Sudan University of Science and Technology

College of Engineering

School of Electronics Engineering

Evaluating Dynamic Resource Management Algorithms In Cloud Systems

A Research Submitted In Partial fulfillment for the Requirements of the Degree of B.Sc. (Honors) in Electronics Engineering

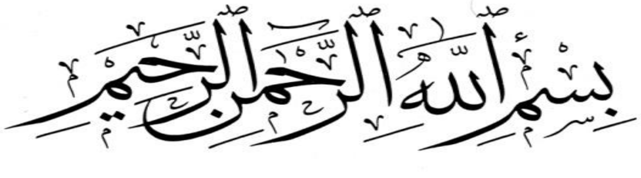
Prepared By:

1. Ahmed Yousif Farah
2. Reem Abubekr Eltayb
3. Solafa Mohamed Elhabib

Supervised By:

Dr. Sami H. O. Salih

November, 2016



قال تعالى :

صدق الله العظيم

سورة طه(114)

DEDICATION

This thesis is dedicated to our beloved mothers and fathers for planting the magic inside us and uplifting our spirit by supporting us all the way along. Also, dedicated to our brothers, sisters, friends and our supervisor for spending his time and effort to make this project on its best way.

ACKNOWLEDGMENT

All praise to Allah, today we fold the days' tiredness and the errand summing up between the cover of this humble work. To our mothers, to whom they strive to bless comfort and welfare and never stint what they own to push us in the success way who taught us to promote life stairs wisely and patiently, to our dearest fathers.

To our supervisor **Dr. Sami H. O. Salih** *who supervised, guided, encouraged and helped us wholeheartedly.*

To those who taught us letters of gold and words of jewels of the utmost and sweetest sentences in the whole knowledge. Who reworded to us their knowledge simply and from their thoughts made a lighthouse guides us through the knowledge and success path, to our honored teachers, thanks very much.

ABSTRACT

Network virtualization and network management for cloud computing systems have become quite active research areas in the last years. More recently, the advent of the Software-Defined Networks (SDNs) introduced new concepts for solving these issues, fomenting new research initiatives oriented to the development and application of SDNs in the cloud.

The goal of this research is to analyze these opportunities, showing how the SDN technology can be employed to develop, organize and virtualize cloud networking. Besides discussing the theoretical aspects related to this integration, as well as the ensuing benefits, we present a practical a case study based on the integration between Opendaylight (ODL) SDN controller and CloudStack cloud operating system.

المستخلص

أصبحت افتراضية وإدارة الشبكة لأنظمة الحوسبة السحابية من المجالات البحثية النشطة جدا في السنوات الأخيرة. وفي الآونة الأخيرة، ظهور الشبكات المعرفة من قبل البرمجيات أدخلت مفاهيم جديدة من أجل حل هذه القضايا، بالتحريض على المبادرات البحثية الجديدة الموجهة إلى تطوير وتطبيق الشبكات المعرفة من قبل البرمجيات في السحابة.

الهدف من هذا البحث هو تحليل هذه الفرص، والتي تبين كيف أن تكنولوجيا الشبكات المعرفة من قبل البرمجيات يمكن استخدامها لتطوير وتنظيم المحاكاة الافتراضية الشبكات السحابية. إلى جانب مناقشة الجوانب النظرية المتعلقة بهذا التكامل، فضلا عن الفوائد التي تلت ذلك، فإننا نقدم دراسة حالة عملية على أساس التكامل بين وحدة تحكم الشبكات المعرفة من قبل البرمجيات و نظام تشغيل الحوسبة السحابية.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

API	Application programming interface
ACS	Apache CloudStack
DHCP	Dynamic host configuration protocol
DPI	Deep Packet Inspection
ForCES	Forwarding and control element Separation
HTTP	Hybrid text transfer protocol
IAAS	Infrastructure as-a-service
IDS	Instruction Detection System
IT	Information Technology
KVM	Kernel based virtual machine
NETCONF	Network configuration
NFV	Network function virtualization
NTP	Network timing protocol
ODL	Open DayLight
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OVSDB	OpenVswitch database
PAAS	Platform as-a-service
SAAS	Software as-a-service
SAL	Service abstraction layer
SDN	Software Defined Networking

SLA	Service level agreement
SR-IOV	Single root I/O virtualization
STT	Stateless Transport Tunneling
VLAN	Virtual local area network
VXLAN	Virtual extended LAN

CHAPTER ONE

INTRODUCTION

1. INTRODUCTION

1.1 Preface

Cloud computing has ushered the information technology (IT) field and service providers into a new era, redefining how computational resources and services are delivered and consumed. With cloud computing, distinct and distributed physical resources such as computing power and storage space can be acquired and used in an on-demand basis, empowering applications with scalability and elasticity at low cost. This allows the creation of different service models, generally classified as [1]: Infrastructure-as-a-Service (IaaS), which consists in providing only fundamental computing resources such as processing, storage and networks; Platform-as-a-Service (PaaS), in which a development platform with the required tools (languages, libraries, etc.) is provided to tenants; and Software-as-a-Service (SaaS), in which the consumer simply uses the applications running on the cloud infrastructure.

Clouds aim to drive the design of the next generation data centers by architecting them as networks of virtual services (hardware, database, user-interface, application logic) so that users can access and deploy applications from anywhere in the world on demand at competitive costs depending on their QoS (Quality of Service) requirements [2]. Developers with innovative ideas for new Internet services no longer require large capital outlays in hardware to deploy their service or human expense to operate it [3].

To actually provide cost reductions, the cloud needs to take advantage of economies of scale, and one key technology for doing so is resource virtualization. After all, virtualization allows creation of a logical abstraction layer above the pool of physical resources, thereby enabling a programmatic approach to allocate resource wherever needed while hiding the complexities involved in their management [4]. The result is potentially very efficient resource utilization, better manageability, on-demand and programmatic resource instantiation, and resource isolation for better control, accounting and availability.

In any cloud environment, the network is a critical resource that connects various distributed and virtualized components, such as servers, storage elements, appliances and applications [5]. For example, it is the network that allows aggregation of physical servers, efficient virtual machine (VM) migration, and remote connection to storage systems, effectively creating the perception of large, monolithic resource pool. Furthermore, it is also the network that enables delivery of cloud based applications to end users. Yet, while every component in a cloud is getting virtualized, the physical network connecting these components is not. Without virtualization, the network is one physical common network, shared by all cloud end-users and cloud components [6].

1.2 Problem Statement

The major issues with building efficient cloud computing systems are the system flexibility and the availability, in particularly the resource management. However, utilizing complex algorithms for resource management such as hybrid resource management will negatively affect the system performance. Moreover, the optimum coefficients for resource to be used is time vary and application dependant. These tradeoffs necessitate a prototype to pre-evaluate the system optimal operation before installing live applications to the cloud. Current test-beds are either based on cloud system operating system, or just addressing the management feature of SDN.

1.3Proposed Solution

A test-bed is to be implemented to emulate the cloud computing system behavior when the SDN approach is used in resource management. In this research OpenDayLight SDN controller is employed to cloud computing systems to better utilize the resource management using Dynamic algorithm. Results show that a considerable improvement is achieved in flexibility, Aailability and system performance.

1.4 Aims and Objective

The main objective of this research is to study the Quality of the Services offered for the end users when benefiting from XaaS packages. This should be achieved by maximum utilization of the available resource and the provided side. This has been detailed as follows;

i. Studying the Cloud and the virtualization approaches
ii. Studying the SDN management approaches
iii. Develop a test-bed environment for Cloud Computing
iv. Applying SDN to manage Cloud Computing system resource based on Dynamic Resource Management algorithm
v. Validate the results

1.5Methodology

A deductive method is used in this research starting from arguing that employing SDN approaches in resource management of Cloud computing systems will improve the system flexibility and performance. Then a test-bed environment is implemented to verify this hypothesis in various scenarios using dynamic resource management.

1.6 Research Outlines

After the introductory chapter, Chapter Two will give a general overview of resource management concepts especially in cloud computing technology. Chapter Three and Chapter Four will highlight the SDN management approaches. The former, presents the methodology with details on the tools used. While the later, evaluate the results of the applied scenarios. The research conclusion and recommendation for future work has been drown in Chapter Five.

CHAPTER TWO

CLOUD COMPUTING AND SOFTWARE DEFINED NETWORKS APROACHES

2. Cloud Computing And Software Defined Networks Approaches

2.1 Hardware Resource Management

resource management is the process of allocating computing, storage, networking and (indirectly) energy resources to a set of applications, in a manner that seeks to jointly meet the performance objectives of the applications, the infrastructure (i.e., data center) providers and the users of the cloud resources [7]. The objectives of the providers center around efficient and effective resource use within the constraints of Service Level Agreements (SLAs) with the Cloud Users. Efficient resource use is typically achieved through virtualization technologies, which facilitate statistical multiplexing of resources across customers and applications. The objectives of the Cloud Users tend to focus on application performance, their availability, as well as the cost-effective scaling of available resources in line with changing application demands. Often, these objectives come with constraints regarding resource dedication to meet non-functional requirements relating to, for example, security or regulatory compliance.

2.2 Cloud Computing systems

2.2.1 Exploring the Cloud Computing Stack

Cloud computing builds on the architecture developed for staging large distributed network applications on the Internet over the last 20 years. To these standard networking protocols, cloud computing adds the advances in system virtualization that became available over the last decade [8].

