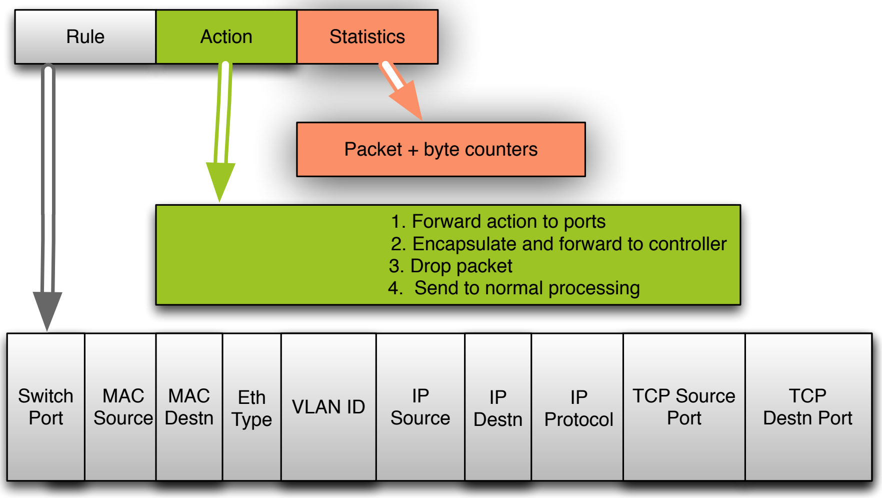


Figure 1.2: Flow table entry.

like data centers or campuses, where different network services such as firewalls and load balancing are needed. Thus using SDN, instead of focusing on a single device that could possibly suffer from packet processing speed restrictions, the rules are distributed among switches and are processed at line rate. SDN is also used where different network paths dedicated to latency or bandwidth are preferred.

Network Management Complexity: Having no SDN, network administrators use scripts to configure the network. Even with SDN, administrators must deal with it being error-prone, which is why new abstractions need to be created that will lead to a simpler system for fixing errors.

The development of OpenFlow permits for the demonstration of many potential benefits of SDN, and multiple vendors have started to offer commercial switches supporting the OpenFlow standard. Researchers have also made progress on SDN components, such as SDN controllers, switches, programming interfaces, verification and debugging tools; deploying SDN in data center networks and campus networks; routing; and traffic engineering. Despite the progress, many important questions regarding SDN still remain, such as longer-term issues around SDN's theoretical foundation; programmability and control logic; formal methods and protocol engineering; abstraction and view; network operating system, etc.

OpenFlow does not directly provide an interface for describing network policies, but do so only on the basis of packet header match fields and network actions [9]. Though it may be a very useful interface, it has limited value when attempting to enforce policies using only these network primitives.

Despite the flexibility that OpenFlow offers, it still requires that flows be installed manually, which is a time-consuming and error-prone process [10]. Many people have gone through the effort to abstract OpenFlow fields and actions so that programmers and network engineers may use the more familiar interfaces [11],[12]. However, none of these interfaces is specifically dedicated to providing mandatory access control.

1.2 Mandatory Access Control

An essential part of any secure system—particularly those that are multi-level security systems [13]—is mandatory access control: a system that determines whether a given subject may access a given object, both of which are given certain security classifications. Mandatory access control is especially difficult to enforce on a network, as it must usually be enforced manually at the endpoints, which are the machines themselves. Otherwise, the only options are expensive and custom systems, which often come with undesirable effects such as those observed when using unidirectional security gateways [14]: difficulty in reconfiguring the system as the network expands, and inflexibility when it comes to the diverse communication needs of the user base. There is demand for a framework that allows administrators to automatically create and enforce network and security policies [15],[16],[17]. With software defined networking (SDN), it is possible to use commercial, off-the-shelf hardware to enforce network-wide policies at each switch [18], and usage management incorporates mandatory access control.

1.3 Tradeoff Between Performance and Security

Despite the rising popularity of cloud computing, disengaging users from the hardware needs has long been criticized for being unable to provide a trustworthy solution to the problem of managing the trade-off between security and performance [5]. As the security requirements become more stringent (in military applications, for example) the performance of the application service decreases [19]. The potential and scope of cloud computing is bound by the challenge of ensuring organizations' and users' security and privacy while maintaining efficient and reliable systems.

This unfortunate dilemma of optimizing the trade-off between security and performance is an obvious but legitimate criticism of cloud computing architectures, and it lies at the cornerstone of this research. The key challenges facing cloud providers are how to balance and to optimize resources and security within a cloud system. Resource allocation optimization ensures maximum returns on the cloud infrastructure to the cloud provider. Security services provided by the cloud are critical parameters that determine whether or not the customer is confident in moving data to the cloud. However, these goals are often at odds with each other. In many cases, security is compromised in the quest for higher resource utilization, and sometimes efficient resource utilization must give way to changes that achieve better security.

An application may display different performance measures in a cloud computing environment, , such as response time, throughput, or latency. In previous works [20], an approach assuming that the cloud can be modeled as a Multi-Input-Multi-Output (MIMO) system is implemented to regulate performance in the cloud. From this perspective, the cloud is seen as a black box with inputs and outputs, where the outputs are the different performance measures of interest, while the inputs are values related to the amount of available virtual resources in the cloud e.g., CPU utilization and available memory in a Virtual Machine (VM) as well as the number of available VMs in a cluster. The authors assume that the hypervisor allows for fine-grained control of the parameters of the Virtual Resources in place. However, based on, in a public cloud service—specifically in IaaS such as EC2 from Amazon—the variation of the internal parameters of the VMs, and therefore the response of the system, are coarse-grained [21],[22]. This imposes several difficulties to applying methodologies such as Linear Quadratic Regulation as in the CPU problem [23], which is effective in regulating MIMO systems. Our framework uses Usage Management in conjunction with SDN to enhance security for the cloud. But an increase in security measures corresponds to a decline in performance.

This research provides a solution to the problem of provisioning resources for security and performance. The resources and security allocated to the VM are calculated based on our statistical learning model for optimization. In this model, like other resources, security is considered as a resource allocated to the VM. The quantity of security allocated to a given VM is measured in terms of the strength of the encryption algorithm applied to the sensitive data to be used within that VM. The greater the strength of the security algorithm, the greater the quantity of security resource allocated to the VM.

1.4 Outline of the Framework

The goal of this research effort is to apply Usage Management (UM), in conjunction with SDN, to control communication in a novel way: to provision and control cloud-based resources and ensure that all security and performance requirements are met. UM provides dynamic security by continuously monitoring the resource, taking into account any change in context and enforcing appropriate action to ensure security requirements are upheld. In our framework the Usage Management Mechanism (UMM) acts as a Network Function Virtualization unit (NFV), the NFV virtualizes network function previously executed by means of hardware. It is used when a user desires a one-way flow of information [24],[25]. For the action "read," it fetches the resource, and for the action "write," it writes to the resource. The OpenFlow mechanism becomes active when data must be transmitted to the physical or virtual machines connected to the SDN.

To understand when either mechanism is in use, let us imagine a scenario in which users of two machines want to communicate. User of Machine A wants to communicate with User of Machine B. Figure 1.3 shows the use cases explained below.

Scenario 1: Machine A is at a higher security level than Machine B. Action is "read." The policy generated is with action "read," and the UMM takes a copy of the resource from Machine B and sends it to Machine A, which is essentially a "read" operation.

Scenario 2: Machine A is at a lower security level than Machine B. Action is "write." The policy generated is with action "write", and the UMM pulls a copy of the resource from Machine A and also sends it to Machine B, which is essentially a "write" operation.

Scenario 3: Machine A is at the same security level as Machine B. Action is both "read" and "write."

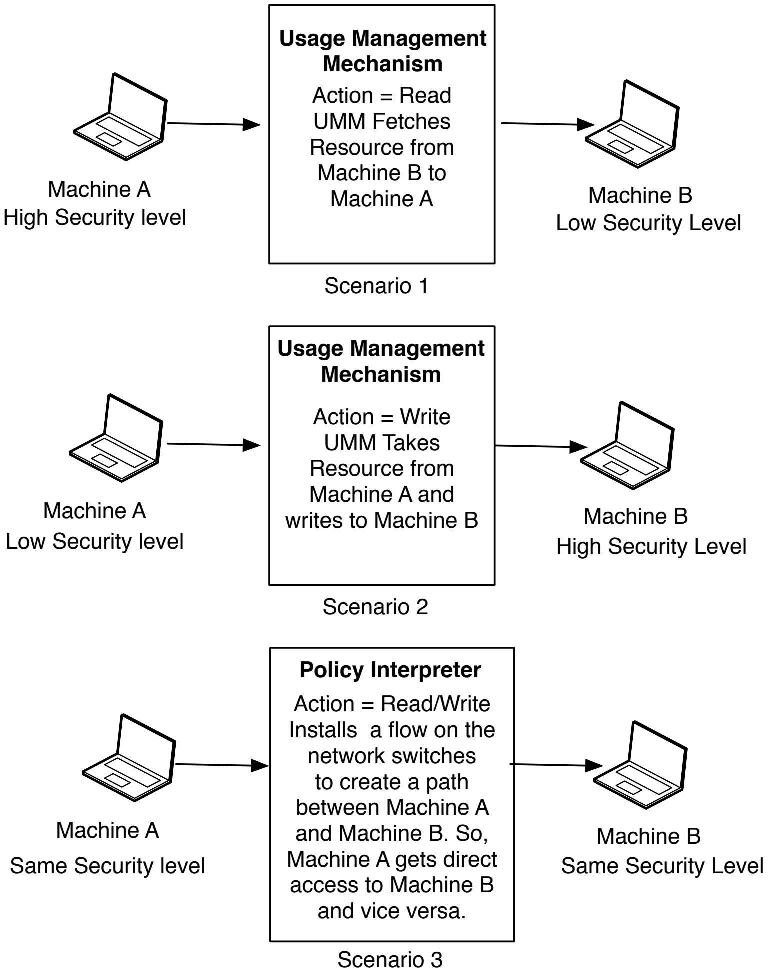

Usage Management Mechanism
Action = Read
UMM Fetches Resource from Machine B to Machine A
Machine A
High Security level
Machine B
Low Security Level
Scenario 1
Usage Management Mechanism
Action = Write
UMM Takes Resource from Machine A and writes to Machine B
Machine A
Low Security level
Machine B
High Security Level
Scenario 2
Policy Interpreter
Action = Read/Write
Installs a flow on the network switches to create a path between Machine A and Machine B. So, Machine A gets direct access to Machine B and vice versa.
Machine A
Same Security level
Machine B
Same Security Level
Scenario 3

Figure 1.3: Use cases for our developed framework.

In this case, we want to give Machine A direct access to Machine B, and vice-versa. The policy generated is with both actions—"read" and "write" in this case—the policy interpreter accepts the policies and converts them into network flows. These flows are then installed on the network switches that are a part of the OpenFlow network to allow communication between Machine A and Machine B. Finally, a network path is established between Machine A and B so they may access one other.

Our framework creates network policies that are based on the following entities: source, destination, and a policy model. These entities are entered by the network administrator through means of a simple web interface that stores them in a relational database. We implement a policy-based usage management system for software-defined networks and for multiple cloud environments. This system takes resources with specific attributes and provisions them according to usage policies. This approach automatically generates usage and network policies, and also enforces them.

In summary, we present a framework that allows for a more user-friendly abstraction, which means that the user doesn't have to worry about the implementation details. Our framework uses OpenFlow to install specified policies on the SDN controller, facilitates abstraction for reasoning over network flows, and provides asymmetric communication by means of an NFV. The framework also provides a single point of entry for specifying policies.

Subsequent chapters of this dissertation are described below:

Chapter 2: The immense popularity of clouds arises from the ease that a cloud infrastructure offers for collaboration and information sharing. To capitalize on this popularity, higher security and privacy enforcement measures are warranted. This would allow the cloud infrastructure to truly become the utopian computing environment that it is idealized to be. These problems have been discussed at length.

Chapter 3: From the scale and dynamic nature of the cloud stems a need for an efficient mechanism to control the network in cloud-based infrastructure. SDN provides fine control over the network, hence it can effectively fulfill the cloud network's needs. It transforms network management from traditional network configuration to network programming, but raises the question if SDN-based clouds provide more network security than traditional clouds. In other words, does SDN obviate the security problems in a virtualized cloud setting?

Chapter 4: This chapter gives a through description of the related work and background, and finally compares the currently existing contrasting technologies.

Chapter 5: Todays systems are cloud-based and virtualized. The systems, however, need to control the access to information. Also, since components in a transaction may not be co-located, a policy-based approach for data access and data transmission complying with confidentiality and integrity requirements. The mechanism offers secure usage control of sensitive data within sensitive VMs.

Chapter 6: Security and resource optimization are two of the most important concerns in cloud computing. This framework offers secure usage control of sensitive data within virtual machines that are dynamically instantiated while optimizing both security and the resources allocated to the VM.

Chapter 7: Since the topology of every network is likely to be unique, and there may be middle boxes, routers, and other devices that may make it difficult to enforce SDN policies [26]. Network applications are generally difficult to implement correctly [12]. The goal of this research is to demonstrate automatic generation and execution of usage and network policies. But in order to control how the information is transmitted over the network, this framework was built and dependent on the UM system data transmission governed by SDN policies.

Chapter 8: This chapter draws conclusions, and provides direction for further research.

Chapter 2

Cloud Computing Security

Cloud computing refers to the provision of services that entails hosting and maintenance of data, confidential or otherwise, on cloud servers. As a result, clients who store their confidential data on cloud servers are essentially handing it over to the cloud providers for safe keeping. The success of cloud computing is therefore heavily contingent on the clients' complete trust of the cloud providers. An individual's trust in a system dwindles if they are not given much control over it [27]. For instance, people trust the withdrawal of money from ATMs because the transaction ends with them receiving money. However, the use of ATMs for deposits is less trusted because people do not know what happens after inserting their money into the machine. In cloud computing, trust is dependent on whether the providers provide said service and whether they ensure confidentiality of their clients' data. In distributed processing, the user has no control over where the data is processed after uploading it to the system. The user may not be well-acquainted with legal rules of the region where the data is stored or processed.

The Trust Computer System Evaluation Criteria (TCSEC) is an example of an early trust that took effect the late 1970s and the 80s. TCSEC was a method of convincing customers of a system's accuracy and security. In cloud computing, customers trust providers based on whether they believe that provider's actions will not deviate from the exact promises and expectations. As a result, trust is based on the credibility and consistency of the provider. A system that does not give sufficient information regarding its capabilities is less trusted. Customers are not merely swayed by catchphrases such as "trust me" or "secure cloud." Instead, they look into the system's transparency in providing details such as where the data is stored and processed in the distributed environment.

The level of discomfort can also be eased using a control mechanism. Such a mechanism determines the physical location of the cloud machines where the data are stored and processed. Another factor upon which trust heavily depends is the cloud computing infrastructures model of deployment. A user will most likely have more faith in a private cloud because overall systems are controlled internally by the same organization. The vulnerability of community clouds is lower because a majority of its users are from the same community and/or organization. On the other hand, the users of public clouds are from different locations, which can reduce their trust. The fact that public clouds serve different types users from different locations makes them more vulnerable.

Service-Level Agreements (SLAs) are of paramount importance because in most of the cases, they provide the only way through which trust can be established. However, in some contexts of cloud computing, SLAs may not be helpful because the breach of data in some cases is irreparable and money cannot help in recovering the cost. Consequently, trust in the context of cloud computing is usually based on the prevention of failure instead of post-failure compensation [27]. The Security Assertion Markup Language (SAML) is a current claim-based access control and federated identity approach used in cloud computing as a security token service.

The prominence of cloud computing is based on an integration of several core technologies without which its services would be rendered unsuccessful. These core technologies include:

- Web Services and Applications: this technology forms the foundation of software-as-a-service (SaaS) as well as platform-as-a-service (PaaS). SaaS refers to the end users' access to web applications. On the other hand, PaaS facilitates easier development by exposing web services and combining them to form web applications. Through PaaS, the development of new applications is made easier because it entails use of pre-built services. Web interfaces are also used in the administration of IaaS (for example, in the management of different users' access control).
- Virtualization: this technology allows users to simultaneously operate numerous isolated virtual machines (VMs) using one single physical machine. It endows the users with high computing power

while maintaining the system's elasticity and the pay-as-you-go model. Through virtualization, the users are provided with pooled resources, and better utilization of the installed infrastructure is realized. Virtualized infrastructure is used in the development of SaaS, PaaS, and IaaS.

- Cryptography: in most of the cloud systems, cryptography is the only technology used to ensure the security of data on the cloud servers. All web services and applications, virtualization and cryptography technologies have vulnerabilities that are inherent or that emerge when the technologies are used in the context of cloud computing. Some of these vulnerabilities follow:

- VM Hopping: through VM hooping an attacker can use one virtual machine to access other VMs. In such an attack, the victim's configuration settings, resource usage and confidential data can be changed or deleted, thus compromising the availability, confidentiality, and integrity of the user's data. For such an attack to be executed, the attacker must know the VM IP address of the victim to gain access to the victim's physical machine. Since cloud computing entails the simultaneous running of numerous machines from the same physical machine, VM hopping can cause a severe security threat in cloud computing. As a result, VM hopping is regarded as one of the core vulnerabilities of cloud computing technologies. Since one machine can be used to operate numerous computers, this attack can be launched on one or more VMs. PaaS and IaaS can be particularly affected by VM hopping. Since SaaS is based on PaaS and IaaS, users' confidentiality and integrity of data can also be affected indirectly by VM hopping [28].

- VM Mobility: the virtual disc contents of a VM are usually saved as files on the physical machine. As a result, the VM can be transferred between different physical machines. Such an endeavor is referred to as VM mobility. Though VM mobility facilitates fast system deployment, it also exposes the system to risks such as vulnerable configuration spreading. If an attacker encapsulates a vulnerable configuration in his VM, the configuration is moved with the VM when it is being transferred to another physical server. When the transfer is complete, the attacker can launch a man-in-the-middle attack. Such an attack can cripple the guest operating system, thus leading to loss of confidential data. The fact that VM mobility brings flexibility to the overall system makes this issue hard to completely prevent.

- VM Diversity: the different types of operating systems make it hard to secure and maintain VMs. The deployment of these OSs can be done in seconds [29]. The process of maintaining and securing VMs is more complicated as a result of their diversity.

- VM Denial of Service (DoS): the virtualization technology allows users to simultaneously operate numerous isolated VMs using one single physical machine. As a result, all VMs share similar physical resources such as memory, CPU, and network bandwidth. In some instances, one VM may consume all resources, thus rendering the other VMs on the machine incapable of accessing the services. In order to avoid such an attack it is imperative that a resource allocation configuration be conducted before the assignment of resources to the VMs. In the context of cloud computing DoS can be prevented using SLAs, as they clearly define each customers configuration.

2.1 Essential Cloud Characteristic Vulnerabilities

According to the National Institute of Standard Technology (NIST), there are several rudimentary features of cloud computing that makes it vulnerable:

1. Internet Protocol: the delivery of cloud services is done via a network, which most of the time is the Internet. The standard protocols used in the Internet are usually not considered trustworthy. Consequently, the vulnerabilities posed by Internet protocol are pertinent to cloud computing.

2. UN-Authorized Access: on-demand self-service that does not require human intervention was one of the key features of cloud computing identified by NIST. The delivery of such a service calls for a web-based management interface that can be accessed anywhere in the world. This means that unauthorized users can find their way into the management interface. This poses a huge risk to cloud computing because the management interface is at a higher risk of unauthorized access than traditional systems.

3. Data Recovery: another major characteristic of cloud computing is resource pooling, wherein a resource is shared by different users at different times. After one user uses a resource, another user

can use the same resource at different time. If an attacker gains access to the physical machine used by a user, he/she can recover that user's data from the machine's storage devices.

2.2 Limitations in Known Security Techniques

- Session Riding/Hijacking: web applications/services form the basis of cloud computing. HTTP is the carriage protocol of these web applications and services. The design of HTTP makes it stateless as a result and there is a loss in the application's state in the numerous requests to the server. The management of HTTP's state is done using multiple techniques, among which is session management. The management of sessions can be accomplished through a myriad of techniques such as cookies, query strings, and state servers. However, at the end of the day, the management of sessions is vulnerable to riding and hijacking. Given that web applications/services form the basis of cloud computing, the cloud computing architecture is susceptible to session riding and hijacking. The security of data stored on cloud can be ensured using cryptography techniques.

In some instances cloud computing can directly impinge on security techniques in a way that renders them ineffective in the context of cloud computing. For example, the standard IP-based network zoning technique is no longer used in cloud computing. Network-based vulnerabilities are strictly proscribed by IaaS providers because it is hard to distinguish between an attacker's scan and friendly scans. Communication in virtualized environments occurs on real and virtual networks. A virtual network refers to a network connecting different VMs on the same physical machine.

Another noteworthy security control issue is poor management of keys. In cloud computing, there are a myriad of keys that need to be generated, stored, and managed. VMs are usually distributed on a geographical basis. As a result, they do not have a fixed physical hardware, and this complicates the incorporation of some hardware security modules (HSM).

Lastly, there are not security metrics given to the cloud computing users to help them keep track of their cloud services security status. No such security metrics have been developed for cloud computing. Until such security metrics are developed, the accountability and security of cloud computing will always be problematic.

2.3 Vulnerabilities in Cloud Offerings

The offerings provided by cloud computing are among the most state-of-the-art in the market. The vulnerabilities that pertain to these state-of-the-art offerings are referred to as cloud-specific vulnerabilities.

Weak authentication is yet another severe vulnerability in cloud computing. For user to access cloud computing services, they have to use web interfaces. The username-password authentication techniques used in these interfaces are not considered entirely secure because of some user-based problems such as weak passwords, not remembering the password, and the like. Cloud-specific vulnerabilities may also arise from the infrastructure and platform used. The infrastructure used is responsible for some of the basic services such as storage of data, resources computation and communication. Platforms of cloud computing facilitate the development of applications. They also provide a run-time environment for services that are developed using one of the supported languages. Below are the vulnerabilities that pertain to cloud computing platforms.

2.4 Security Risks Associated with Storage

The features of cloud computing that mostly increase the vulnerability of cloud data include resource pooling and elasticity. The storage devices used to store data from different users' are the same. When a user stops using the address space assigned to him/her, it is assigned to another user. If this other user is an attacker, he/she can recover the previous user's confidential data.

Both hard and soft media sanitization are also hard to achieve in the context of cloud computing. Data sanitization is conducted to prevent incidences of data remanence. Data remanence can be defined as the data footprint after the removal or deletion of the actual data from the storage media. Data remanence is experienced after a process of file deletion, no matter how insignificant it might be. Formatting the storage media serves as the most ideal process in the quest of media sanitization. Nevertheless, cloud computing is distinctive, as it does not permit formatting.

Another method of sanitization mostly used by organizations is hard sanitization, which entails physical destruction of the storage media. Hard sanitization is also not an option in cloud computing because the storage devices are shared among different users. Cryptography is most appropriate security solution that can be used in cloud computing. However, as previously mentioned, the security ability of cryptography is usually thwarted by poor key generation, storage and management among users.

2.5 Securing Communication

The elasticity and resource pooling features of cloud computing also lead to the sharing of some networking infrastructure among users. When network infrastructure resources such as dynamic host configuration protocol, domain name system, and internet protocol are shared, an attacker can use them to execute a cross-tenant attack. Such attacks are usually executed in IaaS environments. Network virtualization enables the use of real and virtual networks in the same physical environment. It is sometimes impossible to integrate network-based security implementations in a virtual network environment.

2.6 Auditing, Authentication, Authorization and Identity (AAAI)

In almost all of cloud computings services, AAAI are regarded as major requirements. They can sometimes be provided as third-party services, but more often than not they are part of the process of the primary services provided to the customers. Apart from the weak user authentication cloud computing challenge previously discussed here are some more cloud-specific challenges:

- Feeble Credential Reset Process: in the event that the user fails to remember his or her credentials in an attempt to log in to the interface, the procedure utilized in resetting the credentials of the user customized for cloud computing particularly because such details are managed by the providers of cloud computing.

- Denial of Account and Denial of Services: in authentication mechanisms that use usernames and passwords, if the user enters the wrong credentials many times, he/she gets locked out of the system. This can be corrected using some human interaction, such as using CAPTCHAs during the next authentication verification endeavor. If the desktop client applications being used are pre-configured to access the interface at remote locations, denial of services persists until there is some human interaction with the system.

- Authorization Checks: the authorization checks provided by web applications and services are usually inadequate, and this may help the attacker to deduce any subsequent changes that might help them get access to an unauthorized record. For instance, if the record is being shown to a customer (who happens to be an attacker) via ID in the query string, the attacker may be able to deduce the next possible accessible record. If there is a possibility that a pre-determined flow can be bypassed, the application of authorization checks should be done using individual services.

- Customizable Authorization: the authorization configuration platform provided in the cloud service interfaces should have a high level of customization. To achieve this, categories of different users should have different privileges at different times. The configuration panel provided by the management interface should be such that all users are strictly provided only what they require at a given time.

- Activity Logging and Monitory: there are currently no metrics in cloud computing that can aid the process of logging and keeping track of a user's activities. Log files help in recording every activity being conducted on the servers. However, filtering them out for a specific user, at a specific access region, and so on, can be very difficult. Auditing of the user's activities is of paramount importance, as it helps in the provision of standard mechanisms of logging and keeping track of a user's activities.

On-demand self-service that does not require human intervention is a key feature of cloud computing identified by NIST. The delivery of such a service calls for a web-based management interface that can be accessed anywhere in the world. The interface inherits all problems that pertain to the use of protocols in web application/services. If all control is kept at a single place, the attacker can breach the system and access this location, which would lead to an immense loss in that particular cloud computing system.

The offerings being provided by cloud computing have lead to its rapid growth in the market. In spite of this growth, there is a continuing concern among the customers about the security of cloud computing systems. Customers are not well-acquainted with the security being offered to them because cloud computing does not have any clearly defined security standards. The efficiency of most of the current cloud computing security controls needs to be improved because they are not appropriate. Adapting the security modules to an acceptable level requires a lot of interest and effort. In addition, the introduction of cloud computing has lead to the emergence of new security vulnerabilities that are specific to its environment. As a result, there is need to develop some new cloud-specific security modules.

Different cloud computing services require different levels of security. For instance, more security is required in the fields of telemedicine and e-commerce, unlike those of public information. Likewise, different cloud computing users require different levels of security. For instance, more security is required for voice conversations that pertain to business issues on the cloud, unlike in the case of voice conversations between friends on the cloud. Consequently, observing the highest level of security is not always appropriate. In addition, observing the highest level of security for all the services and users can be an expensive venture of the cloud computing provider.

Chapter 3

Software-Defined Networking

Software-defined networking transforms a network into a platform and translates its individual elements into programmable entities [30]. Traditional networks employ control decisions and packet switching inside the switch itself. SDN, on the other hand, separates these two operations: control decisions are performed inside the controller, while switching and forwarding decisions are done in the switch. SDN is a relatively new approach that abstracts network elements as programmable, thus enabling easy and scalable management of the network.

Usually, a computer network is composed of a router, switches and middle boxes, and complicated protocols tailored for it. Network administrators design different network policies to ensure proper functioning and to deal with various network problems. They manually configure these high-level network policies into low-level network commands with limited tools. Therefore, the performance management of the network is difficult and error-prone. This problem has caused difficulties in the evolution of the internet in terms of infrastructure and protocols. In this light, the concept of SDN provides a solution in its ability to make programmable networks. It simplifies network management, treating network resources in a manner similar to storage and other hardware.

In order to develop in-house scripts and software to configure the network without SDN, network administrators are forced to utilize non-standardized APIs. Network programs, however, should become more readable to a larger number of administrators with the adoption of standardized interfaces, which will assist network management code trustworthiness as well the debugging processes. The major reasons a network administrator that inherits a network decides to develop scripts and management tools from scratch include: back doors that would compromise security may have been left by a previous administrator; and poor documentation of the inherited configuration. To avoid forcing the users to deal with complex, low-level Southbound interfaces, it is expected that new abstractions will be generated. Applying software engineering and creating the correct abstractions is what has been advocated by a number of researchers. This will result in a simpler system, enabling the utilization of formal verification techniques and easier debugging[31] [32].

Logically centralized networks management, in which the controller defines data flow, is advocated by the SDN. Regardless of the fact that the global SDN view tactic raises obvious scalability concerns, based on the researchers beliefs, scalability is not going to be an issue because the messages are exchanged at a much lower rate by the control planes, while the data plane operates at line rate. Nevertheless, a well-defined model for federation and collaboration is required by the global view. The boundaries of what each network program user can do must be well-defined in the case where there is sharing of network infrastructure by many users. When dealing with logical central control, the creation of a hierarchical structure where limited functionality can be delegated by the SDN domain to its users is a possible evolution.

Often times, physically, there is traditional isolation of the management networks from regular network traffic. Given that it is only the administrators who have access to the management network, the infrastructure would be protected from attackers, though access to network devices needs to be configured. In this particular circumstance, security is independent of whether or not SDN is utilized, and depends on how well network administrators can operate the infrastructure.

Nevertheless, permitting direct interaction amid the network, the application, and the end users, in a way similar to how they interact with storage and compute resources, is the vision of SDN. The mechanism to coordinate the utilization of network resources and regulate access to programming interfaces, therefore, becomes essential. SDN is considered to be secluded from the data network. Although there is need for much research and development, some features can be seen in network hypervisors as well as in network operating systems.

In order to attain maximum seclusion among users, several techniques, including VLANs and firewall configurations, are utilized by cloud providers. In advanced CloudStack networking, for instance, each tenant is assigned an autonomous VLAN. A large number of secluded broadcast domains are permitted by VLAN technology to coexist in a LAN, and a particular VLAN header is processed by switches. VLAN's complex management can be made controllable by SDN.

Alternatively, utilizing SDN programming, tenant seclusion can be attained. Given that SDN controllers can regulate how packets flow via the data plane, they can implement firewall rules. How well SDN programs are written is what seclusion level depends upon. Cloud resource sharing is susceptible to service attacks denial by exhaustion, from a performance perspective. By placing an extremely high load on shared resources, an attacker may possibly prevent access to CPUs of other users. Correspondingly, a network can be congested by an attacker attacking the machine running the SDN regulator, or sending needless messages. On the other, hand SDN can group flows per port and modern switches handle line rates per port, and regular users are unharmed given that it is possible to quickly detect such patterns and seclude the traffic.

With regard to detection and response to attacks, SDN consists of two major advantages over traditional networks, which are as follows: Instead of depending on expensive intrusion detection systems, detection of attacks a can be made a distributed task amongst switches; and without the necessity of accessing and reconfiguring many heterogeneous pieces of hardware, the administrators are permitted by the SDN centralized management model to quickly seclude or block traffic patterns consistent with an attack. In addition, SDN can be utilized in controlling how traffic is directed to network monitoring devices[33]. It is in a highly dynamic cloud environment that quick response is majorly important. The focus of the traditional Intrusion Detection Systems (IDS) is on detecting suspicious activities. Moreover, it is restricted to simple actions like notifying to a system administrator or disabling a switch port. SDN has the potential to take difficult actions, such as altering suspicious activities' paths so as to seclude them from known, trusted communication. Much attention will be given to determining ways in which existing algorithms in SDN context and IDS mechanisms can be recast, and to taking full advantage of multiple points of action through the development of new algorithms.

Despite the fact that mechanisms to access network statistics data from routers and switches are available, the end users rarely get access to such data. The switches are designed in such a way that the switches match the flow and they process the flow based on the rules defined by a regulator. Built-in performance counters also exposed via programming interfaces are contained in the hardware utilized to match flows. Based on either fine-grain monitored network or statistics data, the counters can be utilized to match rules. Service-level agreements between users and providers can be made easier to manage by leveraging these features. Both the cloud users and providers can be able to verify how and when violation of a network-related SLA has occurred, given a reliable statistics data source.

Because of its potential to offer better cloud resource management, VM migration inter- and intra-clouds utilization has been investigated. When a VM shifts from one server to another, complex network programs or reconfigurations are required to keep similar VLAN configuration access and firewall policies unchanged. The increases in complexity are dependent on the distance of migration that is, in decreasing order: across WAN, across buildings, across server rooms, across racks and within a rack. As already exemplified by the Route Flow SDN switch-based architecture, there is need for further research in order to realize SDN based on WAN. Increased network interfaces and programs supporting VM migration will be developed as SDN evolves and gets more frequently deployed.

Only part of the entire network would be affected by failure to any component, particularly with the traditional network switches. Precisely, network operation is minimally affected by failure of a management server, which is utilized to communicate with routers and switches whenever there is need for change in the configuration, since it will only prevent new configurations to be disseminated. Although it could prevent accepting new users in a cloud, the system will continue to run. Catastrophic consequences—complete shutdown of the network in the worst case—might potentially result from the failure of an SDN controller.

At the network level, achieving confidentiality is very challenging. To start with, the network is always aware of the packets source and destination. What follows is that users would be required to trust the network devices in case the implementation of the encryption is done at the network level. Attaining true end-to-end data confidentiality is possible if it is only the destination and source parties who are aware of the data. Given that there exist several validated application-level protocol s that guarantee data confidentiality, SDN is unlikely to provide encrypted communication confidentiality. When processing sensitive data, applications could constantly depend on traditional approaches.