The cloud creates a system where resources can be pooled and partitioned as needed. Cloud architecture can couple software running on virtualized hardware in multiple locations to provide an on-demand service to user-facing hardware and software. It is this unique combination of abstraction and metered service that separates the architectural requirements of cloud computing systems from the general description given for an n-tiered Internet application.

2.2.2 Cloud feature

The NIST Definition of Cloud Computing that classified cloud computing into the three service models (SaaS, IaaS, and PaaS) and four cloud types (public, private, community, and hybrid), also assigns five essential characteristics that cloud computing Systems must offer [9]:

- **On-demand self-service:** A client can provision computer resources without the need for interaction with cloud service provider personnel.

- **Broad network access:** Access to resources in the cloud is available over the network using standard methods in a manner that provides platform-independent access to clients of all types. This includes a mixture of heterogeneous operating systems, and thick and thin platforms such as laptops, mobile phones.

- **Resource pooling:** A cloud service provider creates resources that are pooled together in a system that supports multi-tenant usage. Physical and virtual systems are allocated or reallocated as needed. Intrinsic in this concept of pooling is the idea of abstraction that hides the location of resources such as virtual machines, processing, memory, storage, and network bandwidth and connectivity.

- **Rapid elasticity**: Resources can be rapidly and elastically provisioned. The system can add resources by either scaling up systems (more powerful computers) or scaling out systems (more computers of the same kind), and scaling may be automatic or manual. From the standpoint of the client, cloud computing resources should look limitless and can be purchased at any time and in any quantity.

- **Measured service:** The use of cloud system resources is measured, audited, and reported to the customer based on a metered system. A client can be charged based on a known metric such as amount of storage used, number of transactions, network I/O (Input/Output) or bandwidth, amount of processing power used, and so forth. A client is charged based on the level of services provided

2.2.3 Cloud Computing and Resource Virtualization

Virtualization is not a new concept in computing, having in fact appeared in the 70's [7]. The concept of virtualization has evolved with time, however, going from virtual memory to processor virtualization up to the virtualization of network resources (e.g., SDN, OpenvSwitch, etc.). With the advent of cloud computing and the demand of virtualizing entire computing environments, new virtualization techniques were developed, among them [8]:

- <u>Full Virtualization or Hardware VM</u>: all hardware resources are simulated via software.
- <u>Para-Virtualization</u>: the hardware is not simulated, but divided in different domains so they can be accessed by VMs.
- <u>Para-virtualized drivers (Para+Full Virtualization)</u>: a combination of the previous techniques

Several studies highlight the benefits of virtualization on a computing environment. Among them, the following can be cited [9]:

- <u>Resource sharing</u>**:** when a device has more resources than what can be consumed by a single entity, those resources can be shared among different users or processes for better usage efficiency [9].
- <u>Resource aggregation</u>**:** devices with a low availability of resources can be combined to create a larger-capacity virtual resource.
- <u>Ease of management</u>**:** one of the main advantages of virtualization is that it facilitates maintenance of virtual hardware resources.

- Dynamics: with the constant changes to application requirements and workloads, rapid resource reallocation or new resource provisioning becomes essential for fulfilling these new demands. Virtualization is a powerful tool for this task, since virtual resources can be easily expanded, reallocated, moved or removed without concerns about which physical resources will support the new demands.
- Isolation**:** multiple users environments may contain users that do not trust on each other. Therefore, it is essential that all users have their resources isolated from other users, even if this is done logically (i.e., in software).

Despite their benefits, there are also disadvantages of virtualized environments, such as [10]:

- Performance**:** even though there is no single method for measuring performance, it is intuitive that the extra software layer of the hypervisors leads to higher processing costs than a comparable system with no virtualization.
- Management: virtual environments abstract physical resources in software and files, so they need to be instantiated, monitored, configured and saved in an efficient and auditable manner, which is not always an easy task.
- Security**:** whereas isolation is a mandatory requirement for VMs in many real case scenarios, completely isolating a virtualized resource from another, or applications running on the physical hardware from virtualized ones, are involved (if not impossible) tasks.

2.2.4 Mechanisms for Network Virtualization

To understand the mechanisms that can implement network virtualization, first we need to understand which resources can be virtualized in a network.

- Virtualization of NICs**:** it's necessary to provide every VM with its own virtual NIC (vNIC)
- Virtualization of L2 Switches: The number of ports in a typical switch is limited. To solve this issue, IEEE Bridge Port Extension standard 802.1BR [32] proposes a virtual bridge with a large number of ports using physical or virtual port extenders (like a vSwitch).
- Virtualization of L2 Networks: In a multitenant data center, VMs in a single physical machine may belong to different clients and, thus, need to be in different virtual LANs (VLANs) [12].
- Virtualization of L3 Networks**:** When the multitenant environment is extended to a layer 3 network, there are a number of competing proposals to solve the problem. Examples include: virtual extensible LANs (VXLANs) [13]; network virtualization using generic routing encapsulation (NVGRE) [14]; and the Stateless Transport Tunneling (STT) protocol [15].
- Virtualization of L3 Router**:** Network Function Virtualization (NFV) [16] provides the conceptual framework for developing and deploying virtual L3 routers and other layer 3 network resources

2.3 Software Defined Networks

The term SDN originally appeared in [1], referring to the ability of Open-Flow [2] to support the configuration of table flows in routers and switches using software. However, the ideas behind SDNs come from the goal of having a programmable network, whose research started short after the emergence of the Internet, led mainly by the telecom industry. Today, the networking industry has shown enormous interest in the SDN paradigm, given the expectations of reducing both capital and operational costs with service providers and enterprise data centers with programmable, virtualizable and easily partitionable networks. These features of SDNs make them highly valuable for cloud computing systems, here the network infrastructure is shared by a number of independent entities and, thus, network management becomes a challenge. Indeed, while the first wave of innovation in the cloud focused on server virtualization technologies and on how to abstract computational resources such as processor, memory and storage, SDNs are today promoting a second wave with network virtualization [14]. The emergence of large SDN controllers focused on ensuring availability and scalability of virtual networking for cloud computing systems (e.g., OpenDayLight [13] and Open-Contrail [30]) is a clear indication of this synergy between both technologies

2.3.1 The Role of Software Defined Network in resource management

The software –defined networking (SDN) paradigm has emerged as a promising Solution to reduce this complexity through the creation of a unified control plane Independent of specific vendor equipment. However, designing a SDN-based solution for network resource management raises several challenges as it should Exhibit flexibility, scalability and adaptability. We will review some of the main challenges associated with SDN-based solutions and present our recent contributions in that direction. Support for both static and dynamic resource management applications.

2.3.2 Software Defined Networks Advancement

We can divide the historical advancements that culminated in the SDN concept into the three different phases [3], as follows:

1. **Active Networks** (from the mid-1990s to the early 2000s):
   This phase follows the historical advent of the Internet, The so-called "active networks" appeared as a first initiative aiming to turn network devices (e.g., switches and routers) into programmable elements and, thus, allow furthers innovations in the area. This programmability could then allow a separation between the two main functionalities of networking elements: the control plane, which refers to the device's ability to decide how each packet should be dealt with; and the data plane, which is responsible for forwarding packets at high speed following the decisions made by the control plane. Specifically, active networks introduced a new paradigm for dealing with the network's control plane, in which the resources (e.g., processing, storage, and packet queues) provided by the network elements could be accessed through application programming interfaces (APIs).

2. Control- and data-plane separation (from around 2001 to 2007):
After the Internet became a much more mature technology in the late 1990's, the increasing complexity of network topologies, together with concerns regarding the performance of backbone networks, led different hardware manufacturers to develop embedded protocols for packet forwarding, promoting the high integration between the control and data planes seen in today's Internet. The importance of a centralized control model has become more evident, as well as the need of a separation between the control and data planes. Among the technological innovations arising from this phase, we can cite the creation of open interfaces for communications between the control and data planes such as ForCES (Forwarding and Control Element Separation) [27], whose goal was to enable a locally centralized control over the hardware elements distributed along the network topology [4], [29]. To ensure the efficiency of centralized control mechanisms, the consistent replication of the control logic among the data plan elements would play a key role.

3. OpenFlow and Network Operating System (from 2007 to 2010):
The ever growing demand for open interfaces in the data plane led researchers to explore different clean slate architectures for logically centralized network control [6], [7], [8]. In particular, the Ethane project created a centralized control solution for enterprise networks, reducing switch control units to programmable flow-tables. The operational deployment of Ethane in the Stanford computer science department, focusing on network experimentation inside the campus, was indeed huge success, and resulted in the creation of OpenFlow protocol [25]. OpenFlow enables fully programmable networks by providing a standard

data plane API for existing packet switching hardware. The creation of the OpenFlow API, on its turn, allowed the emergence of SDN control platforms such as NOX [9], thus enabling the creation of a wide range of network applications. OpenFlow led to the vision of a network operating system that, different from the node-oriented system preconized by active networks, organize the network's operation into three layers: (1) a data plane with an open interface; (2) a state management layer that is responsible for maintaining a consistent view of the overall network state; and (3) control logic that performs various operations depending on its view of network state [10]. Following these advances, solutions such as Onix and its open-source counterpart, ONOS (Open Network Operating System) [31], Analyzing this historical perspective, it becomes easier to see that the SDN concept emerged as a tool for allowing further network innovation, helping researchers and network operators to solve longstanding problems in network management and also to provide new network services. SDN has been successfully explored in many different research fields, including areas such as network virtualization and cloud networking.

2.3.3 Software Defined Networks Features

SDN facilitates network virtualization and may, thus, makes it easier to implement features such as dynamic network reconfiguration (e.g., in multitenant environments). However, it is important to recognize that the basic capabilities of SDN technologies do not directly provide these benefits. Some SDN features and their main contributions to improve network virtualization are:

Control plane and data plane separation: The separation between control and data planes in SDN architectures, as well as the standardization of interfaces for the communication between those layers, allowed to conceptually unify different vendor network devices under the same control mechanisms.

Network programmability: The programming capabilities introduced by SDN provide the dynamics necessary to rapidly scale, maintain and configure new virtual networks. Moreover, network programmability also allows the creation of custom network applications oriented to innovative network virtualization solutions.

Logically centralized control: The abstraction of data plane devices provided by SDN architecture gives the network operating system, also known as SDN orchestration system, a unified view of the network. Therefore, it allows custom control applications to access the entire network topology from a logically centralized control platform, enabling the centralization of configurations and policy management.

Automated management: the SDN architecture enhances network virtualization platforms by providing support for automation of administrative tasks. The centralized control and the programming capabilities provided by SDN allow the development of customized network applications for virtual network creation and management. Auto scaling, traffic control and QoS are examples of automation tools that can be applied to virtual network environments

2.3.4 Data plane and control plane

SDN architecture and its main components, showing that the data and control planes are connected via a well-defined programming interface between the switches and the SDN controller.

The data plane: corresponds to the switching circuitry that interconnects all devices composing the network infrastructure, together with a set of rules that define which actions should be taken as soon as a packet arrives at one of the device's ports.

The control plane: on its turn, is responsible for programming and managing the data plane, controlling how the routing logic should work. This is done by one or more software controllers, whose main task is to set the routing rules to be followed by each forwarding device through standardized interfaces, called the southbound interfaces. These interfaces can be implemented using protocols such as OpenFlow 1.0 and 1.3 [13], OVSDB [14] and NETCONF [15] The control plane concentrates, thus, the intelligence of the network, using information provided by the forwarding elements (e.g., traffic statistics and packet headers) to decide which actions should be taken by them [16].

2.3.5 The OpenFlow Protocol

The OpenFlows protocol is one of the most commonly used southbound interfaces, being widely supported both in software and hardware, and standardized by the Open Networking Foundation (ONF). It works with the concept of flows, defined as groups of packets matching a specific header [25], which receive may be treated differently depending how the network is programmed. OpenFlow’s simplicity and flexibility, allied to the high performance at low cost, ability to isolate experimental traffic from production traffic, and to cope with vendors’ need for closed platforms [25], are probably among the main reasons for this success.

2.3.6 OpenDaylight Controller

Created in April 2013 as a Linux Foundation collaborative project, OpenDaylight is an open source OpenFlow controller and also a scalable SDN framework for the development of several network services, including data plane protocols. As such, OpenDaylight can be the core component of any SDN architecture. The OpenDaylight architecture follows the traditional SDN design, implementing the control layer as well as the northbound and southbound interfaces.

CHAPTER THREE

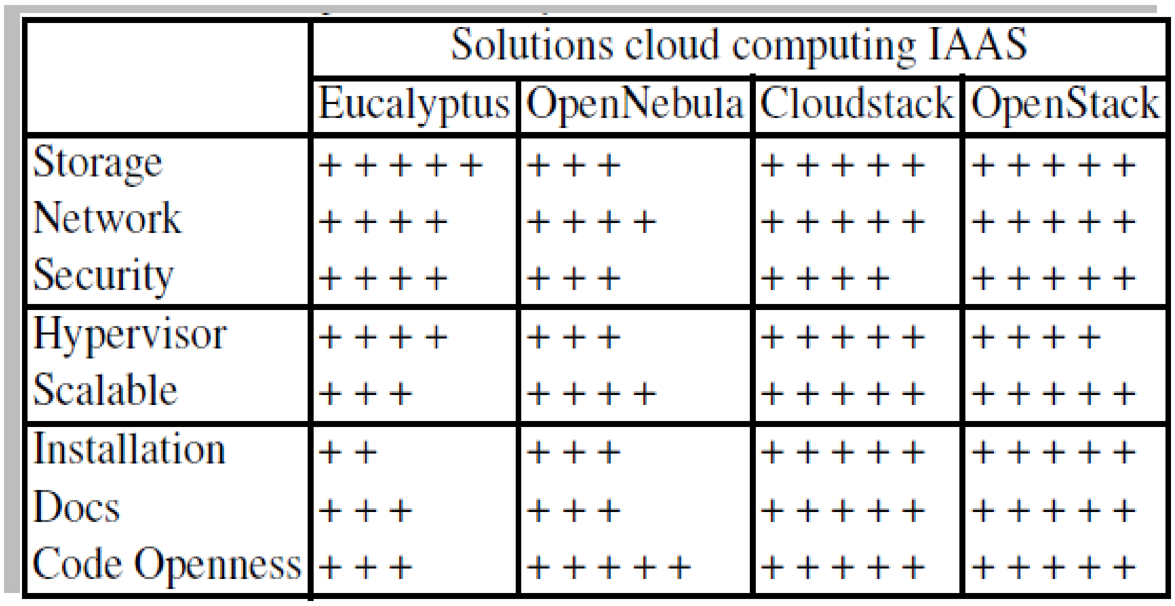
EMPLOYING SOFTWARE DEFINED NETWORKS APPROACHES IN CLOUDSTACK

3. Employing Software Defined Nnetworks Approaches In CloudStack

3.1 Cloudstack cloud operating system

CloudStack is an open source IaaS Cloud platform originally developed by Cloud.com. CloudStack implements the Amazon EC2 and S3 APIs, as well as the vCloud API, in addition to its own API CloudStack, written in Java, is designed to manage and deploy large networks of VMs. Cloud-Stack currently supports VMware, Oracle VM, KVM, Xen Server and Xen Cloud Platform. CloudStack has a hierarchical structure, which enables management of multiple physical hosts from a single web-based interface. Table 3-1 and table 3-2 shows comparison between different open source cloud operating system.

Table 3-1: Comparison between Different Open Source Cloud Operating System [10]



	Solutions cloud computing IAAS			
	Eucalyptus	OpenNebula	Cloudstack	OpenStack
Storage	+ + + + +	+ + +	+ + + + +	+ + + + +
Network	+ + + +	+ + + +	+ + + + +	+ + + + +
Security	+ + + +	+ + +	+ + + +	+ + + + +
Hypervisor	+ + + +	+ + +	+ + + + +	+ + + +
Scalable	+ + +	+ + + +	+ + + + +	+ + + + +
Installation	+ +	+ + +	+ + + + +	+ + + + +
Docs	+ + +	+ + +	+ + + + +	+ + + + +
Code Openness	+ + +	+ + + + +	+ + + + +	+ + + + +

Table 3-2: Comparison between OpenStack and CloudStack Cloud Operating System [10]

	OpenStack	CloudStack
Language	Python, Shell script	Java, Python, Shell script
Lines of code	210,051	1,270,052
Database tables	83	141
Hypervisor support	KVM, XenServer, Hyper-V, Vmware	KVM, XenServer, Oracle VM, Hyper-V, Vmware
Monitoring and billing	No	Monitoring(no), billing (yes)
Control	Basic	Advanced
Live migration support	Poor	Good
High availability	Basic	Advanced
Password encryption	No	Yes
Message passing	RabbitMQ(AMQP)	Java
Documentation	HTML, PDF	PDF

3.2 OpenDayLight controller

OpenDaylight is a Java-based SDN controller built to provide a comprehensive network programmability platform for SDN. OpenDaylight is little different from the others because it allows for other non-OpenFlow southbound protocols. Table 3-3 shows comparison between the main characteristics of open source SDN controllers.

Table 3-3: Comparison between the Main Characteristics of Open Source SDN Controllers [30]

	NOX	POX	Ryu	Floodlight	ODL
Language	C++	Python	Python	Java	Java
Performance	High	Low	Low	High	High
Distributed	No	No	Yes	Yes	Yes
OpenFlow	1.0	1.0	1.2–1.4	1.0, 1.3	1- 1.3
Multi-tenant clouds	No	No	Yes	Yes	Yes
Learning curves	Moderate	Easy	Moderate	Steep	Steep

3.3 CloudStack Deployment

CloudStack cloud operating system consist of four components. Management server, one or more storage servers, and host server.

3.3.1 Management server

Management server responsible of all cloud control including set zones, pods, cluster, storage servers, and host servers. Besides managing all traffic of the cloud, and connecting the cloud components to each other. It has a web-based interface to make the managing process much easier. Figure 3-1shows conceptual view of basic deployment.