Potential security risks can result from the fact that SDN exposes new interfaces to control and operates the network. The entire network security can be easily compromised in case communications amid the data plane and control plane are not appropriately secured. There is need for careful design and implementation of SDN controllers, as well as their access control policies, given that a compromised controller can affect the entire network's security. Once SDN is available, attacks that are challenging or impractical for users may become possible as low-level network services and functions are exposed to more users. Software-defined networking regulators normally operate on traditional computers that are known to be abuse-prone. To maintain the security presently dependent on network configuration, it is essential to properly secure these points of entry.

Tools that can assist experts to study different features of OpenFlow security have been developed by the group. NOX-OpenFlow controller platform is extended by FORT-NOX implementing security kernel that facilitates OpenFlow rule insertion requests by applications[34]. A rule-conflict detection engine that, in collaboration with a role-based authorization mechanism, decides whether OpenFlow rule deletions or insertion should be recognized or not, is implemented by the FORT-NOX security policy enforcement kernel. FRESCO is simply a security application development outline for OpenFlow-based SDN. The scripting language of FRESCO makes it possible to use fewer lines of source code in writing security applications in comparison to the number of lines of source code utilized in writing OpenFlow applications from scratch, facilitating developers to majorly concentrate on security features of OpenFlow application.

Different researchers have also studied the tools utilized in the development of OpenFlow or OpenFlow-based security applications. Network slicing is utilized by FlowVisor to generate multiple logical networks, and assesses OpenFlow conflicts amid logical networks[35]. The need for fine-grained rules that can be altered in response to network monitoring are recognized by the Resonance architecture, which also implements a security system that tracks diverse states of each host to accordingly apply security rules. Network programming language abstractions development that will assist in guaranteeing the consistency of policies in SDN has been proposed by some researchers. Moreover, the researchers have conducted research on the ways that existing security mechanisms are recast by the use of the SDN paradigm. The two services offered by Amazon Elastic Computer Cloud (EC2) are security group and elastic IP[36]. Where security groups are EC2 instances groups to which firewall rules are assigned by the users, elastic IP refers to a static IPv4 address which Amazon leases to certain cloud users that can be mapped programmatically to an instance (a VM).

The utilization of the Big Data Security is an interesting technique to uncovering and mitigating cyber-threats. Treating the complete organization network traffic as Big Data and utilizing Big Data mechanisms to implement security solutions is the major idea here. A potential solution to this is Security Intelligence and Analytics (SIA), as introduced by Piper[37]. Besides having the ability to capture every flow and packet that traverses a network, SIA can possibly sense threats that would be impossible detect with traditional solutions. There is an expectation that the functionality and performance of Big Data security solutions can further be improved by SDN.

The idea of a network instruction set may be conceptualized as SDN shifts to a notion of programming the network from network device configuration. Much attention has been directed by the OpenFlow specification to this particular layer, defining how the data plane and control plane interact with each other. Just like computer architecture, whereby there is direct access of the hardware by the stand-alone application minus operating system control, it is probable to utilize the network instructions directly to implement stand-alone SDN controllers. These stand-alone SDN controllers are currently the majority of OpenFlow controllers that are available, and are adequate for small deployments. A number of SDN features can be studied even with this simple setup: how to correctly seclude the control massages from the data plane; the ways in which the controllers should be protected against external attacks; the number of switches that can be handled by a single controller; which is the right set of network instructions that is to be exposed; the ways in which a distributed controller is to be implement; (among other features).

It is likely that, in a cloud environment, several controllers will be required to accommodate varied and often conflicting wants among system administrators, providers, as well as other end users. Therefore, a Network Operating System (NOS) that can coordinate several applications and find solution to potential conflict is required[38]. Security studies should therefore direct much attention on the NOS layers as follows: how well is each application secluded from each other? Are there vulnerabilities exposed by a function? What is the suitable set of interfaces exported by a NOS? The obtained lessons from computer operating system development years should be leveraged.

In addition, maximum flexibility in network programming would be enabled by the network hypervisors that have the ability to coordinate the action of a number of network operating systems. Just as in computer systems, each layer requires security consideration. A step toward the vision of networked sandboxes offering missing network virtualization services to clouds, campus-based networking test beds and fully virtualized datacenters is SDN. As an upcoming technology, there is need for much research and development to have a clear understanding of its security implications. Creating an analogy with machine virtualization, as required by the cloud middleware, users and providers, virtual networking services can be developed by the use of a layered architecture. There is need for data plane to be able to securely accept and execute commands form the control plane.

In making open specifications for the control and data plane interface, with less attention to security and more on flexibility, considerable effort has been spent. Based on the argument of a number of individuals, this particular interface needs to be physically secluded and secured, and accessed only by trusted hypervisor, network operating system or controller. Sophisticated authentication and access control are required in the control plane. There already exists a user base, well-defined mapping amid users, and resources in the deployed clouds. For instance, there is a clear mapping for a given user running VMs in an IaaS cloud. There will be need for the network hypervisors to restrict the actions of a controller to a specific set of VMs. Whenever a solution to cloud collaboration is to be leveraged and multiple clouds are involved, this particular task becomes more complex[39][40].

A large number of interfaces with potential security vulnerabilities are exposed since SDN allows for the network to be programmed. Considerable changes to cloud network management security are not probable in the case where SDN is simply utilized as a better technology to operate networks and there is physical restriction of the interfaces to system and network administrators. Network management security will remain the same, having similar threats and vulnerabilities in the case an assumption that SDN programming is properly done, is made. At least in the initial SDN deployment phases, a number of the deployments will follow the administrators-only network management model. There will be need to securely expose SDN programmability to a large number of users as SDN matures. Moreover, there will be incorporation of the increasingly complex security into SDN design.

With or without, SDN operation and configuration of networks is a difficult and error-prone activity. Given that a larger community is to be involved in interacting with the network control plane promoted by SDN, there will continue to exist unintended security vulnerabilities that may increase in certain scenarios. The flexibility in influencing low-level elements provided by SDN will at the same time facilitate the development of new ways of improving security that were impossible before. Security is highly dependent on the ways in which network programs are implemented.

In coordinating network programming responsibilities, a layered approach similar to computer systems can be utilized. In the data plane, the programming interfaces can be deliberated as a network instruction set, having network hypervisors coordinating several network operating systems. This in turn provides services to several applications. There will be need to appropriately secure each layer. To prevent applications from interfering with each other, the access to network services need to be well-coordinated and controlled. Consequently, there will be need for complex control of the users, and interoperation and federation encounters the same as ones faced by clouds emerge.

Chapter 4

Problem Description and Previous Approaches

4.1 Software Defined Networking

One of the most widely used SDN protocols is OpenFlow [41]. An OpenFlow-enabled switch implements a flow table, and the OpenFlow protocol is used to access that flow table. The communication between the southbound side of the controller and the switch is standardized by OpenFlow protocol, relieving the users from intricate details. The API to the northbound side of the controller is used by network administrators to program the network.

The precursor to OpenFlow is Ethane , which was motivated by the difficulty in maintaining the security needs of a large network [15]. Such a network has coarse-grained access control and is vulnerable to IP spoofing attacks. Ethane uses a centralized controller architecture for enterprise networks and security achieved via a global network policy. It establishes secure network access between switches and controllers; the switch flow tables are updated by the controller, and the packets are strongly bound to the original sender. IP addresses are assigned by the controller and source devices need to be registered to the network, users bind to the host via authentication , and controllers get their rules from the application. The policies use high-level names, and are called flow-based security language (FSL).

Like Ethane, Resonance also controls network traffic using policies that a controller implements via programmable switches [42]. Using this approach, the authors were able to build a dynamic access control framework that combines the controller with the monitoring sub-system. Resonance focuses on granularities of policies and segregates compromised hosts, whereas traditional compromised node and host segregation is often done using VLAN.

One key element of Resonance is a central authentication system in the controller. Upon successful client authentication to their portal, the controller moves the host to authenticated state and updates flow table entries in the switches and initiates a scan of the host. Based on this scan, the client is either raised to operation state or quarantined state. There are four progress states in resonance, namely: quarantined, registration, authenticated, and operation. This is similar to how policies are used in our work.

FRESCO is a security framework to add security policy to SDN, specifically in conjunction with NOX controller [43]. It has its own scripting language and creates flows with NOX on that basis. It also has its own database storage language. FRESCO can load modules, and has a specific security enforcement kernel that sits inside NOX.

A few problems in network management are enabling frequent changes to network conditions and state: providing support for network configuration in a high-level language, and providing better visibility and control over tasks for performing network diagnostics and troubleshooting. The authors of [16] present a solution where they create network policies in a high-level policy language and of the source of the problem. They take into account time, data usage, status, and flows for the policy generation. They present a four-state architecture which resembles the one used in Resonance to implement transitions in a campus network.

FORT-NOX is a security enforcement kernel for OpenFlow Networks [38]. The kernel is designed for role-based authentication, rule-based authorization, conflict detection and resolution, and security directive translation. One interesting thing about this is that it is designed to help prevent using OpenFlow switches to do creative hacks, such as packet header modification to bypass a firewall (dynamic flow tunneling).

Frenetic provides programmers with a collection of abstractions for writing controller programs for SDN [44]. The work focuses on three stages of network management: monitoring the network state, configuring new policies, and reconfiguring the network.

Procera is a control architecture for SDN based on functional reactive programming [45]. The authors mention that it is possible to fully express network policies in Procera, but the language has a steep learning curve, hence a specialization is required. In fact, the authors themselves mention that there is a need for a simple configurable interface to simplify the task.

In OpenSec, the authors are interested in simplifying how network security policies are implemented in a campus setting and how their framework responds to alerts [46]. They create and implement network policies in simple, human-readable language. They describe a flow in terms of OpenFlow matching fields and define which security services must be applied to that flow and specify security levels that define how OpenSec would react to malicious traffic, if detected.

SIMPLE is a policy manager for sending packets through various middleboxes [47]. SIMPLE translates middlebox-specific policies into rules that can be installed on SDN switches while ensuring that the load across middleboxes is balanced. In order to address the challenge of policy composition, they introduce processing tags. These processing tags allow packets to take network loops in order to fulfill policies. These tags use the extra bits in an IP header and can be abstracted as a state machine. This keeps the OpenFlow policies from being static and avoids a packet getting stuck in a loop. The work also introduces ‘resource management’ and ‘flow correlation’. Middleboxes, such as a proxy, may modify the packet header and in order to make a correct flow, the packet itself must be inspected. ’Flow correlation’ inspects the payload of a packet in order to correlate flows.

FlowTags uses modified middleboxes to create tags to help the flow pass along [26]. It works similar to SIMPLE, except that the FlowTags are actually changed at the middleboxes and not the switches.

4.2 Usage Management (UM)

UM supervises the usage of resources (and data) across and within computing environments [48]. Once users have legal access, usage management provides the owners of information with an assured ability to regulate the fashion in which users can use these data. Early work focused on the protection and distribution of copyrighted material in electronic form, and has motivated the developments in the conceptualization of usage management. Today's developed computing environment has excellent network capabilities: they not only support intensive data transfers, but also have high transfer rates resulting in short transfer times. Users can also quickly download data to high-speed and high-density media like thumb drives and portable disk drives. These abilities are truly convenient, discounting the concerns regarding ownership and secrecy of the data. However, it becomes a serious problem if the owners are not willing to disperse their data freely, which very often is the case. Hence, there exists a need for automatic control of the use of digital data, which is one of the motivations for research described here.

4.2.1 $\mathrm{UCON}_{\mathrm{ABC}}$

Moving beyond traditional access control systems, which utilize server-side mechanisms and an access matrix to make access decisions, a conceptual framework was introduced by the $\mathrm{UCON}_{\mathrm{ABC}}$ usage control model [49]. Through this work, models were introduced that integrate Authorizations (A), Obligations (B), and Conditions (C), which form the base of UM systems. An important addition proposed by them is that resources, for which access has been granted, must be incessantly controlled. They mentioned in their work that a client-resident trusted computing base and a reference monitor are required for enforcement if control is to be provided to the client. Though their work focused on an operational model, they failed to deal with that requirement.

$\mathrm{UCON}_{\mathrm{ABC}}$ uses Digital Rights Management (DRM). With it, the information provider has some ability to restrict the activity of the user. The set of features of the operational models and their structure is illustrated in Figure 4.1.This rich set of features can be measured and used to make access decisions. The subjects (S), subject attributes (SA), objects (O), and object attributes (OA), are considered in this model.

Rights (R) are the privileges that a subject can apply on an object and can be included in consumer rights (CR) and provider rights (PR). Authorizations are the functional predicates that have to be measured for usage decisions. Obligations are also functional predicates which validate necessary things to be performed by a subject before or at the time of usage. Conditions are decision factors that are environmental or system-oriented.

The $\mathrm{UCON}_{\mathrm{preA0}}$ model can be used for representing the action of the enforcement mechanism on the basis of data flow. L is described as a framework of security labels with the dominance relation, $\leq$, and functions: $S \rightarrow L$, maxClearance: $S \rightarrow L$, and classification: $O \rightarrow L$. The lattice or framework is used at first for making decisions on whether a subject can be allowed to access an object on the basis of the clearance level and conditions of the subject and the objects classification. After that, the information that will be used by an enforcement mechanism for controlling data flows can be provided by the lattice. In this, the following function will be used: $\mathrm{allowed}(o_1, o_2, \mathrm{write}) \Rightarrow \mathrm{classification}(o_1) \leq \mathrm{classification}(o_2)$.

4.2.2 An Interoperable Usage Management Framework

A framework for UM in open, distributed environments that focuses on interoperability was proposed by Jamkhedkar et. al[50]. Their system is a combination of the access control and usage control functions of the $\mathrm{UCON}_{\mathrm{ABC}}$ system and DRM. Content management, license management, simple access control and specification of usage rules are included in DRM. They make an important observation, which is that UM policies should be strongly attached to a data resource as resources will be typically moved to locations not known a priori. They also recognize that each computing environment should be able to interpret a policy language and apply the policy. In this framework, policies are defined in a license. Also, the interpretation and application of these licenses are done within a computation environment. The operation of the system is done in two stages: a set-up stage, and a working stage. In the set-up stage, the computational environment is set up and license is generated. In the working stage, license is interpreted (as required) and applied in operational environment, following which policies mentioned in the license are applied to the computational environment.

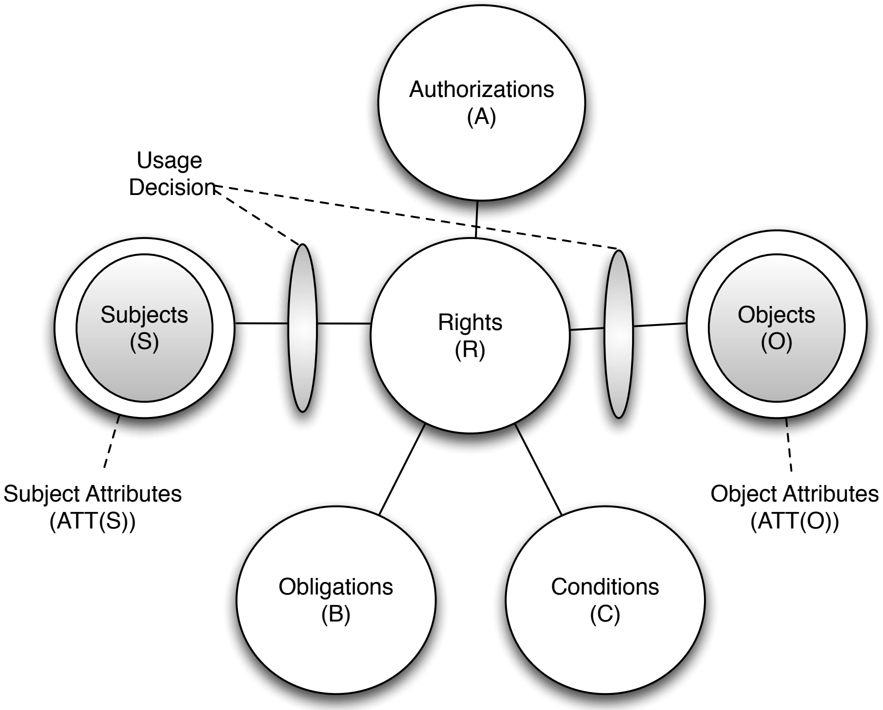


Figure 4.1: Components of UCON$_{\mathrm{ABC}}$ model.

4.2.3 Usage Management in Cloud Computing

A concept based on their earlier design of an open, interoperable framework was later presented by Jamkhedkar et. al[51]. They consider an operational environment, consisting of the systems in which different cloud computing providers can operate. They state that a common cloud ontology is required so that definite policies can be made and can be applied for the different set of systems. They recommend a setup stage and a working stage. Context information from each service is used by the set up stage and using this information, data set policies are produced. This usage is presented in the framework of the common cloud ontology. In working stage, policy management, interpretation and validation are done. A Usage Management Cloud Service is suggested by them. This service interacts with individual cloud computing systems to interpret if the operations in the given context given are allowed.