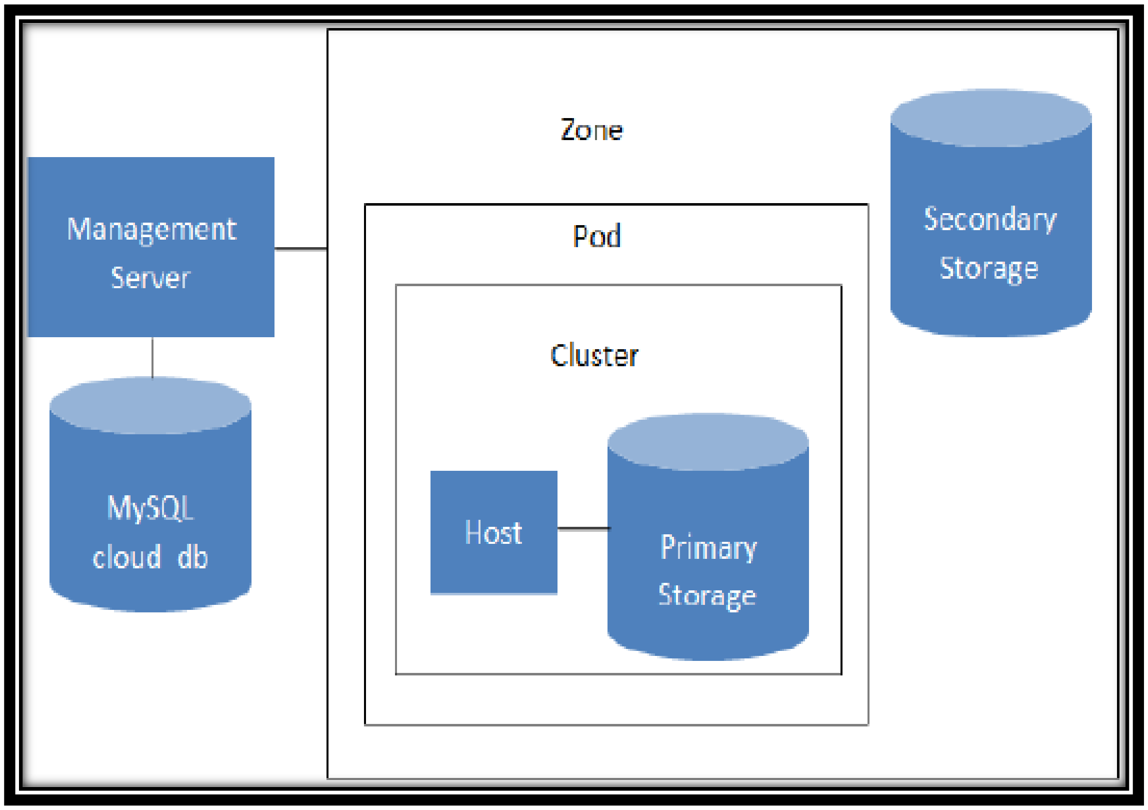


Figure 3-1: Conceptual View Of Basic Deployment [23]

3.3.2 Storage servers

Instead of saving data in the host servers that are offered to tenants, in CloudStack the tenant data are saved in separated server. This will result in more reliability, and availability, and make a simple way to live migration. CloudStack at least must have a primary Storage server. And for more availability one or more secondary storage server.

If the primary storage went down, a secondary storage automatically goes up and receive all tenant traffic.

3.3.3 Host servers

The host servers are the actual infrastructure that offered to tenants. One host server may guest more than a tenant, and each tenant will have one or more virtual machine. CloudStack Cloud operating system offers iaas (infrastructure as a service).

3.4 OpenDayLight Deployment

The OpenDaylight architecture follows the traditional SDN design, implementing the control layer as well as the northbound and southbound interfaces. However, differently from the majority of controllers, the OpenDaylight architecture clearly separates its design and implementation aspects. The OpenDaylight SDN controller is composed by the following architectural layers [22]:

Network Applications, Orchestration and Services: Business applications that make use of the network services provided by the controller platform to implement control, orchestration and management applications.

Controller Platform: Control layer that provides interfaces for all the network services implemented by the platform via a REST northbound API. The controller platform also implements a service abstraction layer (SAL), which provides a high-level view of the data plane protocols to facilitate the development of control plane applications.

Southbound Interfaces and Protocol Plugins: Southbound interfaces contain the plugins that implement the protocols used for programming the data plane.

Data Plane Elements: Physical and virtual network devices that compose the data plane and are programmed via the southbound protocol plugins. The variety of southbound protocols supported by the OpenDaylight controller allows the deployment of network devices from different vendors in the underlying network infrastructure.

The service abstraction layer (SAL) is one of the main innovations of the Open-Daylight architecture to enable communication between plugins, this message exchange mechanism ignores the role of southbound and northbound plugins and builds upon the definition of Consumer and Provider plugins. Providers are plugins that expose features to applications and other plugins through its northbound API, whereas consumers are components that make use of the features provided by one or more Providers. This change implies that every plugin inside OpenDaylight can be seen as both a provider and a consumer, depending only on the messaging flow between the plugins involved. In OpenDaylight, SAL is responsible for managing the messaging between all the applications and underlying plugins. Figure 3-2 shows communication between producer and consumer plugins using SAL

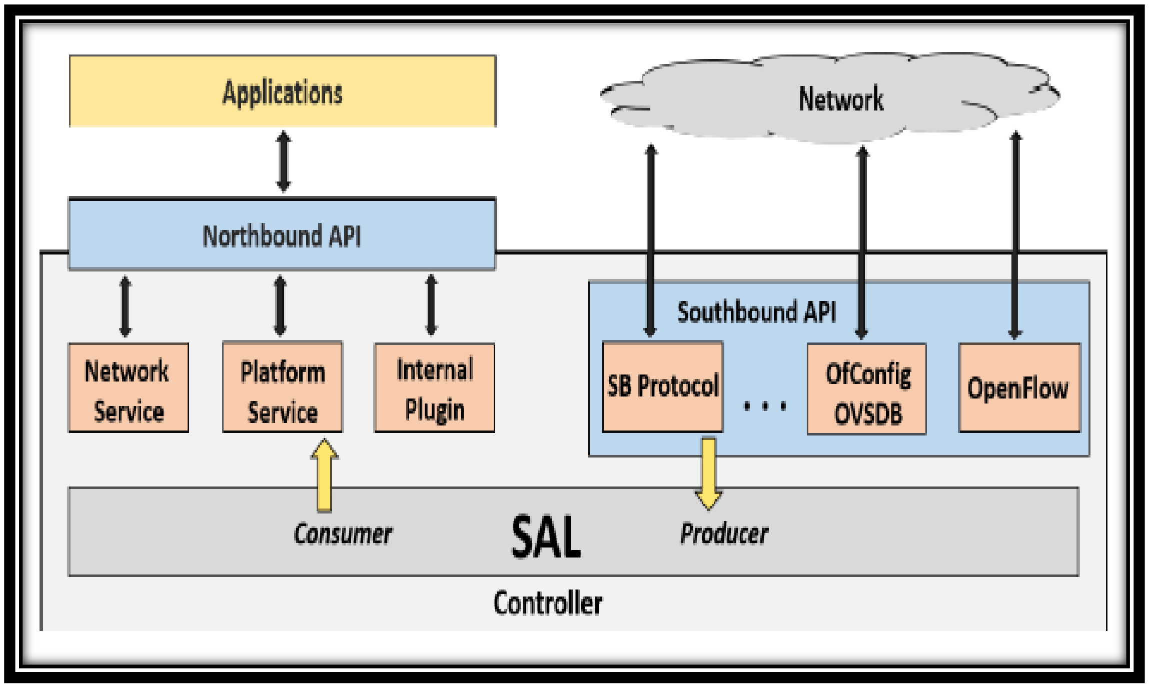


Figure 3-2: Communication Between Producer And Consumer Plugins Using SAL [22]

In OpenDaylight, SAL is responsible for managing the messaging between all the applications and underlying plugins. Figure 3-3 shows the life of a package inside the OpenDaylight architecture, depicting the following steps [22]:

1. A packet arriving at Switch1 is sent to the appropriate protocol plugin.
2. The plugin parses the packet and generates an event for SAL.
3. SAL dispatches the packet to the service plugins listening for DataPacket.
4. Module handles the packet and sends is out via the IDataPacketService.
5. SAL dispatches the packet to the southbound plugins listening for DataPacket.
6. OpenFlow message sent to appropriate switch.

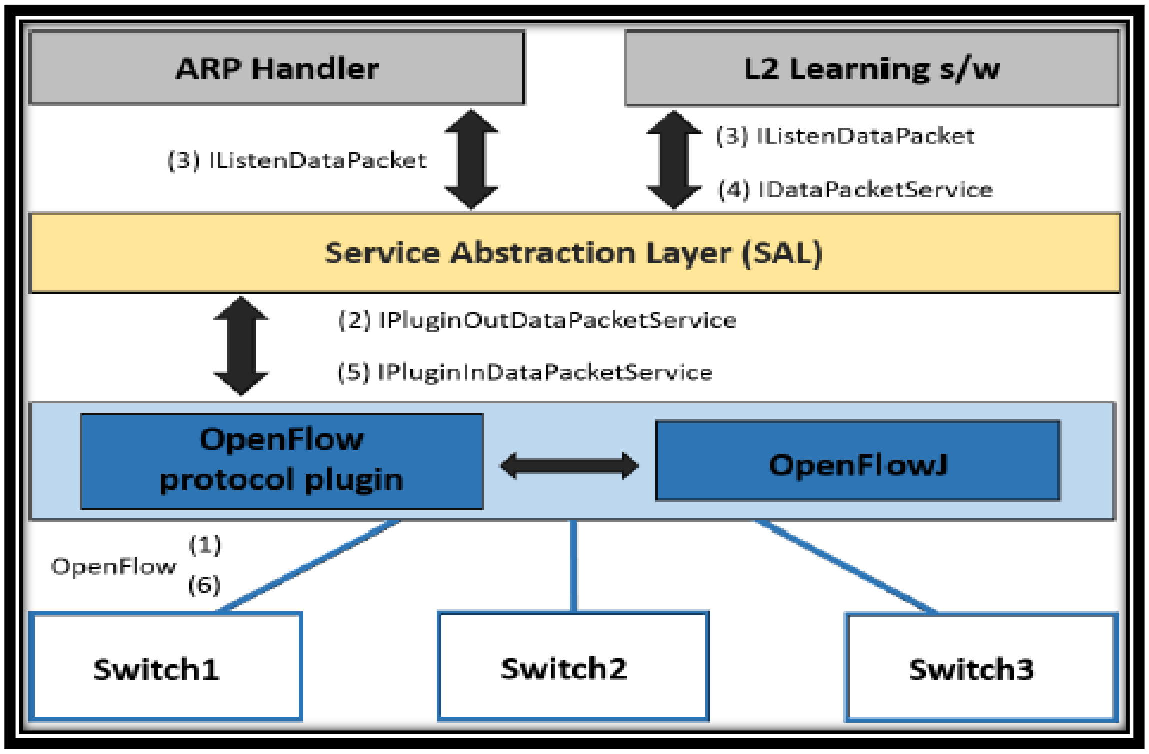


Figure 3-3: Life of a Package In OpenDaylight. [22]

3.5 Integration Architectures

SDNs and clouds display similar designs, with a 3-layer architecture composed by a Infrastructure Layer with computational resources controlled by a Control Layer, which in turn is controlled via APIs by applications in an Application Layer (see Figure 3-4). One simple form of integrating SDNs and clouds is to run their stacks in parallel, with both technologies being integrated by the applications themselves. Even though applications can benefit from both technologies with this strategy, it also brings a significant overhead to application developers. After all, applications would need to be SDN-aware and cloud-aware, and accessing APIs for both technologies in an effective manner, which is prone to complicate their design and implementation.

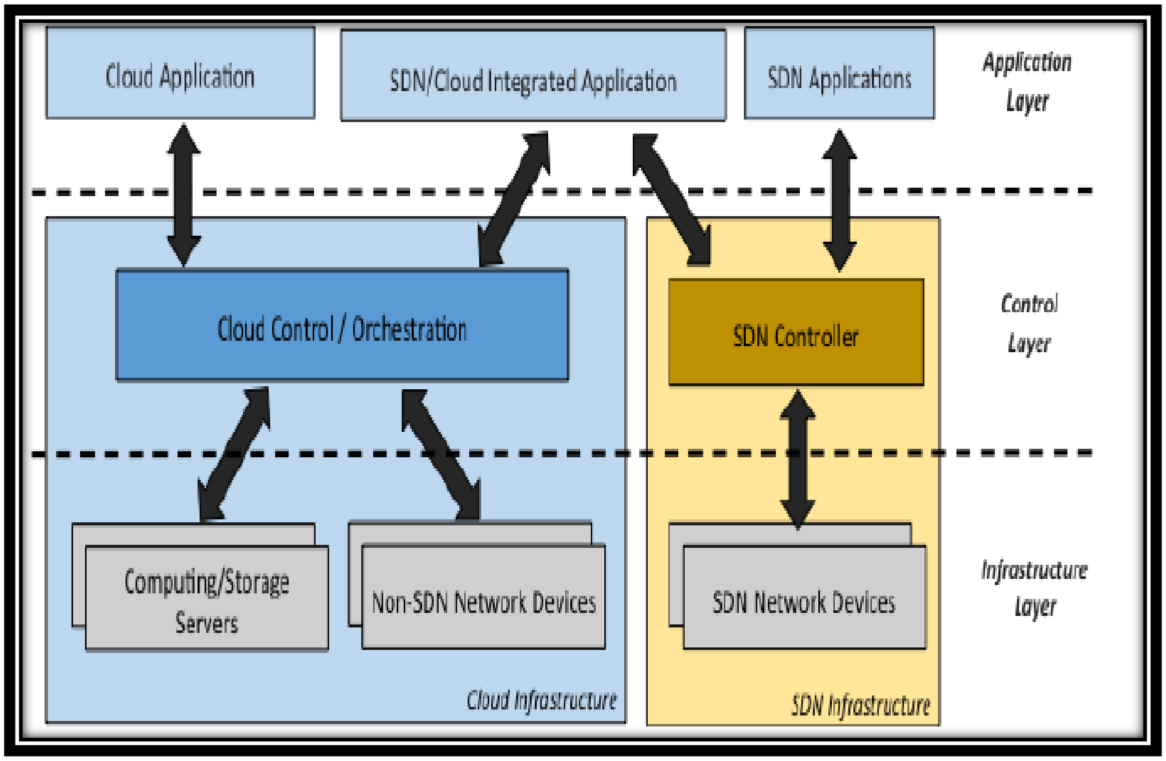


Figure 3-4: SDN Function Incorporated In Cloud Control [19]

A second and probably preferable is to consider the cloud control/orchestration system as an SDN application to the SDN controller. In this scenario, depicted in Figure (3-5) the Cloud Control/Orchestration subsystem is augmented with modules that translate Cloud Operations to SDN operations, using existing SDN controllers APIs. This approach brings the benefits of the previous approach while allowing greater flexibility. It is possible to evolve both the Cloud and SDN infrastructures separately, with minimal or no changes to their integration interface. It would also allow the use of existing SDN solutions without alterations, including proprietary SDN solutions or hardware-based controllers.

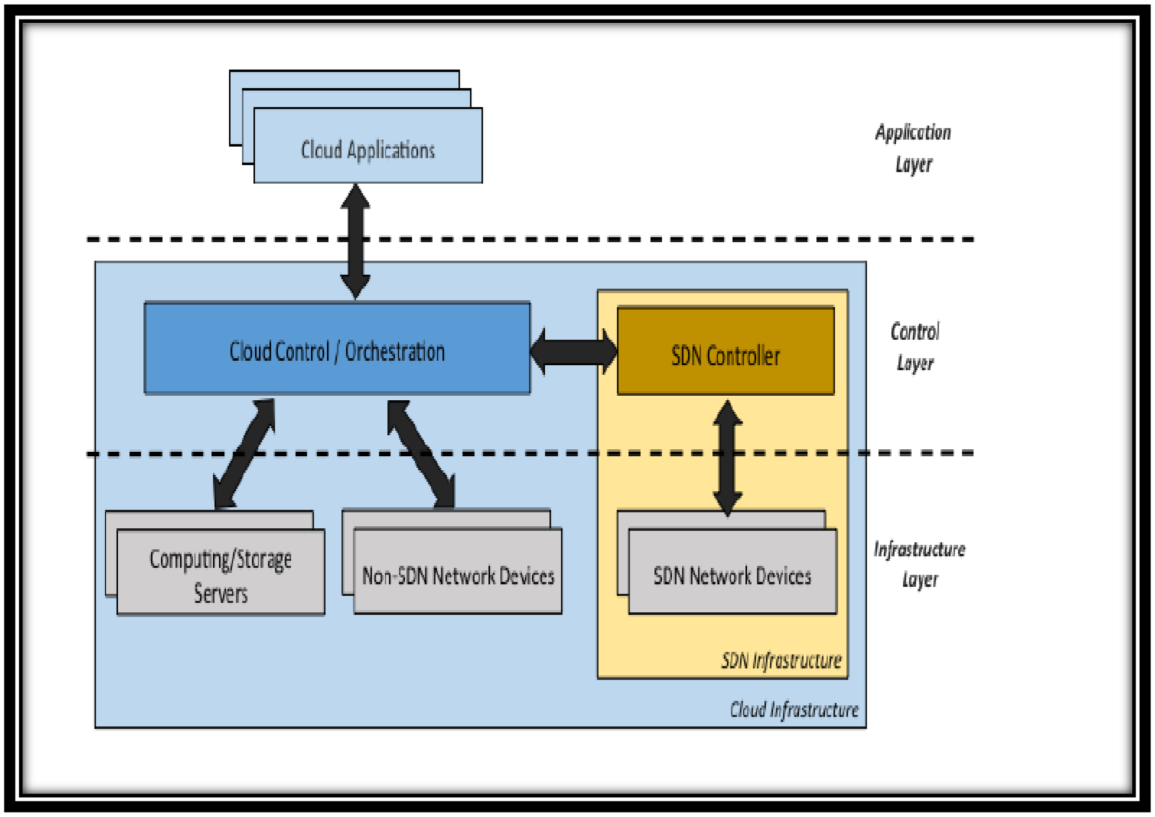


Figure 3-5: SDN Function Incorporated In Cloud Control Orchestration Subsystem [19]

3.6 Deploying Cloud Networking

In what follows, we conduct a simple experiment showing how to build an architecture that bring the benefits of SDN to cloud computing systems. For this, we use CloudStack and OpenDaylight, giving a practical example of the CloudStack networking in an SDN architecture, analyzing the interactions between CloudStack and the OpenDaylight controller, as well as their specific roles in this deployment.

3.6.1 Experiments

In this Section three practical Scenarios are introduced to show the benefits of SDN to cloud computing systems. The first one is CloudStack host with HTTP, And DHCP services based on Static resource allocation. The Second is CloudStack host with HTTP, And DHCP services based on Dynamic resource allocation. . The third is CloudStack host with HTTP, And DHCP services based on Hybrid resource allocation.

Static resource allocation practical example

In this example the resources of the host server will be statically allocated (fixed amount of resources). Then HTTP, and DHCP services will be setup on the host server.

Dynamic resource allocation practical example

In this example the resources of the host server will be dynamically allocated (A pool of shared resources). Then HTTP, and DHCP services will be setup on the host server.

Hybrid resource allocation practical example

In this example the resources of the host server will be dynamically allocated (minimum fixed amount of resources, and a pool of shared resources). Then HTTP, and DHCP services will be setup on the host server.

CHAPTER FOUR

Deploying Cloud Networking with CloudStack and OpenDaylight

4. Deploying Cloud Networking with CloudStack and OpenDaylight

This project has been done on actual hardware resources completely. No simulations. Actual devices are involved in this evaluation. So the outcome results are more practical rather than theoretical. All command lines related to this chapter are included in Appendix A.