4.2.4 Security Policy

A security policy can be completely described by the following :

Subjects: The agents that interact with the system. Can be either individuals or roles; ranks that groups of individuals might possess in any organization.

Objects: The resources (computational or informational) that the security policy is required to manage and protect.

Actions: The operations that the subjects may be allowed or disallowed to do with respect to objects.

Permissions: The association between the subjects, objects and the actions exhibiting which actions are permitted or prohibited with regard to an object and a subject.

Protections: The rules that are included in the policy to safeguard certain security aspects such as confidentiality, integrity and availability.

4.2.5 Security Models

A security model is an abstraction that provides a policy specification language to administrators. Usually security models specify groupings of access or modification rights that users can possess based on their position in the organizational hierarchy. For example, in a military setting the documents can be categorized as secret, top secret, classified or unclassified. The models are:

Discretionary Access Control: Follows a methodology where the users have the right to determine the permissions governing access to their files.

Mandatory Access Control: Follows a methodology where the users do not have the right to determine the permissions to their files – rather, the administrator reserves this right.

Role-Based Access Control: Access is provided based on the position an individual fills in an organization. It simplifies the work for the system administrator of the organization, though the issue with this access control model is that if access to other actions that aren't permitted then another method is required since the only way in this method is to provide access to a role, exposing the possibility of unauthorized access.

Rule-Based Access Control: Users are dynamically assigned roles based on criteria defined by the custodian or system administrator.

Trust Management: A framework for defining security policy usually specified in terms of a programming language and typically combined with an enforcement mechanism for these policies. It consists of policy language and compliance checker. The rules are specified in the policy language and enforced by the compliance checker.

In trust management systems; the rules describe

Actions: The operations that explain the security-related ramification.

Principals: The entities that can perform the actions on the system.

Policies: The rules that define which principals are permitted to perform what actions.

Credentials: The digitally signed documents that associate the principals to permissible actions, including the right to allow them to delegate privileges to other principals.

4.2.6 Access Control Models

Bell-La Padula Model

Bell-La Padula model is a multilevel security mandatory access control model providing confidentiality [52]. It has been traditionally used in military organizations for document classification. Each user and each document is assigned a strict level of linearly ordered security. Thus, only the user with an access that corresponds to the security level of the document can view it. Let us say an object x is assigned to a security level $L(x)$; likewise, each user u is assigned to security level $L(u)$. The user's access to the object is defined by the following two rules. Simple security property: A user can read an object if and only if $L(x) \leq L(u)$. It is also called ’no read up’ as it prohibits users from viewing objects with higher security than themselves. Star property: A user can write an object if and only if $L(u) \leq L(x)$. It is also called ‘no write down’ because it prohibits information flow to users at lower security levels.

BIBA Model

This is a similar model to Bell-La Padula, but it addresses integrity rather than confidentiality [52]. Similar to the Bell-La Padula model, the subjects and objects are assigned partial ordering. The model takes integrity levels into account that define the degrees of trustworthiness for objects and users, rather than different levels for establishing confidentiality. The access rules definitions are opposite of the ones in Bell-La Padula model. It doesnt allow writing to upper levels and reading from lower levels. Let us say an object x is assigned to a integrity level $L(x)$; likewise, each user u is assigned to integrity level $L(u)$. The user's access to the object is defined by the following two rules: A user can read an object if and only if $L(u) \leq L(x)$. A user can write to an object if and only if $L(x) \leq L(u)$. Therefore BIBA model defines a top-down approach – that is, the information can only flow down from higher integrity levels to lower integrity levels.

Another model that is similar to the BIBA model is the Low Watermark model. In this model the users with higher integrity levels can read objects with lower integrity levels, although after this reading the integrity level of the user is downgraded to the level of the object that he read. Role-based access control model usually defines rules for access to a set of resources, such as documents in accordance with the hierarchical role of the user in the organization.

Papers like Ethane and Resonance describe the use of policy languages to create policies. The problem is that the network administrators have to learn the language in order to create network policies that can be implemented on the network directly. In my framework the security policies are generated and implemented automatically. A source, an object, a security policy model, and an action are the only requirements for the creation and enforcement of policies. Any security model that can be expressed as a logic block can be used in this framework.

4.2.7 Contrasting Technologies

Frenetic is a programming language made for OpenFlow-based networks [44]. While it may be a simpler means of programming for network administrators, our system generates policies automatically, and is thus more accessible to users. Like Frenetic, Procera, which is a functional reactive programming framework, also simplifies the response to network events, but does so by means of high-level policies created by the user [45].

CloudWatcher, which is a security monitoring framework for the cloud, generates security policies to specify a flow and describe the security services to be applied to it [33]. If network packets match those policies, CloudWatcher then locates security systems for processing, whereas our framework applies the policies it generates automatically. FRESCO, which is an OpenFlow-based security framework that allows users to define security policies using security modules, performs the security processing in the controller [43].Our framework, in contrast, uses externally independent units like the NFV and policy generator and interpreter module to send information to the controller for processing.

The already-existing system most similar to our own is OpenSec, which also automates the implementation of security policies [46].OpenSec creates network security policies that, when they detect malicious traffic, send it to different processing units that provide security services like encryption and denial of service attack detection. Our system does not recognize those threats, but uses a security model like Bell-LaPadula and enforces mandatory access control on the network by means of an NFV, which provides asymmetric communication and abstraction for reasoning over network flows.

Chapter 5

Policy-Based Security Provisioning in the Cloud

5.1 Introduction to Usage Management

Usage management (UM) is the usage of resources (and data) across and within computing environments, incorporating characteristics of both traditional access control and digital rights management [48].

A conceptual UM system is described in Figure 5.1 for a cloud computing environment. The UM system determines if a user can be granted access to a resource, such as a data file. In a cloud computing environment, once a user is granted access to a resource, there is a need to control not only how the resource can be used, but also where. Our primary goal is to develop and implement a secure, robust, and inter-operable attribute-based usage management system for data transactions in cloud computing. This system will merge usage management systems with modern cloud computing technologies. Given a data resource with an associated sensitivity characteristic, one part of the proposed cloud UM process will be to determine on which type of cloud computing system the resource will be made available.

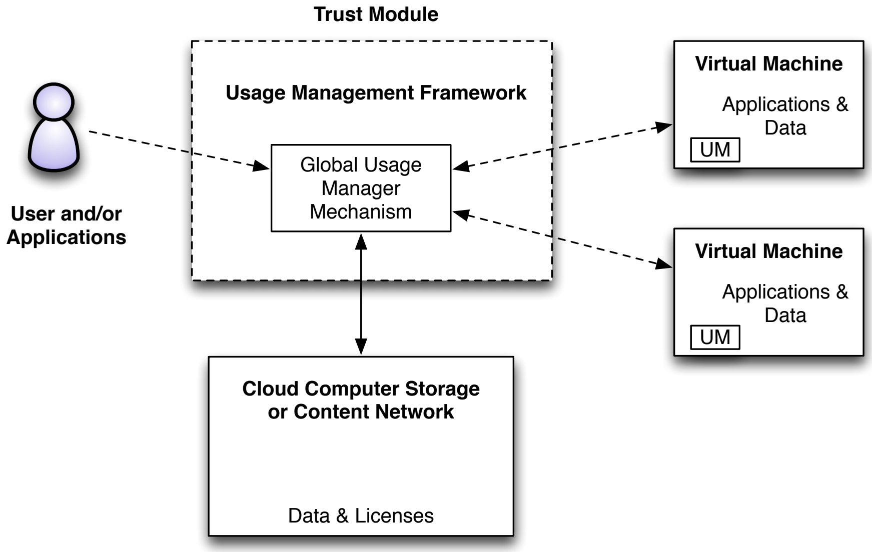


Figure 5.1: Hierarchical UM operation concept

5.2 Access Control and Beyond Access Control

In order to be effective, UM systems must be flexible enough to be scalable and inter-operable enough to provide services across different computational environments. In an implementation of UM, the first action provided by the system is access control. A user logs into a system with credentials, and with context that is provided by the user and/or determined automatically, and the users identity, information is compared with a resources policies to determine if that user is qualified to use that particular resource. The next role of the UM is to ensure that data are used in an environment that complies with security policies that are either specified in an associated license, or are applicable to a security category for unclassified data. For instance, a user might specify at which sensitivity level or classification level he or she wishes to operate, and UM checks the operating system and communication channels, i.e. the context of the user, before providing access to any resources. Once the user gets access to the resource, the UM system monitors how the resource is being used based upon the policy requirements. Based on the examination, UM commands a cloud computing system to instantiate a VM and load an image that contains the necessary security mechanisms. Then it transfers the actual data to the VM. In this paper we use VM images as the controlled resources, where each image has a set of policies associated with it that describe the circumstances under

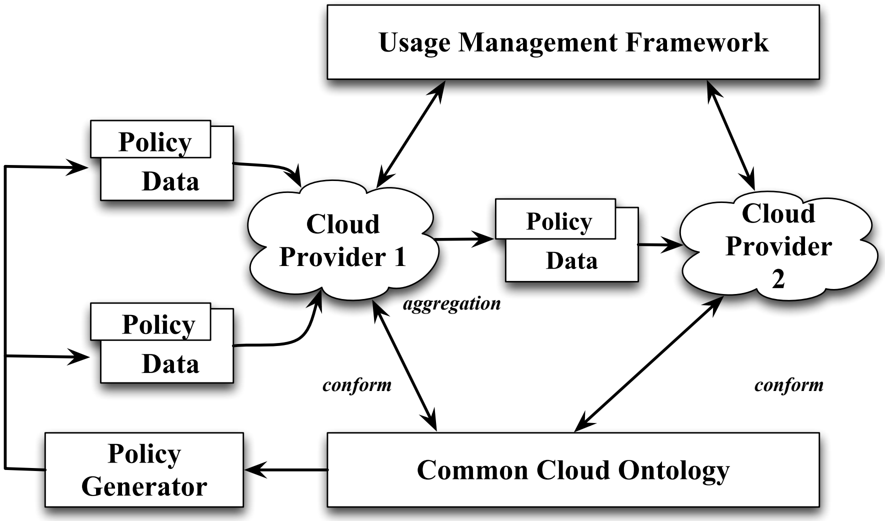


Figure 5.2: Usage Management in a Multi-Cloud Environment.

which that image can be used.

5.3 Usage Management in the Cloud

UM is important in the Cloud domain because it provides security by ensuring that cloud resources meet the security needs of any sensitive or non-sensitive data as defined by the corresponding policies [51]. Cloud resources can be provisioned according to the policy set—therefore the provisioned cloud resources are guaranteed to meet the security needs of the resource. UM enables Cloud Bursting: due to a computational need in the private cloud, if more Cloud resources are required, the UM system helps in moving data or even a job to a more capable public cloud system while also ensuring that the public cloud meets any security needs. This is done by evaluating the public cloud context (environment and multi-security level) with the policy set associated with given resources. The same can be done when moving from a public to a private cloud [53].

5.3.1 Usage Management System for Cloud Computing

In order to provide an assured information sharing capability in a cloud computing environment, a UM system capable of interacting with, and controlling, different provider systems is required. The high-level conceptual view is shown in Figure 5.2. Each system will likely be slightly different, so a common framework, or ontology, is required in order to use common policy semantics so that the policies can be applied consistently in each system. A policy generator will be required in order to generate the licenses necessary for each data set, and the policies will use the common ontology as a basis.

Based on the policies that are specified in a license for a particular resource, the UM first grants access to the resource. Then, based on the policies, the UM instantiates a VM in an appropriate cloud system and copies the data to the VM for processing. Once a data set has been transferred to a VM, without additional controls, the user and associated applications can use the data with no restrictions. In order to ensure that all resources are used in accordance with the policies, UM capability is also required within the VM for control.

5.3.2 System Architecture/Model

For a complete UM system in a cloud computing environment, a hierarchical design is required. Our proposed system implementation includes the following key elements:

Usage Management Mechanism: This component acts as a central controller that manages communication between all components of the usage management framework and with external services, such as storage and content networks. The usage management mechanism provides an interface for the user. Different applications can use the common interface provided by the usage management mechanism.

Trust Module: This module includes a capability to dynamically update the trust values of the cloud resources and update the policies within the licenses. The global Usage Management mechanism in the trust module will then force actions in response to changes, as necessary.

Virtual Machine Usage Management Mechanism: As mentioned previously, UM within each

VM is necessary to ensure that users and applications use data and other resources only in ways that are allowed by the policies specified within the licenses.