The test environment include three laptop are used as hardware resources, and tow VMware virtual machines are in each of them, one for the cloud management, one for the cloud primary storage, one for the cloud secondary storage, and three for the hosts servers. All VMware virtual machine operating on Centos 6.3 operating system.

The first laptop contain the management server and a host server, and the second contain the primary storage server and a host server, and the third contain the secondary storage server and a host server. Connected to each other via switch. Figure (4-1) cloud infrastructure.

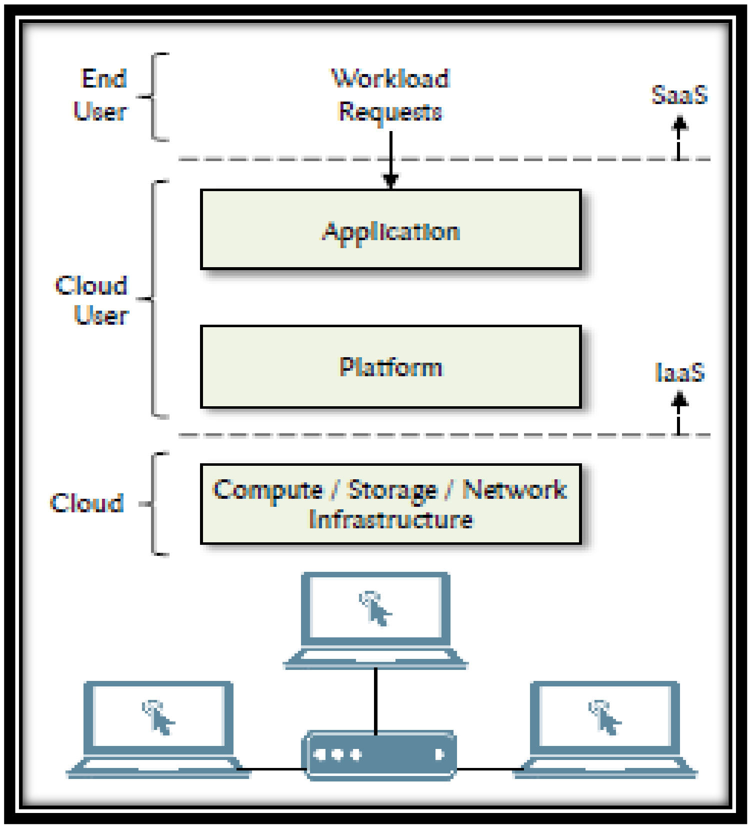


Figure 4-1: Cloud Infrastructure

4.1 CloudStack deployment

CloudStack cloud operating system consist of four components.

Next how to setup and configure all of four components.

4.1.1 Hardware and software requirements for CloudStack

Management Server, Database, and Storage System Requirements

The machines that will run the Management Server and MySQL database must meet the following requirements.

The same machines can also be used to provide primary and secondary storage, such as via localdisk or NFS. The Management Server may be placed on a virtual machine.

• Operating system:

– Preferred: CentOS/RHEL 6.3+ or Ubuntu 12.04(.1)

• 64-bit x86 CPU (more cores results in better performance)

• 4 GB of memory

• 250 GB of local disk (more results in better capability; 500 GB recommended)

• At least 1 NIC

• Statically allocated IP address

• Fully qualified domain name as returned by the hostname command

Host/Hypervisor System Requirements

The host is where the cloud services run in the form of guest virtual machines. Each host is one machine that meets the following requirements:

• Must support HVM (Intel-VT or AMD-V enabled).

• 64-bit x86 CPU (more cores results in better performance)

• Hardware virtualization support required

• 4 GB of memory

• 36 GB of local disk

• At least 1 NIC

• Latest hotfixes applied to hypervisor software

• When you deploy CloudStack, the hypervisor host must not have any VMs already running

• All hosts within a cluster must be homogeneous. The CPUs must be of the same type, count, and feature flags.

4.1.2 Cloudstack management server

Setup Management server:

- ➢ Configure the interface
- ➢ Change the hostname
- ➢ Add the CloudStack repository.

Repository content:

```
[cloudstack]
name=cloudstack
baseurl=http://cloudstack.apt-get.eu/centos/6/4.6/
enabled=1
gpgcheck=0
```

- ➢ Install the next packages:
  1. Network Timing Protocol

     NTP is required to synchronize the clocks of the servers in the cloud. Unsynchronized clocks can cause unexpected problems.
  2. MYSQL-SERVER
  3. CLOUDSTACK-MANAGEMENT

- ➢ Stop iptables
- ➢ Configure Selinux

  Change in /etc/selinux/config the Selinux configuration from enforce to Permissive

CloudStack does various things which can be blocked by security mechanisms like SELinux. It has to be disabled to ensure the Agent has all the required permissions.

- Secure the installation of database
- Set up the database
- After the database is set up, finish configuring the OS for the Management Server.

4.1.3 CloudStack Storage

Cloudstack storage includes primary storage server and one or more secondary storage server.

Primary Storage

CloudStack is designed to work with a wide variety of commodity and enterprise-rated storage systems.

Setup Primary Storage server:

- configure the interface
- Change the host name
- Install the NFS package
- Stop iptables

Secondary storage

CloudStack is designed to work with any scalable secondary storage system. The only requirement is that the secondary storage system supports the NFS protocol.

Setup Secondary Storage Server:

- ➢ configure the interface
- ➢ Change the host name,
- ➢ Install the NFS package
- ➢ Stop iptables

4.1.4 CloudStack Host servers

Setup host server:

- ➢ configure the interface
- ➢ Change the host name
- ➢ Install next packages:

1- Network Timing Protocol

NTP is required to synchronize the clocks of the servers in the cloud. Unsynchronized clocks can cause unexpected problems.

2- CLOUDSTACK-AGENT

3- LIBVIRT

CloudStack uses libvirt for managing virtual machines.

- ➢ Stop iptables
- ➢ Configure Selinux

Change Selinux configuration to Permissive in /etc/selinux

CloudStack does various things which can be blocked by security mechanisms like AppArmor or SELinux. These have to be disabled to ensure the Agent has all the required permissions.
Centos 6.3 comes with selinux, and it’s set to enforce by default.

- ➢ Connect the components with each other and managing the cloud from the management web-based user interface.

4.2 OpenDayLight controller

Setup OpenDayLight controller:

- ➢ Install ODL package in the management server.
- ➢ Run the OpenDaylight SDN controller, which will be used by ACS to dynamically program the created virtual network bridges. The SDN controller receives REST requests from the ODL driver, which implements the methods called by plugins for implementing the layer 2 network services provided by ACS API. To run OpenDaylight, the commands in Appendix A should be executed on the Network and Controller node.

4.3 The Integration

For the integration between ACS & ODL the second method is used (CloudStack cloud operating system controller deals directly with OpenDayLight controller).

- ➢ Link the ODL with ACS.

Now that we have the OpenDaylight SDN controller running, we can start the CloudStack services on the in the management server. These nodes will then make use of these software resources to create the entire virtual network infrastructure.

4.4 Practical Scenarios

In what follows, we conduct a simple examples showing the benefits of SDN to cloud computing systems. For this, we use CloudStack and OpenDaylight, giving a practical example of the CloudStack hosts in an SDN architecture.

Host based on static resource allocation

In this host server the resources are statically allocated (fixed amount of resources). The host server resources are:

1. One CPU cores.
2. 512MB of RAM.
3. 10GB of hard disk.

The host server is running CenOS6.3 operating system, and should be running HTTP, and DHCP services.

Host based on dynamic resource allocation

In this host server the resources are dynamically allocated (A pool of shared resources). The shared pool resources are:

1. 3 CPU cores.
2. 1.5GB of RAM.
3. 30GB of hard disk.

The host server is running CenOS6.3 operating system, and should be running HTTP, and DHCP services.

Host based on Hybrid resource allocation

In this host server the resources are dynamically allocated (fixed minimum amount of resources, and a pool of shared resources). The minimum fixed resources are:

1. One CPU cores.
2. 512MB of RAM.
3. 10GB of hard disk.

The shared pool resources are:

4. 3 CPU cores.
5. 1.5GB of RAM.
6. 30GB of hard disk.

The host server is running CenOS6.3 operating system, and should be running HTTP, and DHCP services.

HTTP service setup

- Install httpd packages
- Check configuration file

DHCP service setup

- Install dhcpd packages
- Check configuration file
- Check next files existence in /etc/dhcp directory
    1. dhcpd.conf
    2. dhcpd.leases
    3. dhclient/dhclient.conf
- Configure rsyslog service
- Run DHCP service

4.5 Comparison

To show the benefits of SDN to cloud computing systems a comparison between the three hosts is necessary. The comparison depends on the performance of the hosts. To find witch host has a better performance, the throughput, latency, and CPU utilization are measured.

Testing throughput

Using Nload to test the throughput. Nload is a command line tool that allows users to monitor the incoming and outgoing traffic separately. It also draws out a graph to indicate the same, the scale of which can be adjusted. Easy and simple to use, and does not support many options. It's used when a quick look at the total bandwidth usage without details of individual processes is needed.

Testing latency

Using ping to test the latency. Ping localhost then compare by pinging all other computers on the network from the same host.

Testing CPU utilization

Using Iostat to test the CPU utilization. The iostat command (provided via the sysstat package on CentOS) provides three reports: CPU utilization, device utilization, and network file system utilization. If you run the command without options it will display all three reports, specify the individual reports with the (-c, -d and -h) switches respectively.