5.3.3 Policy

These policy statements are contained in a document, which is often called a license. For this design, the policies are expressed in XML, a format that can be read and interpreted by a computer executing UM control software. An example of the policy or license file is shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<license id="secret">
 <permissions>
   <restricted-activity num="1" activity="view">
     <entity-restrictions type="Subject">
       <restriction property="SecurityClassification"
       function=">=">Secret</restriction>
     </entity-restrictions>
   </restricted-activity>
   <restricted-activity num="1" activity="move_to_instance">
     <entity-restrictions type="Environment">
       <restriction property="SecurityClassification"
       function=">=">Secret</restriction>
       <restriction property="VMPlatform"
       function="==">Ubuntu</restriction>
       <restriction property="VMType"
       function=">=">Micro</restriction>
     </entity-restrictions>
   </restricted-activity>
 </permissions>
</license>
<?xml version="1.0" encoding="utf-8"?>
```

This policy file defines the resource's security classification as Secret. The policy states that the corresponding resource can be used in the environment where the VM has security classification of Secret, the VM Platform Ubuntu, and Type is Micro instance. Our framework supports dynamic interpretation of these policies in order to ensure policy enforcement in different cloud domains.

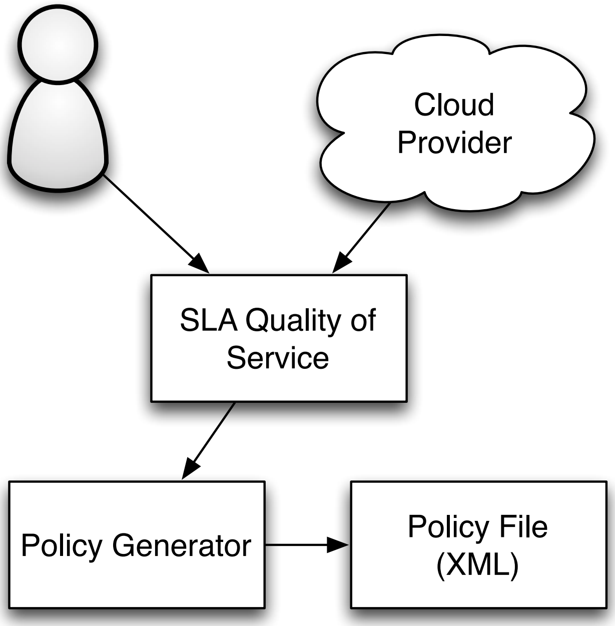


Figure 5.3: Policy Generation

These policies are the result of the SLA metrics discussed between the user and the cloud provider as shown is Figure 5.3. The policy generator uses this information and creates the machine readable polices using a common cloud ontology.

5.3.4 Context

These policies are interpreted within a particular context. The context is a representation of various entities and their relationship within the cloud environment. The context contains information about the user and the environment. User information could be, for example, his Security Classification, which determines whether he can operate at a Secret or a Top Secret level. The context also contains information about the VM environment whether it is a secure VM with classification as Secret. Additionally the context also contains information about the VM (Micro, Small, Medium, Large). The context and policies are generated from a common cloud ontology shared between different cloud systems so that the policies are inter-operated, and hence can be interpreted in different domains.

5.3.5 Implementation

UM Framework

UM Framework controls access to and use of information in a cloud computing environment. In a cloud domain, a user presents his credentials and context. The context information and requested operation information along with the policy is transferred to the UM Framework. The UM Framework decides whether the requested operation is permitted. Based on this query, the user gets to see a list of resources that he has access to. The user selects a resource and the UM Framework instantiates a VM that meets the security requirements and provisions the VM with the data. The UM is also responsible for the usage of this resource throughout the lifetime of the resource within the VM.

Architecture

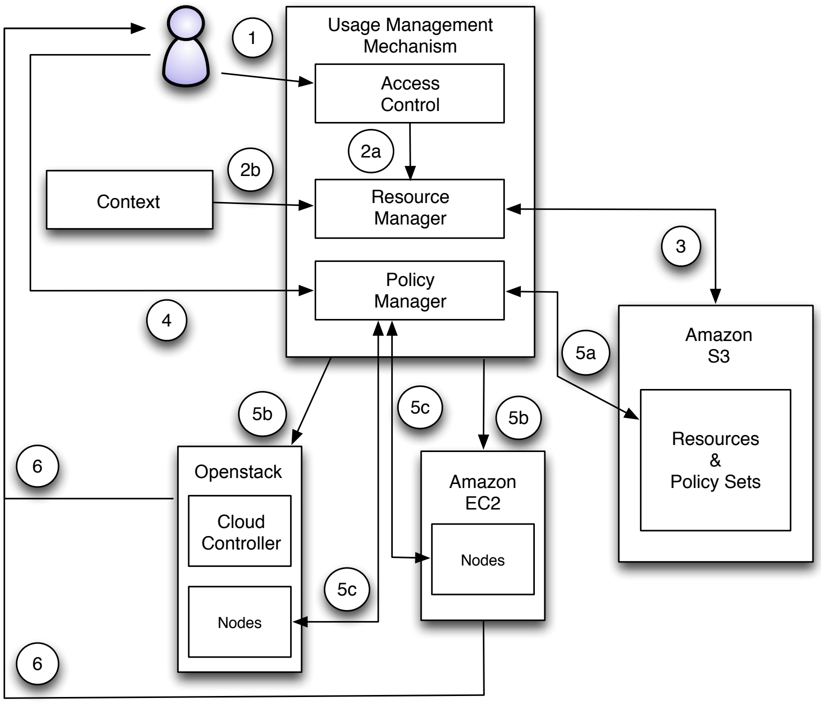
In this section we describe how UM is implemented within the cloud domain and show a technique to enforce UM policies within a VM, thereby successfully demonstrating how UM monitors and controls the usage of resource after the user has been granted access.

Setup

The architecture contains a Central UM framework that manages data in Amazon S3 (cloud repository) and cloud resources (VMs) on Amazon EC2 (public cloud) and OpenStack (private cloud). The framework can provision cloud VMs based on policies and also ensures that resources can only be moved to VMs that meet security requirements, and can redact resources when the cloud resources no longer meets the security requirements (i.e. context changes).

Amazon S3 holds the data and the respective policies in the form of buckets. The meta-data option of the resource links the resource with the policy file. Amazon EC2 and OpenStack provide cloud resources which have a context associated with them. A user context describes the information about the user . A local Usage Management Mechanism (UMM) is injected into any cloud resources that are provisioned by the central UM framework.

By means of a web interface, the user logs into the system and is presented with a list of resources the user can access based upon the user's context. Each resource carries an associated policy, and these are stored in a repository. The policies are generated by the license generator based upon the SLA requirements. If the user wishes to access a resource, the user and environment information stored in the context is retrieved by the UM Framework and checked against the policy of the respective resource. If the user is granted access to the resource based upon the context and the policy of the resource, the resource and the associated policy file are moved to a virtual machine with respective security attributes pertaining to the policy. The operational details of our usage management system are visualized in Figure 5.4.



1. User logs in & the access control verifies his credentials.
2.a. User credentials are passed to resource manager.
2.b. Context information is passed to resource manager.
3. Resource manager compares the context with the policy set and gets the list of references to the available resources and displays it to the user.
4. User selects a resource.
5a. Policy manager retrieves the selected resource from Amazon S3.
5b. Policy manager creates appropriate VM instance either in public(Amazon EC2) or private cloud(Openstack)
5c. Control monitoring & processing.
6. Returns results of processing.

Figure 5.4: Component Diagram of Usage Management for hybrid cloud-based system.

The security and resources of private and public clouds have an inverse relationship that is, a private cloud is more expensive, and so contains fewer resources, but is more secure, while a public cloud is less expensive, and so contains more resources, but is less secure. The UM framework is also embedded inside the virtual machine.

Once the user has been granted access to the resource, the usage management system continues to process the policy agreements inside a virtual machine. The UMM is responsible for upholding policies throughout the lifetime of the resource and therefore accounts for any change in contexts. If the UMM determines that the context of the public cloud no longer confirms to the policy of the resource on it, then the resources are redacted.

UM Enforcement

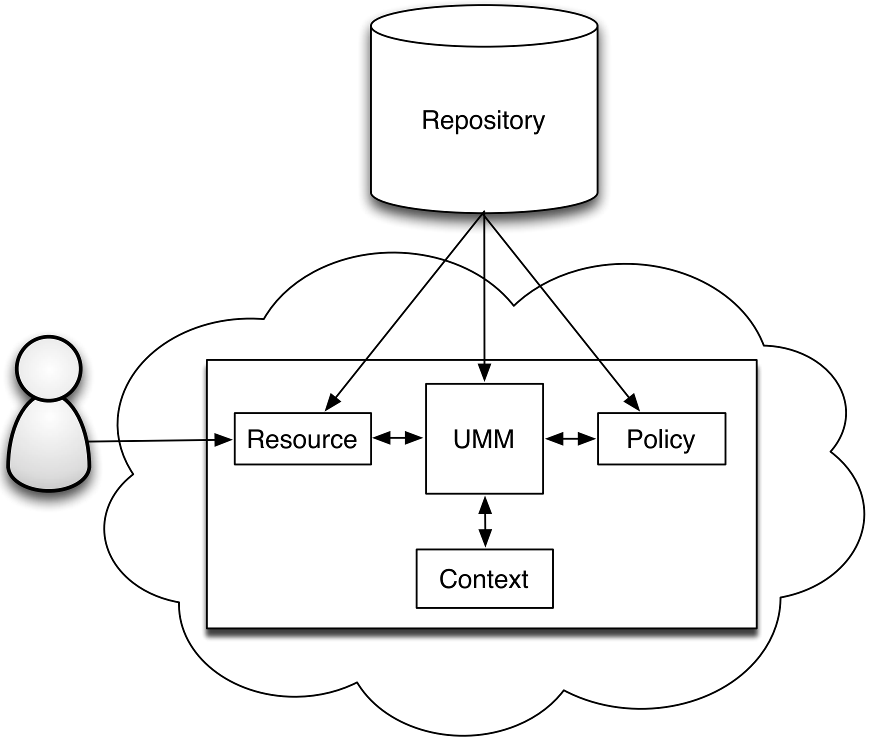


Figure 5.5: Usage management inside a VM.

Figure 5.5 illustrates the details of how the UM works inside a VM. Whether it is public or private, we use a base VM Image to create our VMs. With this approach we are able to successfully inject our UM Framework locally inside a VM. Thus, every VM comes with a UM mechanism already built into it. When a VM is created for a user to fetch the resource that the user wants to access, the associated policy files are also pulled along with the resource. The local UMM is responsible for upholding the policies of the resource within the VM. Any contextual change in user is reported to the UM mechanism within the VM by the Central UM Framework.

The central UM framework is intelligent; it monitors any changes in the database and correspondingly interacts and updates the UM Mechanism within the VM. The UM mechanism in turn verifies the policy of the resource inside this VM and gathers the user context and VM environment context to uphold the policy requirements of the resource. If the policy requirements of the resource are not met because of any contextual changes, then the resource is redacted. This is a preliminary approach. We are in the process of implementing a proof of concept for this enforcement of Usage Management inside a VM.

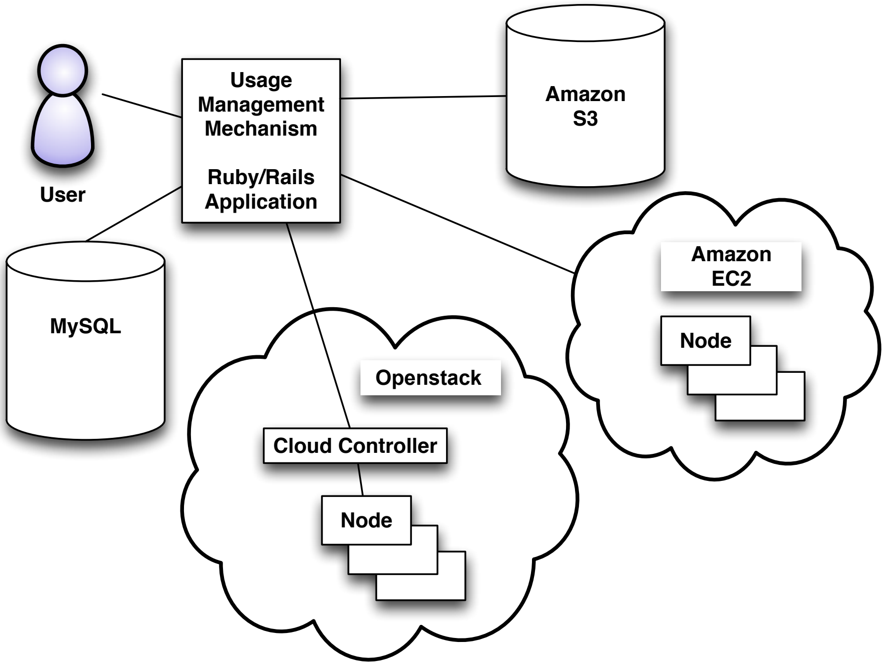


Figure 5.6: Technology Architecture

The implementation of the system is shown in Figure 5.6. The application is written using Ruby on Rails, and we use MySQL for storing user context. When a user logs in, we use the Ruby on Rails framework to authenticate the user. Once the user is authenticated we use Resource Manager written in Ruby to decide what resources the user can access by comparing context information with the policy set of the resources and the list of resource references are returned to the user interface. The user then selects a resource, and the policy manager retrieves the resource from Amazon S3 based on the policy information. The policy manager sends the resource to a VM running appropriate images either to OpenStack or to Amazon EC2 cloud. It is important to note the actual resources and corresponding policy sets are stored in Amazon S3, and some of the features of the OpenStack cloud system have been omitted from our diagram for the sake of brevity. With the help of our software we monitor the VMs for performance as defined in the SLA requirements and control the VMs appropriately.

Usage Management within a Virtual Machine

The key elements of a usage management mechanism that will be used in a VM are shown in Figure 5.7. When we consider UM at the VM level, the control is implemented with a finer level of granularity than is possible when implementing UM with a centralized mechanism because the centralized mechanism cannot control the actions within the individual VMs.

The process of license enforcement within a VM using a variant of our existing UM design takes place in six steps. First, an application queries the usage management mechanism about whether or not a particular action on a given resource by a given user is allowed. As a part of the request, the application provides information about the user, the resource, and the action that is to be performed. In step 2, the usage management mechanism obtains the current state of the computing environment from the operating system. The type of information obtained by the usage management mechanism depends upon the manner in which the context is specified. The information may include current location, date, time, IP address, etc. In step 3, the usage management mechanism generates the context instance by using the current values of system parameters, user information, and resource information. This is followed by step 4, in which the usage management mechanism queries an interpreter to determine the activity that corresponds

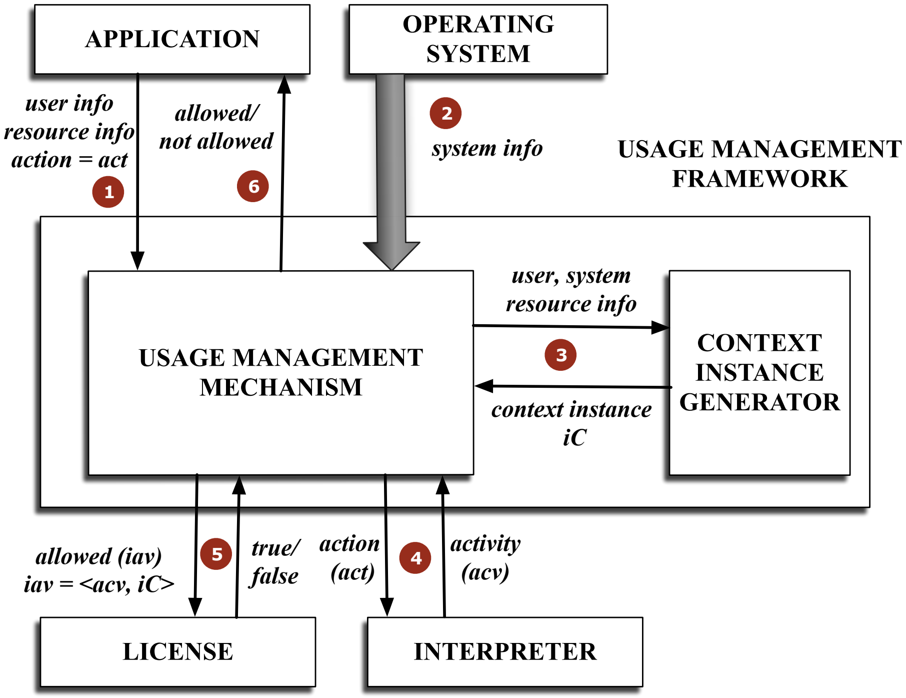


Figure 5.7: Operation of User Management framework within a Virtual Machine.

to the information provided by the application. Once the activity corresponding to the action is obtained, the usage management mechanism generates the activity instance denoted by the tuple. In step 5 the usage management mechanism invokes the allowed function provided by the license, the license executes an allowed function, and returns a Boolean value to the usage management mechanism. Finally, in step 6, the usage management mechanism notifies the application of whether the action is authorized.

The UM design provides the basis for deciding which operations are allowed and which are not. The outcome of these decisions must be transmitted, in a verifiable manner, to enforcement mechanisms in the operating system executing within the VM. The decision must be verifiable in order to ensure that the decision is properly enforced.

Chapter 6

Provisioning Security and Performance Optimization for Dynamic Cloud Environments

As mentioned in the Introduction, one of the goals of our approach is to calculate the optimal distribution of resources among different users of IaaS cloud while offering different levels of security. The inherent randomness of the behavior of the cloud complicates the search for an optimal solution. The ever-increasing number of potential users and available virtual resources motivates the implementation of an scalable methodology. Within the scope of SLA requirements between users and cloud service providers, we propose to optimize the distribution of cloud resources based on user needs, while guaranteeing power savings to the provider. In this section, an unconstrained optimization problem is proposed and a description of its mathematical framework is provided [54].

6.1 Cost Function

Because of the random variation of the workload in the cloud, we propose to model the following performance measures as *Random Variables* (RVs), as follows,

$$\begin{aligned} C_\mu &= \text{\% of CPU utilization} \\ M_\mu &= \text{\% of memory utilization} \\ T &= \text{Total execution time of benchmark} \\ \overline{S} &= \text{Measure of Security, (see Section 6.2)} \\ W &= \text{Hourly cost associated to the VMs in use} \end{aligned}$$

Let us define the set Θ as the set of VMs offered to the clients, *e.g.,* based on the Amazon EC2 literature we can define Θ as a subset of the general purpose instance family of the EC2 service [21], as follows,

$$\Theta \equiv \{\text{m1.small},\ \text{m1.medium},\ \text{m1.large},\ \text{m1.xlarge}\}. \tag{6.1}$$

Except for the processing time $T \geq 0$, all the foregoing variables are upper and lower bounded by finite real numbers. It is reasonable to assume an upper bound for T that can be statistically estimated by a randomized algorithm for *probabilistic worst-case performance* [55]. By defining upper and lower bounds for the RVs, we are able to normalize the variables and define the normalized versions, $\overline{C}_\mu, \overline{M}_\mu, \overline{T}, \overline{W} \in [0,1] \subset \mathbb{R}$.

Now, let us define the vector,

$$\Delta = \left(\overline{C}_\mu, \overline{M}_\mu, \overline{T}, \overline{S}, \overline{W}\right)^T.$$

Every random vector $\Delta \in \mathcal{D}$ is considered a sample of the performance vector of the system. Moreover, by assuming that a unique benchmark problem is used to measure performance, every normalized RV can be assumed to be a measurable function of $\theta \in \Theta$. where θ is one of the available VMs.

Let us assume that users can be grouped under some criteria, *e.g.,* geographical location or security requirements. We assign an index $i \in (1, \ldots, N) \subset \mathbb{N}$ to each client in the group, and for the i-th client we propose the cost function $J_i(\Delta, \theta)$ s.t., $J_i : \mathcal{D} \times \Theta \to [0,1]$ given by,

$$J_i = \mathbf{E}\left\{\alpha_{1i}\overline{C}_{\mu i} + \alpha_{2i}\overline{M}_{\mu i} + \alpha_{3i}\overline{T}_i + \alpha_{4i}\frac{1}{\overline{S}_i} + \alpha_{5i}\overline{W}_i\right\}, \tag{6.2}$$

then we define the multi-cost function,

$$\mathbf{J} = \left(J_1, \ldots, J_N\right)^T, \tag{6.3}$$

where $\alpha_{1i}, \ldots, \alpha_{5i} \in [0,1] \subset \mathbb{R}$, s.t. $\sum_{j=1}^{n} \alpha_{ji} = 1$, correspond to the weights given to the variables based on the SLA requirements of the i-th client in the cloud [7].

Remark 1. *Although minimizing $\overline{C}_\mu$ affects the throughput of the instance, it means power savings in the data center hosting the virtual resources.*

Remark 2. *It is expected that $\overline{C}_\mu$ and $\overline{M}_\mu$ are inversely proportional to $\overline{T}$, so the weights $\alpha_{ki}, k = 1 \ldots 5$, would determine whether minimizing $\overline{C}_\mu$ and $\overline{M}_\mu$ is more important than minimizing $\overline{T}$ based on the user's needs.*

6.2 Measure of Security

Securing information stored in the cloud is a crucial problem. In this approach, we focus on data encryption in virtual resources. However, given the variety of encrypting ciphers, measuring security levels is not straightforward.

For this specific problem we propose assigning values to different security levels as shown in Table 6.1. In the first column we have the security measure values given by numbers between 1 and 10, in order to guarantee that $1/S_i \in (0,1]$. In the second column we describe the cipher, the key size and its mode of operation by using initials. The lowest value 1 is assigned to the *No encryption* option. The next security level value of 2 goes to the old Data Encryption Standard (DES). After that, with a value of 3, follows the Advanced Encryption Standard (AES) with the Electronic Codebook (ECB) mode of operation and a key size of 128 bits. Following the sequence, we proceed to increase the key size up to 256 bits, and then we proceed to add enhancements by progressively changing the modes of operation which go from Cipher-Block Chaining (CBC), Cipher Feedback (CFB), Cipher Feedback with shift registers (CFB-1/CFB-8), Output Feedback (OFB) and Counter (CTR). All the relevant advantages and disadvantages of each mode of operation have been specified in the third column of the table.

Encrypting and decrypting data implies an overhead that affects CPU and memory utilization, therefore the client might have no interest in adding additional security to the service. In that case, the weight α_{4i} in (6.2) would get a value close to zero.

Using Table 6.1, we are certain that security performance increases going from top to bottom. This is not only based on the key sizes, but on the ability of the cryptographic algorithm to be run in parallel (faster), synchronizable (able to correct bit errors in finite time), and be unsusceptible to cryptanalysis.

6.3 Statistical Learning

Aside from uncertainty of the system, the size of the sample space of such an optimization problem encourages to look for alternatives to be able to get quantitative conclusions about an optimal or suboptimal solution. Statistical Learning presents an alternative to solve this problem. It takes advantage of powerful results associated to *Monte Carlo Simulations* and the *Uniform Law of Large Numbers*.

Under a rigorous mathematical framework, the available literature offers necessary conditions to design efficient *Randomized Algorithms* to estimate a cost function whose closed form is not available. However, these algorithms are not guaranteed to work all the time but most of the time.

From (6.2) and (6.3) we see that since we do not have knowledge of the probability distributions of the RVs, we are unable to determine a closed form of the multi-cost function. However, based on [55], under the assumption that $J(\Delta, \theta)$ is measurable we can estimate it by calculating the sample mean for a number of $M_2 \in \mathbb{N}$ samples of Δ.

$$\hat{\mathbf{E}}_{M_2}\left(J(\Delta, \theta)\right) = \frac{1}{M_2} \sum_{k=1}^{M_2} J(\Delta^{(k)}, \theta). \tag{6.4}$$

Since we are limited to a finite number of samples, we are subject to errors. However, we can assume certain confidence and accuracy of our estimate of $J(\Delta, \theta)$ and calculate the minimum number of samples of

$\Delta \in \mathcal{D}$ and the minimum number of instances $\theta \in \Theta$ that guarantee the assumed confidence and accuracy. This is illustrated by the following result from [55] which we present without proof.

Theorem 1. *Assume that the cost function* $J(\Delta, \theta)$ *is measurable s.t.* $\mathcal{D} \times \Theta \to [0, 1]$*. Given* $\epsilon_1, \epsilon_2, \delta \in (0, 1)$*, let,*

$$M_1 \geq \frac{\ln \frac{2}{\delta}}{\ln \frac{1}{1-\epsilon_2}}, \quad M_2 \geq \frac{\ln \frac{4M_1}{\delta}}{2\epsilon_1^2}. \tag{6.5}$$

Then, with confidence $1 - \delta$*, it holds that*

$$P_R \left\{ \mathbf{E}(J(\Delta, \theta)) < \hat{\mathbf{E}}_{M_2}(J(\Delta, \hat{\theta}_{M_1 M_2})) - \epsilon_1 \right\} \leq \epsilon_2.$$

with,

$$\hat{\theta}_{M_1 M_2} = \arg \min_{l=1,\ldots,M_1} \hat{\mathbf{E}}_{M_2}(J(\Delta, \theta^{(l)})). \tag{6.6}$$

From (6.6), notice that $1 - \delta$ is a measure of confidence and $1 - \ \epsilon_n, \ n = 1, 2$ are measures of accuracy. As a simple example, if we assume $\delta = \epsilon_1 = \epsilon_2 = 0.05$, then $M_1 \geq 72$ and $M_2 \geq 1,133$. This means that by assessing 72 combinations of instances $\theta \in \Theta$ for all the N users (*i.e.*, $4 \times N$), and for each parameter we take $1,133$ samples of $\Delta \in \mathcal{D}$ to calculate (6.4), we can assert with a confidence of 95% that we find a minimum of the cost function (6.2) with accuracy 95%.

Finally, the randomized algorithm for this case as is proposed in [55] is,

Algorithm 1. *1. Determine* M_1 *and* M_2 *according to Theorem 1.*

2. Draw M_1 *independent identically distributed samples of the performance function*

$$\hat{\mathbf{E}}_{M_2}\left(J(\Delta, \theta^{(1)})\right), \ldots, \hat{\mathbf{E}}_{M_2}\left(J(\Delta, \theta^{(M_1)}\right).$$

3. Draw M_2 *independent identically distributed samples* $\Delta^{(1)}, \ldots, \Delta^{(M_2)}$.

4. Return the empirical instance,

$$\hat{\theta}_{M_1 M_2} = \arg \min_{i=1,\ldots,M_1} \frac{1}{M_2} \sum_{k=1}^{M_2} J(\Delta^{(k)}, \theta^{(i)}).$$

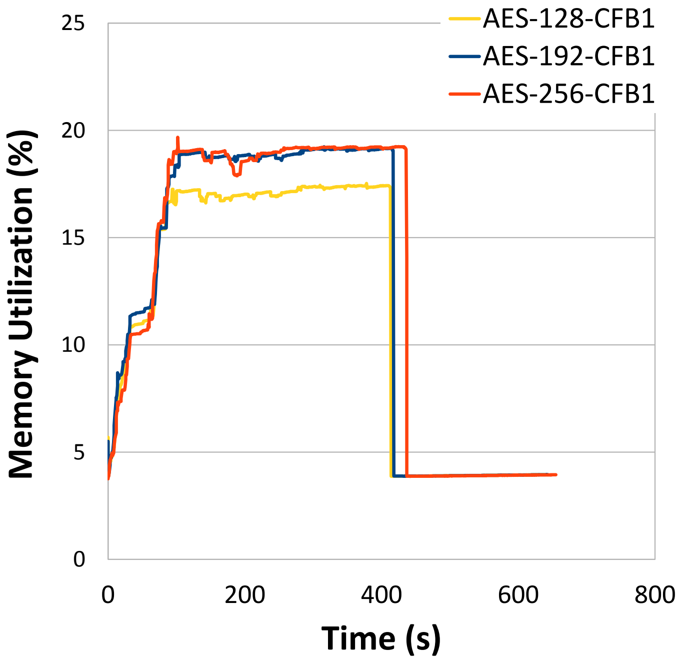


Figure 6.1: Memory overhead produced by the implementation of the AES cipher with key sizes of 128, 192 and 256 bits, while running the eclipse benchmark from the dacapo benchmark suite.

6.4 Encryption and Benchmark Overhead

By incorporating security in the optimization process we are considering the overhead produced by the ciphers over the performance of the instance. In this example, a video file of 4.7 GB was being encrypted while the eclipse benchmark from the DaCapo benchmark suite [56] was running in an Amazon EC2 VM of the type m1.large. The plot in Figure 6.1 illustrates the effect of the increase in the key size. Notice that the response that uses the less memory is the AES cipher with a key size of 128 bits. Although the memory utilization for the key sizes 192 and 256 bits do not seem to differ on the average, notice that the running time T of the benchmark is 437 s for the AES of 256 bits (red line) in contrast to 415 s for the AES of 192 bits (blue line). The abrupt decay that is seen in the plot indicates the instant where the benchmark completes 10 repetitions and then stops running. These responses justify the presence of the variables $\overline{S}_i$, $\overline{M}_{\mu i}$ and $\overline{T}_i$ in the cost function (6.2).

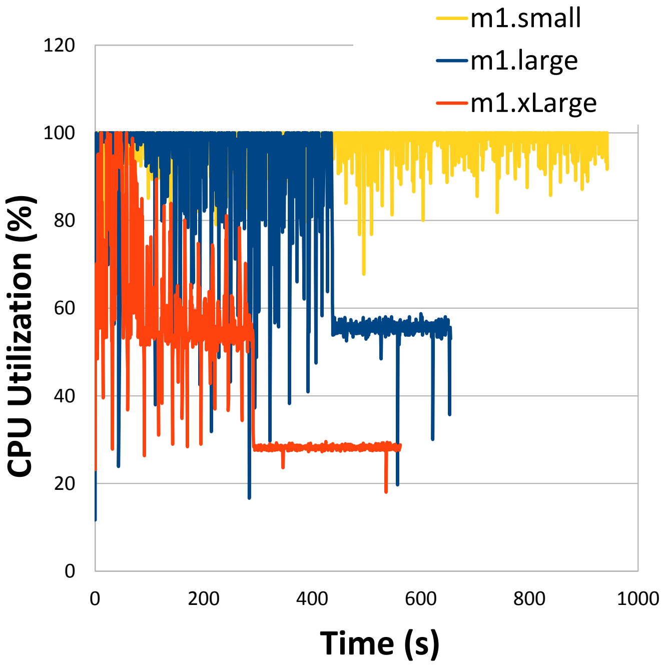


Figure 6.2: CPU overhead produced by the implementation of the AES cipher with key size 256 bits, while running the eclipse benchmark from the DaCapo benchmark suite for three different VMs..

Furthermore, in Figure 6.2 we illustrate the effect of the change of instances over the CPU utilization while encrypting using AES with key size 256. The VMs used were m1.small (yellow line), m1.large (blue line) and m1.xlarge (red line) [21]. It can be inferred that as the capacities of the VMs increase, the CPU utilization (C_μ) decreases on the average and the benchmark running time T decreases. These responses justify the incorporation of the variable $\overline{C}_{\mu i}$ and $\overline{W}_i$ in (6.2).

Table 6.1: Measure of security associated to ciphers and modes of operation.

Sec. Lvl $\overline{S}_i$	**Cipher** **Mode of Op.**	**Observation**
1	No encryption	Determined by $\alpha_5 = 0$ as well
2	DES	Short key sizes, 64-bit encryption blocks, issues with large files, prone to cryptanalysis
3	AES-128-ECB	Does not hide data patterns well, 128-bit key size
4	AES-192-ECB	Does not hide data patterns well, 192-bit key size
5	AES-256-ECB	Does not hide data patterns well, 256-bit key size
6	AES-256-CBC	Non-parallel encryption, 256-bit key size
7	AES-256-CFB	Non-parallel encryption, 256-bit key size, No padding
8	AES-256-CFB-1 AES-256-CFB-8	Non-parallel encryption, 256-bit key size, Synchronizable, No padding
9	AES-256-OFB	Non-parallel encryption, 256-bit key size, Synchronizable, faster block cipher operations
10	AES-256-CTR	Parallel encryption, 256-bit key size, Synchronizable

Chapter 7

Policy Generation and Enforcement for Software defined Network in Cloud Systems using Usage Management

7.1 Design & Approach

This research presents a framework that enhances security in a cloud based environment. It is easy to use and does not impose learning requirements on network administrators. This section explains the entire framework and its working in detail.

7.1.1 Access Control

The system has a web GUI (Graphical user interface), which is where the user registers himself and describes the classification level he is operating on, and thus, based on his context, the system determines the security level he has access to. Context could include both user and his environment information, such as geographical information or the identity of his machine in the network. The SDN controller we are using is OpenDaylight and the programming language used to generate these policies is Ruby [57].

7.1.2 Entire Framework

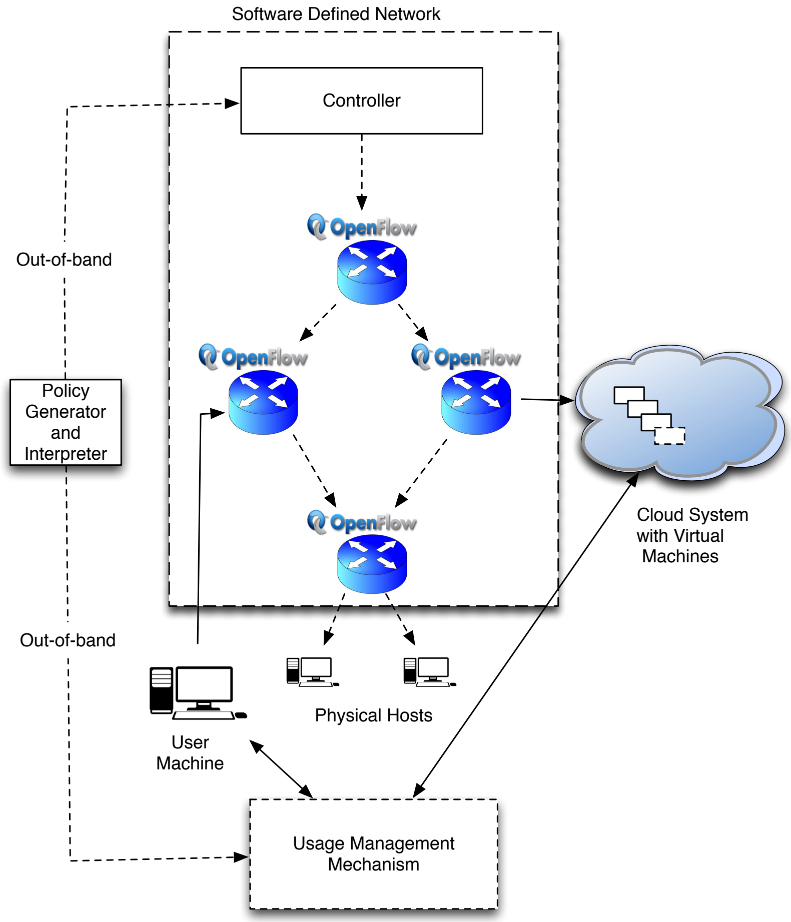


Figure 7.1: System Architecture diagram: User machine is accessing the physical and virtual machines via the OpenFlow switches based on the user's security privileges.

As shown in Figure 7.1, there are three major blocks: the Usage Management Mechanism, the Software Defined Network, and the Cloud System. When a user tries to get information, they can do so by requesting the UMM, which fetches the resource from either the virtual or physical machine, without the user getting direct access.

In the case where the user wants direct access to the VM or physical host for some manner of computation, the the OpenFlow mechanism comes into play and creates a data path between the user and the virtual or physical hosts.

The policy generator and interpreter module installs policies onto the UMM using a the UMM's REST API, and installs flows onto the SDN controller by using OpenDaylight's REST API. These flows and policies combine to enforce the behavior model specified by an administrator. The UMM's REST API is facing the northbound side of the network, so the UMM and policy generator have an out-of-band interface by which to communicate policies.

7.1.3 Architecture

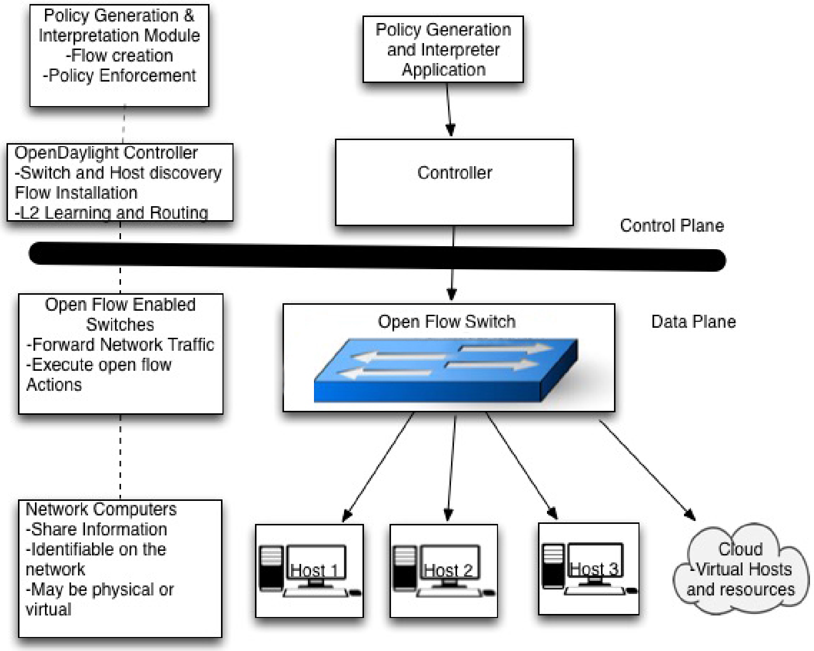


Figure 7.2: SDN and application architecture The blocks on left side explain what is happening at each level on the right.

The design and architecture of this follows a conventional SDN model, with an out-of-band OpenFlow controller that controls the SDN switches [24]. As shown in Figure 7.2, our application interacts with the northbound interface of the SDN controller. In our setup, the controller we use is OpenDaylight, which will take care of the basic network functions such as L2 learning and routing, and provides an excellent REST API [58]. This makes it simple to interact with the network without having to manually handle low-level network functions in our application.

All policies are installed on the application and enforced at a higher match priority than the low-level network function flows that OpenDaylight automatically installs. This allows us to seamlessly operate with network activity that has no policies applied to it, and only protect network operations and machines that are part of policy groups.

7.1.4 Mandatory Access Control

Traditional *type enforcement* models separate *subjects*, which are generally processes, and *objects*, which may be files, subjects, or other resources [13]. This model works well because the subjects are the only classes that can initiate an action, while objects receive an action. This is not the case in networks, where machines may initiate connections to other machines and effectively act as peers [59].

It is possible to create network entities as subjects and objects, and in fact it may be easier to do so by correlating policies to one-way SDN flows. However, the most common use of computer networks is TCP traffic [60], which requires two-way communication. In this case, policies must be sensitive to the symmetric network traffic, and in order for a subject to interact with an object the policy must be enforced in such a way that it permits this communication.

7.1.5 Policy Language

The policy language is designed to operate seamlessly with the policy enforcement mechanism, as well as offer support for programmatic extension. The policy language accepts a definition file of subjects, objects and enclaves, as well as policies between pairs. Any programmatic extension is expected to compose its policies into pairs. Any policy that is definable can be defined as a collection of policies between pairs of entities.

A machine is defined as a computer that has a MAC address and a IP address. A machine can be grouped with others in order to form a set of *machines*.

$$machines = \left\{ \begin{array}{ll} id & 1 \\ mac\_address & 43:32:44:b3:01:c3 \\ ip\_address & 156.180.60.47, \\ & \\ id & 2 \\ mac\_address & a2:d9:25:34:40:cd \\ ip\_address & 44.124.252.175 \end{array} \right\}$$

A flow corresponds to match fields in a flow table and can be grouped into a set of multiple *flows*.

$$flows = \left\{ \begin{array}{ll} id & 1 \\ transport\_src & 80 \\ transport\_dst & 80 \end{array} \right\}$$

Enclaves are sets of multiple *members*, where *members* can be either groups of machines or flows.

$$Enclaves = \left\{ \begin{array}{ll} id & = 1, \\ members & = \left\{ \begin{array}{ll} type & machine \\ id & 1, \\ & \\ type & machine \\ id & 2, \\ & \\ type & flow \\ id & 1 \end{array} \right\} \end{array} \right\}$$

A policy consists of four main parts: a *subject*, *object*, *action*, and *priority*. The subject defines the source and the object defines the destination. Both subject and object can exist as either a machine, a flow, or an enclave. An action is a pre-defined operation described by OpenFlow specification, and a *priority* is the preference given to a flow that defines the order in which a flow is processed in relation to others.

$$policy = \left\{ \begin{array}{ll} subject & = \left\{ \begin{array}{l} type = enclave \\ id = 1 \end{array} \right\} \\ object & = \left\{ \begin{array}{l} type = enclave \\ id = 2 \end{array} \right\} \\ action & = allow, \\ priority & = 600 \end{array} \right\}$$

This particular *policy* will establish a network where enclave 1 and enclave 2 can communicate with each other.

7.2 Policy Generation

Now we must explain how to generate a set of policies for a group based on an arbitrary network behavior model. The policy generation engine can programmatically generate policies based on an intended network behavior model, such as the Bell-LaPadula or Biba models [52]. This requires the specification of all enclaves as a set where each enclave has a particular attribute such as 'security' or 'integrity level.' This framework is already prepared to operate in scenarios where these enclaves are disjoint sets (no network entity is contained in an enclave more than once).

As a proof of concept, we will use the **Bell-LaPadula** model, which has two given properties:

Simple Security Property: No subject may read an object at a higher security level.

⋆ **- Property**: No subject may write to an object at a lower security level.

These policies shall have the following format:

```
[ subject: ‘‘name’’, object: ‘‘name’’,
          action: ‘‘action’’ ]
```

In this case, our actions may be “read,” “write,” or “read/write.” Let us define an abstract set of subject and objects. For the purpose of this study, we consider these to be homogeneous, i.e. subjects and objects are the same type of entity (machines on a network). We will refer to them as machines.

Now we define a list of machines: Alice, Bob, Charles, Dave, Eva, Frank, Gary, and Henry.

Now we define a set of enclaves with a given security level:

Enclave	Security Level
Public	0
Low Security	1
Medium Security	2
High Security	3

Next we define the memberships of those enclaves:

Enclave	Members	
Public	Alice	Bob
Low Security	Charles	Dave
Medium Security	Eve	Frank
High Security	Gary	Henry

The policies must then be enforced. Let E be the set of all enclaves, with members as elements. Note that each enclave is a disjoint set:

$$\mathcal{R} = \{(s, o) \mid (s, o) \in E \times E, s \neq o\}$$

Then we check each ordered pair against a security level. Let $P(s, o)$ be a policy from one object to one object in $\mathcal{R}$, where s represents the security level of the “subject” and o represents the security level of the “object.”

If the security level of the subject is greater than the security level of the object, then the policy between them would be “read:” the source can read from the object. This is shown in (1) below.

If the security level of the subject is less than the security level of the object, then the policy between them would be “write:” the source can write to the object. This is shown in (2) below.

If the security level of the subject is the same as the security level of the object, then the policy between them would be “read/write:” the source can both read from and write to the object. This is shown in (3) below.

$$s > o \rightarrow P(s,o) = read \tag{7.1}$$

$$s < o \rightarrow P(s,o) = write \tag{7.2}$$

$$s = o \rightarrow P(s,o) = read/write \tag{7.3}$$

7.2.1 Generate policies

Policies determine the action between two enclaves based on their security levels. The action is associated to every ordered pair of enclaves and these form the policies.

Algorithm 1 generates policies between two enclaves, and Algorithm 2 generates the final policies by assigning action for each subject and object based on the policies between enclaves, which are generated in Algorithm 1.

7.2.2 Policy Model

On a small network it may be reasonable to enforce policies that affect networks with particular network flow attributes, such as a TCP destination port. However, on larger networks, and in particular for multi-level security, it is preferable to be able to group flows into enclaves. This enclave model can be characterized as attributes, as a policy object may be entered into several enclaves.

Algorithm 1 Generate the policies between enclaves

```
procedure PolicyGen(Enclaves)
    Policies = [ ]
    for all enclave ∈ Enclaves × Enclaves do
        if enclave[1].level = enclave[2].level then
            push [action: "read/write"] onto enclave
        else if enclave[1].level < enclave[2].level then
            push [action: "write"] onto enclave
        else if enclave[1].level > enclave[2].level then
            push [action: "read"] onto enclave
        end if
        Add enclave to Policies
    end for
    return Policies
end procedure
```

Representation of Network Entities

Each network entity can be represented with a set of OpenFlow values, including switch connections, physical ports, MAC addresses, IP addresses, TCP ports, VLAN IDs, and anything else compatible with OpenFlow 1.3. These network attributes are stored in a key-value store, with the network attribute type as a key. Flows and abstractions may be built, on top of the structure.

1) Flows and abstractions: in order to satisfy symmetrical relationships between network entities, it can be helpful to add abstractions to network flows. A network flow, within the scope of our application, is defined as the set of all possible match fields that our OpenFlow controller supports. An abstraction will have a proper subset of these match fields. Because match fields are asymmetric (source and destination), abstractions may also contain a symmetric match field, which will then be split into the flow going from the abstract entity, as well as attending to it.

Algorithm 2 Generate the low level, paired policies

```
procedure PolicyLowered($Policies, Memberships$)
    FinalPolicies = [ ]
    names = (u,v) | (u,v) ∈ Memberships
            × Memberships, u ≠ v
    for all name in names do
        subject = name[1]
        object = name[2]
        action = get action for subject and object enclaves from Policies
        push into FinalPolicies(subject,object,action)
    end for
    return FinalPolicies
end procedure
```

Example of policies being generated:

PolicyLowered (PolicyGen(enclaves) , memberships)

= [:subject = "Alice" , :object = "Bob", :action = "read/write",

:subject = "Bob", :object = "Alice", :action = "read/write"]

Let N represent the set of all possible match fields and let K,V denote the key-value pair. Each flow may be represented as follows:

$$F = \{(K, V)^c \mid K \in N, K_i \neq K_j \forall i \neq j, 0 \leq c \leq |N|\}$$

As an example, we define a machine, which has an address, an IP address, a node (switch to which it is connected) and node connector port (physical port number on its node). While flows are very specific, unidirectional from one source to one object, and are therefore inherently asymmetric, machines will have a symmetric relationship—a bidirectional relationship—and have policies enforced to allow asymmetric communication between machines.

As shown in Section 7.1.2 we represent machines as having a particular subset of the flow attributes.

The machine has a node and a node connector port. The machine also has abstracted attributes, which are its MAC address and IP address. These are symmetric, meaning it must be divided into an inbound and outbound flow. From this we can generate two flows for each machine.

Let N_s and N_d represent all the attributes that belong to the source and destination flows respectively. Let $\overleftarrow{M}$ be the inbound flow for machine M, and $\overrightarrow{M}$ be the outbound flow for machine M.

$$\overleftarrow{M} = \{(K,V)^c \mid K \in N_d, K_i \neq K_j \forall i \neq j, 0 \leq c \leq |N_d|\}$$
$$\overrightarrow{M} = \{(K,V)^c \mid K \in N_s, K_i \neq K_j \forall i \neq j, 0 \leq c \leq |N_s|\}$$

We define a special operator called Hashmerge and for the simplicity of notation, denote it by $\uplus$ Now, Hashmerge is defined as:

$$\begin{aligned} Let, C_1 &= \{\{K_1, V_1\}, \{K_2, V_{21}\}\} \\ Let, C_2 &= \{\{K_2, V_{22}\}, \{K_3, V_3\}\} \\ C_1 \uplus C_2 &= \{\{K_1, V_1\}, \{K_2, V_{22}\}, \{K_3, V_3\}\} \end{aligned} \tag{7.4}$$

Given two sets of key-value pairs, C_1 and C_2, which both contain the same key, when the sets are merged the value associated with the key of the second set is taken as the value associated with that key in the resulting set. Table 7.3 shows the results of the Hashmerge between Table 7.1 and Table 7.2. The Hashmerge is a non-commutative operator, however, by commuting the terms we have a simple mechanism for conflict checking. In cases where $C_1 \uplus C_2 \neq C_2 \uplus C_1$ then the policy has a conflict and is not compatible with the policy mechanism we describe in the next section.

Policy Flow Generation

Given a policy between two machines, the policy flow generation is straightforward. The inbound flow from the subject is guaranteed not to have a conflict with the outbound flow from the object machine, and vice versa. Since both the sets are disjoint, the Hashmerge would behave as a union operator in this case. This can be represented as the union of both pairs of flows.

Let M_s be the subject machine and M_o be the object machine. Let $\overleftarrow{M_s}$ and $\overleftarrow{M_o}$ be the inbound flows for the subject and object machines respectively. Likewise, let $\overrightarrow{M_s}$ and $\overrightarrow{M_o}$ be the outbound flows for the subject and object machines, respectively.

$$P(M_s, M_o) = \overrightarrow{M}_s \cup \overleftarrow{M}_o = \overrightarrow{M}_s \uplus \overleftarrow{M}_o$$

$$P(M_o, M_s) = \overrightarrow{M}_o \cup \overleftarrow{M}_s = \overrightarrow{M}_o \uplus \overleftarrow{M}_s$$

Furthermore, a policy may be imposed between a machine and flow. While a flow may have characteristics that are not provided in a machine abstraction, it may also have attributes that a machine may have. For example, if a flow only has a TCP port, and the machine does not, the policy must be enforced symmetrically to that machine. If a flow has an IP address source, and a machine has an IP address, that policy should only be enforced with the machine's IP address as the destination IP address.

Let F be the flow and M be the machine.

$$P(F, M) = F \uplus \overrightarrow{M}$$

$$P(M, F) = F \uplus \overleftarrow{M}$$

Because a policy between two flows is always asymmetric (creates a single flow), the union operator only needs to be applied once. Unlike in the case of machines, the policy between is symmetric (creates two flows). The rule will be the same, any flows with matching keys that have non matching value.

Let F_1 be the subject flow and F_2 be the object flow.

$$P(F_1, F_2) = F_1 \uplus F_2$$

When a policy is applied between two enclaves, the policy must be applied between each pair in each enclave. From there, the policy enforcement becomes the same as a policy enforcement between a singleton pair, with the exception that the policy does not need to be installed both ways if singleton objects are members of the subject and object enclaves. This logic can be displayed by representing these enclaves as a set relation.

Let E_s be the subject enclave and E_o be the object enclave, and F be the set of all ordered pairs where both are elements of flows.

$$P(E_s, E_o) = ((E_s \times E_o) \cup (E_o \times E_s) \setminus F)$$

This will generate the correct and minimum number of individual policies to install. The resulting policy set is a symmetric set. If E_s and E_o are distinct sets (the sets differ by at least one element), then the final result may be asymmetric. This being the case, we add values to make it symmetric.

MAC source	IP source
00:0A	10.0.0.1

Table 7.1: Flow Table 1

MAC destination	IP destination
00:0B	10.0.0.2

Table 7.2: Flow Table 2

MAC source	MAC destination	IP source	IP destination
00:0A	00:0B	10.0.0.1	10.0.0.2

Table 7.3: Flow Table 3 = Hashmerge Table 1 and Table 2

One important thing to note is that a switch only processes network entities. The machines, policies and enclaves are an abstraction of those network entities and ultimately boil down to a network flow that the network switch responds to.

7.2.3 Installing Flows On Switches

After compiling the policies into OpenFlow flows, the OpenFlow flows must be installed on relevant switches. For machine and flow abstractions that have a Node ID attribute, the policy is stored on the node specified for the machine or node specified as the source or subject.

For policies that do not have a member specified, the flow must be installed on all applicable switches. For now, the number of installed flows stays manageable by partitioning the network using FlowVisor[35], which delegates control of switches to multiple controllers in OpenFlow. In future work we will examine installing flows without switch specifications only on applicable switches.

7.2.4 Policy Database

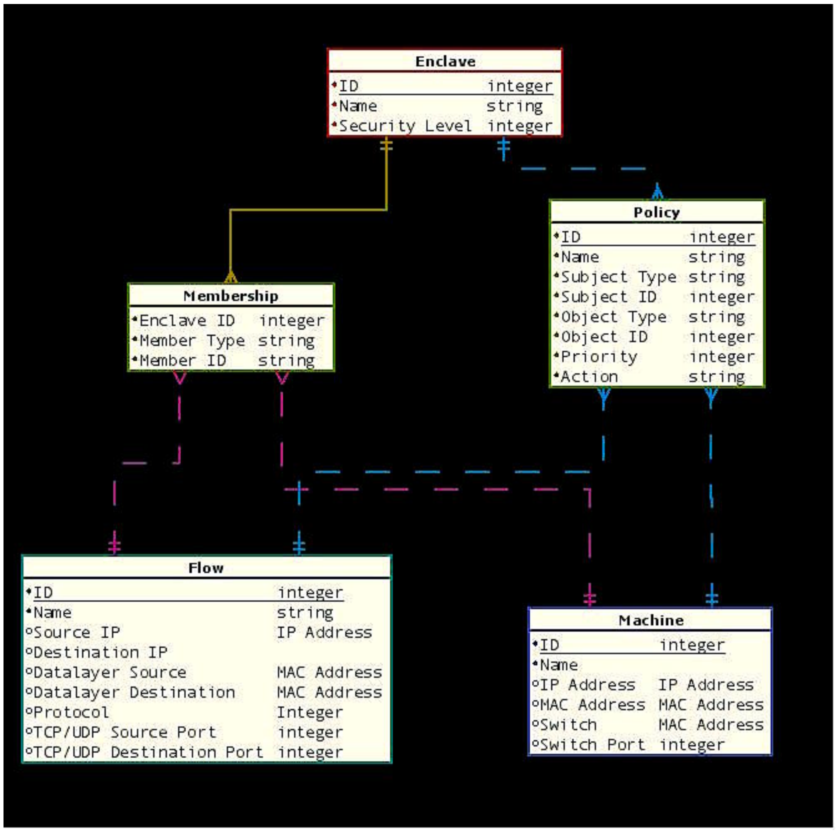


Figure 7.3: Entity-Relationship Diagram where dashed lines represent polymorphic associations

These policies are stored in a relational database. Fig. 7.3 shows the Entity-Relationship diagram. The database has the following tables:

- **Enclave** is table containing the name of the enclave and security level.
- **Flow** is table containing the name of the flow and flow attributes for e.g. transport source port, transport destination port.

- **Machine** is table containing the name of the Machine and Network attributes such as MAC and IP address.
- **Membership** is a polymorphic join table to give machines and flows memberships within enclaves.
- **Policy** is a polymorphic join table between enclaves, machines and flows that gives a priority and an action to this relationship.

This database structure enables any policy language to be used that may enter traits that can be recognized by OpenFlow, and given actions and policies between two enclaves or collection of network attributes.

7.3 Information Architecture

Fig. 7.4 shows the overall information architecture diagram.

Although the policy model we use for this example is the Bell-LaPadula model, any policy model that can be represented as a logic block can be used here – the Biba model, for example.

As discussed in the previous section, both subject and object machines are made up of network attributes. They also have a context and a security level at which they can operate. The policy generator takes the subject and object and a policy model (e.g. Bell-LaPadula model) to generate policies.

Policy Model: A logic block that determines interactions between subjects and objects. In our example, we use the Bell-LaPadula Model as a logic block.

Machine, Flow and Enclave inherit the properties of the Network Entities class.

Policy Generator: Generates policies given a subject, object and Policy model.

Subject: The source of the network communication with a context that is described by the Security Level. The subject may be a Machine, Flow or Enclave.

Object: The destination of the network communication. Like the subject, the object also inherits the

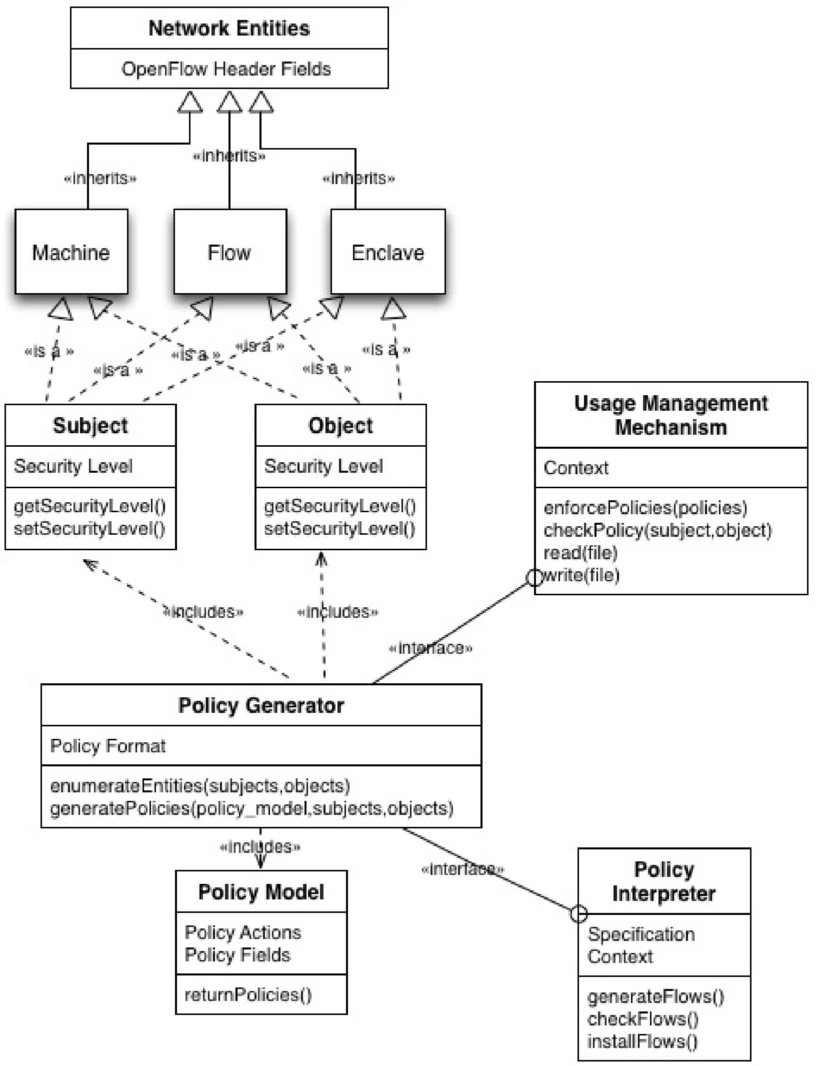


Figure 7.4: Information Architecture for SDN in Cloud systems.

properties of the Network Entities class and has a context described by the Security Level. The object could be a Machine, Flow or Enclave.

Interpreter: When the subject needs direct access to the object, then the interpreter is used to create the data path between them. The interpreter processes the generated policies and converts them into flows to be consumed by the OpenFlow switch. It serves an interface to the policy generator.

Usage Management Mechanism: Acts as a Network Functions Virtualization module and enforces the policies that do not require the OpenFlow mechanism to create a data path. It enforces the read/write policies between subjects and objects. It also serves an interface to the policy generator.

7.4 Result

The space analysis can be modeled using a complete digraph. In a complete digraph the bound, given n nodes (These Nodes are the endpoints of the network e.g. machines) as n grows large (worst case analysis) is represented as $n(n-1)$ edges [61]. If we assume that the number of switches in a given context is constant, this gives us a space bound of $O(n^2)$. Hence, a framework, such as the one we presented, makes it easier for the user to enforce policies using OpenFlow.

Chapter 8

Conclusions

For some time now, the cloud has been looked at as a solemn promise for massive collaboration and as a means to handle most users' infrastructural and software needs—and rightly so, because it offers a cost-effective, high-performance capable option, both of which can be enabled on demand. However, despite the ease and capability that the cloud offers, large-scale usage of cloud has not dawned, especially among business or social organizations.

One of the major reasons for this has been the minimally addressed issue of security. Current security measures offer limited security to users, and there is an absence of a mechanism that can translate the service-level agreement into the cloud allocation chain. An increase in usage of cloud for SaaS impresses upon us a greater need for assured information sharing.

The nature of the cloud's function substantiates that a context-oriented mechanism is aptly suited. A policy-based approach that regulates data access and transmission and complies with confidentiality and integrity seems essential.

This research yielded an architecture currently realizable with modern open-source tools that enables this kind of dynamic information control. It provides a prototypical implementation of the proposed system architecture. The approach is an abstraction based on an amalgam of Usage Management and Software-Defined Networking. The approach includes a high-level, web-based interface that will aid the network administrators to realize policy flows without performing the painstaking task of forming and enforcing them.

While maintaining security is an essential aspect of the management of services in a cloud-based system, it is also necessary to guarantee performance. Obtaining this balance is a particularly hard problem, as the overhead of application service increases with an increase in security requirements. This research presents a novel approach to optimize the distribution of virtual resources between clients of IaaS cloud using Usage Management framework in conjunction with Statistical learning. A control systems approach based on model identification and proportional thresholding is used. This assures performance regulation whilst satisfying the security requirements of the applications served on multiple cloud systems.

Our technology is a full stack: it is a framework that works for both cloud computing and SDN. It increases the ease with which network applications can be implemented, and reduces risk of misunderstanding and errors on the users end. Given a subject, an object, and a policy model, our technology can automatically generate policies that a user or administrator might otherwise have to create manually. By developing a more user-friendly policy generation system, we provide users more refined control over their personal and business networks, and streamline the process of network maintenance and growth for system administrators. This makes enforcement of complex behavior models, such as those found in multi-level security, simple to enforce. As well, the framework interprets the policies that it generates and sends the cloud policies to the appropriate UM mechanism, and the network policies to the interpreter for processing in OpenFlow switches. Additionally, this system eliminates the need to learn domain-specific languages in order to program network policies.

Thus, we have accomplished a framework that provides automatic generation and enforcement of both usage and network policies. It provisions cloud VMs based on the usage policies and enforces UM within the VM. Finally, ensures a balance of performance and security requirements in a VM.

Our framework as of now involves a static network and pre-programmed flows (proactive forwarding), but a change to the network requires recompilation of all of the flows. In the future, we would like the framework to incorporate a dynamic network such that only the network flows that need changes must be recompiled using techniques likes memoization and dynamic programming.

Additionally, we want the ability to add license file/policies to the header of the packets. Thus, the license would travel with the payload/data, which would ease the tracking of policies.

References

[1] H. Takabi, J. B. Joshi, and G.-J. Ahn, “Security and privacy challenges in cloud computing environments,” *IEEE Security and Privacy*, vol. 8, no. 6, pp. 24–31, 2010.

[2] Peter Mell and Tim Grance, “The NIST Definition of Cloud Computing,” http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc, 2009.

[3] S. Pearson and A. Benameur, “Privacy, security and trust issues arising from cloud computing,” in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 693–702. [Online]. Available: http://dx.doi.org/10.1109/CloudCom.2010.66

[4] S. Pearson, Y. Shen, and M. Mowbray, “A privacy manager for cloud computing,” in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom ’09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 90–106. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_9

[5] D. Zissis and D. Lekkas, “Addressing cloud computing security issues,” *Future Gener. Comput. Syst.*, vol. 28, no. 3, pp. 583–592, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.12.006

[6] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, “Security and privacy challenges in cloud computing environments,” *IEEE Security and Privacy*, vol. 8, no. 6, pp. 24–31, Nov. 2010. [Online]. Available: http://dx.doi.org/10.1109/MSP.2010.186

[7] V. Nandina, J. M. Luna, C. C. Lamb, G. L. Heileman, and C. T. Abdallah, “Provisioning security and performance optimization for dynamic cloud environments,” in *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*. IEEE, 2014, pp. 979–981.

[8] S. Jajodia, K. Kant, P. Samarati, A. Singhal, V. Swarup, and C. Wang, *Secure cloud computing*. Springer, 2014.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[10] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, “Veriflow: verifying network-wide invariants in real time,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.

[11] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker *et al.*, “Composing software defined networks.” in *NSDI*, 2013, pp. 1–13.

[12] M. Musuvathi, D. R. Engler *et al.*, “Model checking large network protocol implementations.” in *NSDI*, vol. 4. Citeseer, 2004, pp. 12–12.

[13] N. Peter Loscocco, “Integrating flexible support for security policies into the linux operating system,” in *Proceedings of the FREENIX Track:... USENIX Annual Technical Conference.* The Association, 2001, p. 29.

[14] A. Ginter, “Unidirectional security gateways stronger than firewalls.”

[15] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane: taking control of the enterprise,” in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[16] H. Kim and N. Feamster, “Improving network management with software defined networking,” *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 114–119, 2013.

[17] A. Lara, A. Kolasani, and B. Ramamurthy, “Simplifying network management using software defined networking and openflow,” in *Advanced Networks and Telecommuncations Systems (ANTS), 2012 IEEE International Conference on.* IEEE, 2012, pp. 24–29.

[18] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak, “Maple: Simplifying sdn programming using algorithmic policies,” in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 87–98.

[19] V. Nandina, J.-M. Luna, E. J. Nava, C. C. Lamb, G. L. Heileman, and C. T. Abdallah, “Policy-based security provisioning and performance control in the cloud,” in *CLOSER*, 2013, pp. 502–508.

[20] R. Nathuji, A. Kansal, and A. Ghaffarkhah, “Q-clouds: Managing performance interference effects for qos-aware clouds,” in *Proceedings of the ACM European Society in Systems Conference 2010*, Paris, France, April 2010, pp. 237–250.

[21] J. V. Vliet and F. Paganelli, *Programming Amazon EC2.* Oŕeilly Media Inc., 2011.

[22] H. Lim, S. Babu, J. Chase, and S. Parekh, “Automated control in cloud computing: Challenges and opportunities,” in *Proc. of 1st Workshop on Autom. Ctrl for Datacenters & Clouds.*, Barcelona, June 2009, pp. 13–18.

[23] J. Yao, X. Liu, X. Chen, X. Wang, and J. Li, “Online decentralized adaptive optimal controller design of cpu utilization for distributed real-time embedded systems,” in *Proceedings of the 2010 American Control Conference (ACC’10)*, Baltimore, MD, June 2010, pp. 283–288.

[24] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," *Queue*, vol. 11, no. 12, p. 20, 2013.

[25] M. Ciosi *et al.*, "Network functions virtualization," Technical report, ETSI, Darmstadt, Germany, Tech. Rep., 2012.

[26] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking.* ACM, 2013, pp. 19–24.

[27] K. M. Khan and Q. Malluhi, "Establishing trust in cloud computing," *IT professional*, vol. 12, no. 5, pp. 20–27, 2010.

[28] M. Roman and S. Khan, "Cloud computing security: A survey," *Global Journal on Technology*, 2015.

[29] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: Security challenges in virtual machine based computing environments." in *HotOS*, 2005.

[30] T. A. Limoncelli, "Openflow: a radical new idea in networking," *Queue*, vol. 10, no. 6, p. 40, 2012.

[31] N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proceedings of the first workshop on Hot topics in software defined networks.* ACM, 2012, pp. 55–60.

[32] R. W. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, "Verifiably-safe software-defined networks for cps," in *Proceedings of the 2nd ACM international conference on High confidence networked systems.* ACM, 2013, pp. 101–110.

[33] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on.* IEEE, 2012, pp. 1–6.

[34] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[35] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[36] G. Stabler, A. Rosen, S. Goasguen, and K.-C. Wang, "Elastic ip and security groups implementation using openflow," in *Proceedings of the 6th international workshop on Virtualization Technologies in Distributed Computing Date.* ACM, 2012, pp. 53–60.

[37] S. Piper, *Big Data Analytics For Dummies.* John Wiley and Sons, 2012.

[38] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks.* ACM, 2012, pp. 121–126.

[39] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow, "Blueprint for the intercloud-protocols and formats for cloud computing interoperability," in *Internet and Web Applications and Services, 2009. ICIW'09. Fourth International Conference on.* IEEE, 2009, pp. 328–336.

[40] K. Keahey, M. Tsugawa, A. Matsunaga, and J. A. Fortes, "Sky computing," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 43–51, 2009.

[41] "Openflow - Enabling Innovation in Your Network," http://www.openflow.org, November 2011.

[42] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking.* ACM, 2009, pp. 11–18.

[43] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks." in *NDSS*, 2013.

[44] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ACM SIGPLAN Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[45] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the first workshop on Hot topics in software defined networks.* ACM, 2012, pp. 43–48.

[46] A. Lara and B. Ramamurthy, "Opensec: A framework for implementing security policies using openflow," in *Global Communications Conference (GLOBECOM), 2014 IEEE.* IEEE, 2014, pp. 781–786.

[47] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 27–38.

[48] P. A. Jamkhedkar, G. L. Heileman, and C. C. Lamb, "An interoperable usage management framework," in *Proceedings of the tenth annual ACM workshop on Digital rights management.* ACM, 2010, pp. 73–88.

[49] J. Park and R. Sandhu, "The uconabc usage control model," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 1, pp. 128–174, Feb. 2004. [Online]. Available: http://doi.acm.org/10.1145/984334.984339

[50] P. A. Jamkhedkar, G. L. Heileman, and C. C. Lamb, "An interoperable usage management framework," in *Proceedings of the tenth annual ACM workshop on Digital rights management*, ser. DRM '10. New York, NY, USA: ACM, 2010, pp. 73–88. [Online]. Available: http://doi.acm.org/10.1145/1866870.1866885

[51] P. A. Jamkhedkar, C. C. Lamb, and G. L. Heileman, "Usage management in cloud computing," in *IEEE CLOUD*, 2011, pp. 525–532.

[52] R. S. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9–19, 1993.

[53] V. Nandina, J.-M. Luna, E. J. Nava, C. C. Lamb, G. L. Heileman, and C. T. Abdallah, "Policy-based security provisioning and performance control in the cloud." in *CLOSER*, 2013, pp. 502–508.

[54] J. M. Luna, "Optimization and regulation of performance for computing systems," Ph.D. dissertation, University of New Mexico, 2014.

[55] R. Tempo, G. Calafiore, and F. Dabbene, *Randomized Algorithms for Analysis and Control of Uncertain Systems*, E. D. Sontag, M. Thoma, A. Isidori, and J. V. Schippen, Eds. London: Springer, 2005.

[56] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo benchmarks: Java benchmarking development and analysis."

[57] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on.* IEEE, 2014, pp. 1–6.

[58] S. Ortiz, "Software-defined networking: On the verge of a breakthrough?" *IEEE Computer*, vol. 46, no. 7, pp. 10–12, 2013.

[59] R. Sandhu and X. Zhang, "Peer-to-peer access control architecture using trusted computing technology," in *Proceedings of the tenth ACM symposium on Access control models and technologies.* ACM, 2005, pp. 147–158.

[60] M. Zhang, M. Dusi, W. John, and C. Chen, "Analysis of udp traffic usage on internet backbone links," in *Applications and the Internet, 2009. SAINT'09. Ninth Annual International Symposium on.* IEEE, 2009, pp. 280–281.

[61] S. Pirzada, "An introduction to graph theory," *Orient BlackSwan, Hyderabad*, vol. 130, p. 131, 2012.