4.6 Results

After the three practical scenarios have been completed and the performance metrics comparison figures have been plotted, some results are achieved. These are:

4.6.1 Throughput Results

The throughput results obtained using the NLoad commandtool. The results were obtained under the same circuimstances in all hosts. Then the results were ploted using matlab. Figure 4-2 depicts that the throughput improved and became better (increased) in the case of Hybrid and the dynamic hosts due to deploying ODL controller in the cloud operating system. As observed the result showed that the host with Hybrid resource management has the highest throughput. Figure 4-2 shows the throughput for each case individully:

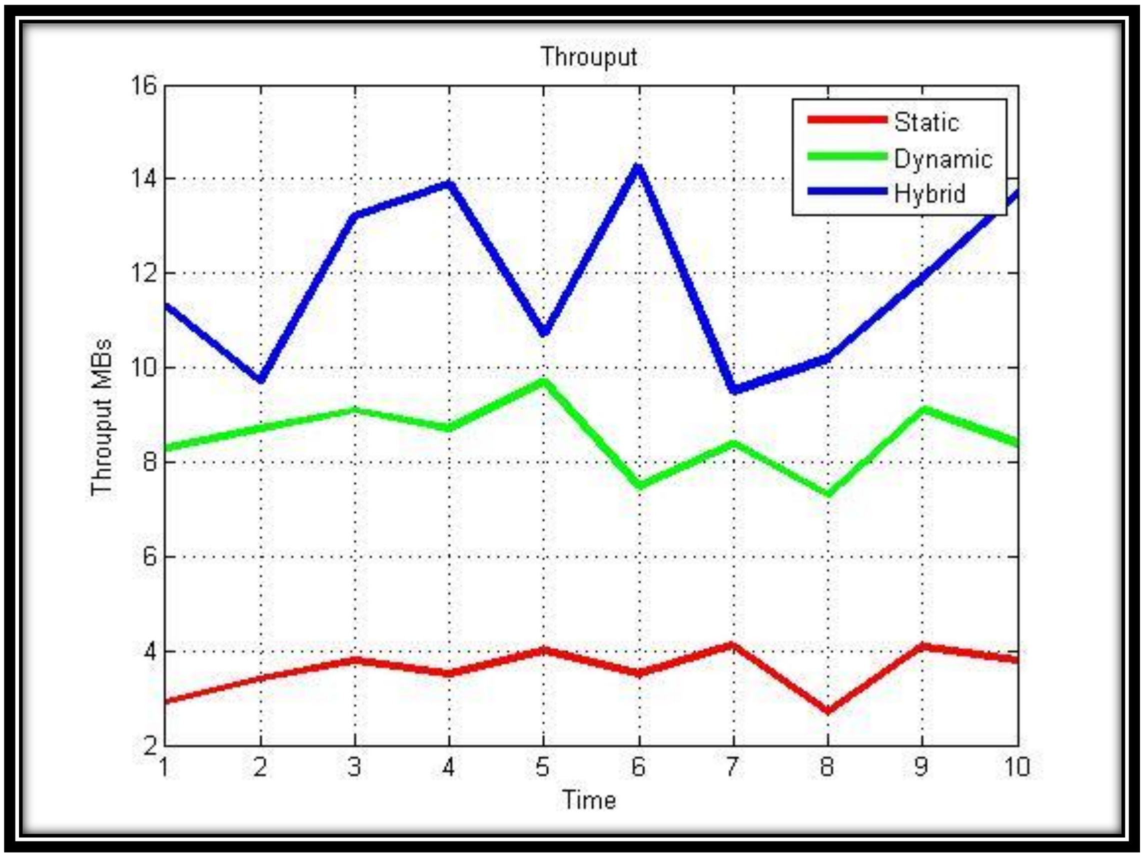


Figure 4-2: Throughput Results

4.6.2 CPU Utilization Results

The CPU utilization results results obtained using the Iostat commandtool. The results were obtained under the same circuimstances in all hosts. Then the results were ploted using matlab. Figure 4-3 shows that the CPU utilization have been improved in the cases of Hybrid and the dynamic hosts due to deplying ODL controller in the cloud operating system As observed the result showed that the host with Hybrid resource management has the best CPU utilization. Figure 4-3 shows the CPU utilization for each case individully:

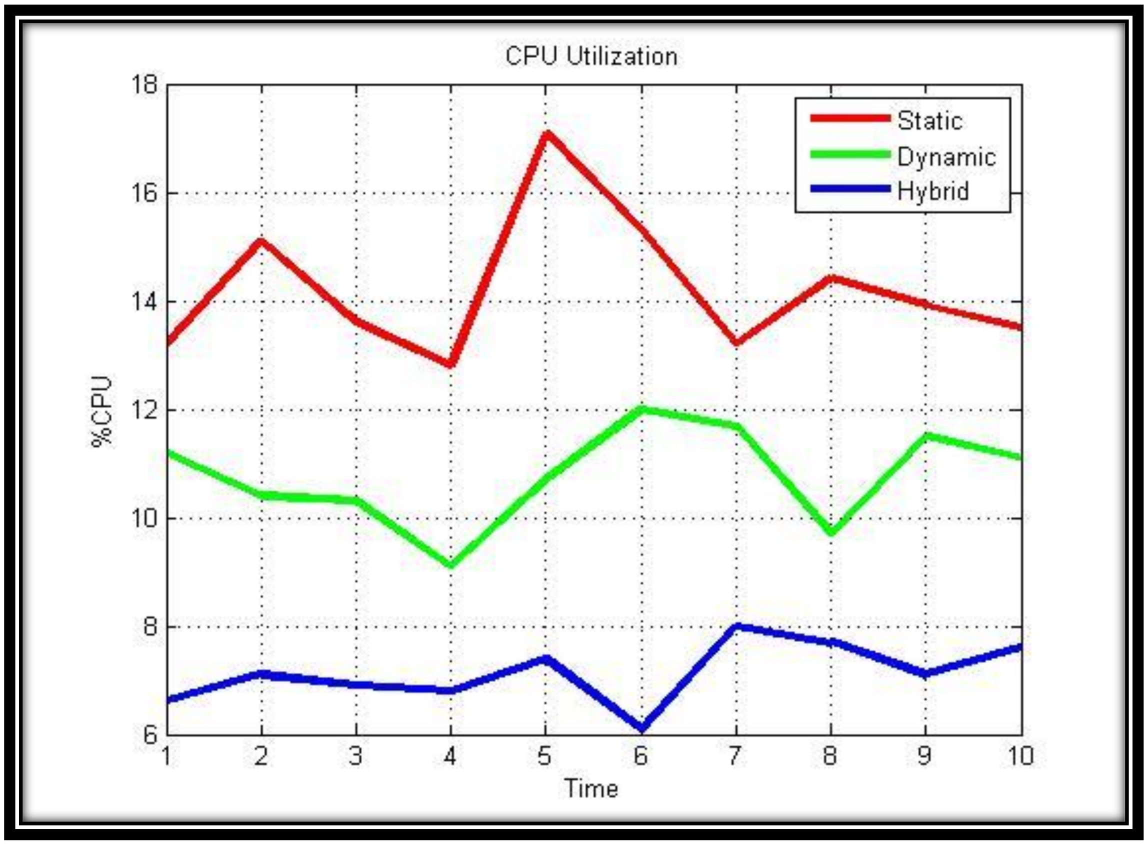


Figure 4-3: CPU Utilization Results

4.6.3 Latency Results

The latency results results obtained using the ping command. The results were obtained under the same circuimstances in all hosts. Then the results were ploted using matlab. Figure 4-4 shows that the latency have been improved (decreased) in the cases of Hybrid and the dynamic hosts due to deploying ODL controller in the cloud operating system. As observed the result showed that the host with Hybrid resource management has the lowest latency. Figure 4-4 shows the CPU utilization for each case individully:

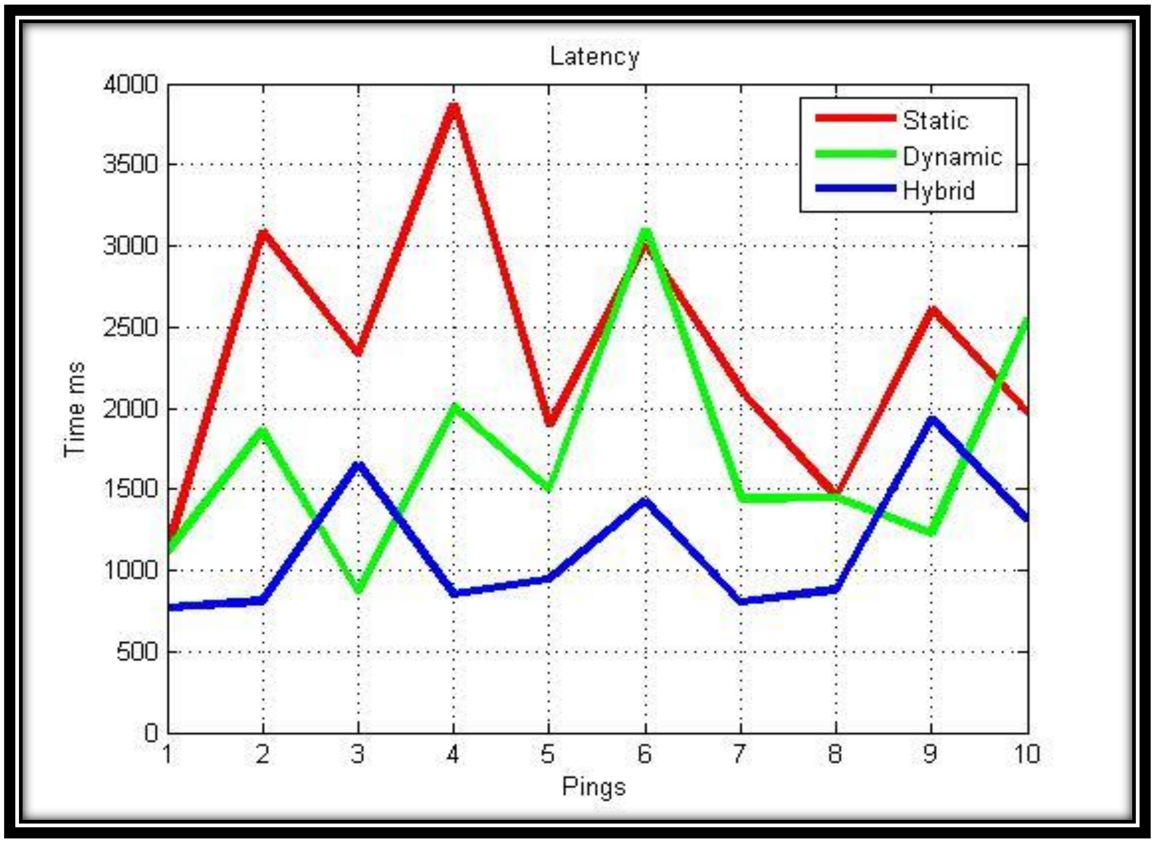


Figure 4-4: Latency Results

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION FOR FUTURE WORK

5. Conclusion And Recommendation For Future Work

5.1 Conclusion

In this project we introduced the role of network virtualization on implementing and delivering cloud computing services. We presented the available network virtualization mechanisms and describe how they can address the cloud networking requirements. The advent of SDN introduced new forms of approaching network virtualization and network control inside cloud. Cloud systems presents some challenges, such as, efficient resource management, fast provisioning and scalability. By separating a network's control logic from the underlying infrastructure, software defined networking (SDN) promises an unprecedented simplification in network programmability. SDN improved cloud network control and accelerated the provision of innovative network services. To illustrate the feasibility of integrating both cloud computing and SDN paradigms, we presented a practical study of the deployment of CloudStack and OpenDaylight. The practical study included three practical examples, the study were mentioned to show the benefits of SDN to cloud computing systems. The three hosts were compared with each other in manner of performance aspect. The comparison depends on the throughput, CPU utilization, and the latency of the hosts. The result showed that the host with Hybrid resource management has the highest throughput, and the best CPU utilization, and has the lowest latency. Then comes the host with the Dynamic resource management. And the host with the static resource allocation appeared to have the lowest throughput, and the worst CPU utilization, and has the highest latency.

5.2 Recommendations for Future Work

After we finished this project, we can provide some recommendations and research issues for who wants to carry on from the point we stopped on. Future work on this topic can include:

Insufficient bandwidth can cause significant latency on the interaction between users and the application, reducing the quality of the service (QoS) provided to and by cloud tenants. The emergence of control models such as SDN, together with hardware-assisted virtualization technologies such as SR-IOV, is expected to improve the control capacity over the shared network resources.

SDN and NFV appliances can also address the challenges on policy enforcement complexity. These policies define the configuration of each virtual and physical resource in the cloud network. Traffic isolation and access control to end users are among the multiple forwarding policies that can be enforced by deploying SDN and NFV solutions in the cloud.

REFERENCES

[1] Autenrieth, A., Elbers, J.-P., Kaczmarek, P., and Kostecki, P. (2013). Cloud orchestration with SDN/OpenFlow in carrier transport networks. In 15th Int. Conf. on Transparent Optical Networks (ICTON), pages 1–4.

[2] Yang, L., Dantu, R., Anderson, T., and Gopal, R. (2004). RFC 3746 – forwarding and control element separation (ForCES) framework. https://tools. ietf.org/html/rfc3746.

[3] OpenContrail (2014). OpenContrail: An open-source network virtualization platform for the cloud. http://www.opencontrail.org/. Accessed: 2015-02-28.

[4] Feamster, N., Rexford, J., and Zegura, E. (2013). The road to SDN. Queue, 11(12):20–40.

[5] Caesar, M., Caldwell, D., Feamster, N., Rexford, J., Shaikh, A., and van der Merwe, J. (2005). Design and implementation of a routing control platform. In Proc. of the 2nd Symposium on Networked Systems Design & Implementation, volume 2, pages 15–28. USENIX Association.

[6] Casado, M., Freedman, M., Pettit, J., Luo, J., McKeown, N., and Shenker, S. (2007). Ethane: Taking control of the enterprise. In Proc. of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'07), pages 1–12, New York, NY, USA. ACM.

[7] Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4d approach to network control and management. ACM SIGCOMM Computer Communication Review, 35(5):41–54.

[8] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., and Bowman, M. (2003). PlanetLab: an overlay testbed for broad-coverage services. ACM IGCOMM Computer Communication Review, 33(3):3–12.

[9] Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST).

[10] Kreutz, D., Ramos, F. M. V., Veríssimo, P., Rothenberg, C. E. Azodolmolky, S. and Uhlig, S. (2014). Software-defined networking: A comprehensive survey. CoRR, abs/1406.0440.

[11] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., and Parulkar, G. (2014). ONOS: towards an open, distributed SDN OS. In Proc. of the 3rd Workshop on Hot topics in software defined networking, pages 1–6. ACM.

[12] Ricci, R., and Seskar, I. (2014). GENI: A federated testbed for innovative network experiments. Computer Networks, 61:5–23

[13] OpenFlow (2012). Specification, openflow switch – v1.3.0

[14] Pfaff, B. and Davie, B. (2013). The Open vSwitch Database Management Protocol. RFC Editor.

[15] Enns, R., Bjorklund, M., Schoenwaelder, J., and Bierman, A. (2011). RFC 6241 – network configuration protocol (NETCONF). https://tools.ietf.org/html/rfc6241.

[16] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., amanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. (2010). Onix: A distributed control platform for large-scale production networks. In OSDI, volume 10, pages 1–6.

[17] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. ACM SIGCOMM Computer Communication Review, 38(3):105–110.

[18] Koponen, T., Amidon, K., Balland, P., Casado, M., Chanda, A. Fulton, B. Ganichev, I., Gross, J., Gude, N., Ingram, P., et al. (2014). Network virtualization in multi-tenant datacenters. In USENIX NSDI.

[19] Jain, R. and Paul, S. (2013b). Network virtualization and software defined networking for cloud computing: a survey. Communications Magazine, IEEE, 51(11):24–31.

[20] Amazon (2014). Virtualization Types – Amazon Elastic Compute Cloud.http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/virtualization_types.html. Accessed: 2016-04-27.

[21] Menascé, D. A. (2005). Virtualization: Concepts, applications, and performance modeling. In CMG Conference, pages 407–414.

[22] Al-Shaer, E. and Al-Haj, S. (2010). FlowChecker: Configuration analysis and verification of federated Openflow infrastructures. In Proc. of the 3rd ACM Workshop on Assurable and Usable Security Configuration (SafeConfig'10), pages 37–44, New York, NY, USA. ACM.

[23] CloudStack Installation Documentation, Release 4.6.0. Apache Software Foundation, November 17, 2015

[24] Greene, K. (2009). TR10: Software-defined networking. Technology Review (MIT).

[25] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2):69–74.

[26] Kim, D., Gil, J.-M., Wang, G., and Kim, S.-H. (2013). Integrated sdn and non-sdn network management approaches for future internet environment. In Multimedia and Ubiquitous Engineering, pages 529–536. Springer.

[27] Pan, J., Paul, S., and Jain, R. (2011). A survey of the research on future internet architectures. Communications Magazine, IEEE, 49(7):26–36.

[28] Lin, Y., Pitt, D., Hausheer, D., Johnson, E., and Lin, Y. (2014). Software-defined networking: Standardization for cloud computing's second wave. Computer, 47(11):19–21.

[29] Medved, J., Varga, R., Tkacik, A., and Gray, K. (2014). Open-Daylight: Towards a Model-Driven SDN Controller architecture. In 2014 IEEE 15th International Symposium on, pages 1–6. IEEE.

[30] OpenContrail (2014). OpenContrail: An open-source network virtualization platform for the cloud. http://www.opencontrail.org/. Accessed: 2016-04-27.

[31] IEEE (2012a). 802.1BR-2012 – IEEE standard for local and metropolitan area networks–virtual bridged local area networks–bridge port extension. Technical report, IEEE Computer Society.

[32] IEEE (2012b). IEEE standard for local and metropolitan area networks–media access control (MAC) bridges and virtual bridged local area networks–amendment 21: Edge virtual bridging. IEEE Std 802.1Qbg-2012, pages 1–191.

Appendix

Appendix A

Setup Management server

- Add the CloudStack repository.

```
#vi /etc/yum.repos.d/cloudstack.repo
```

- Install the next packages:

1- Network Timing Protocol

```
# yum install ntp -y
```

2- MYSQL-SERVER

```
# yum install mysql-server -y
```

3- CLOUDSTACK-MANAGEMENT

```
# yum install cloudstack-management -y
```

- Stop iptables

```
# /etc/init.d/iptables stop
# chkconfig iptables off
```

- Configure Selinux

  Change in /etc/selinux/config the Selinux configuration from enforce to Permissive

```
# setenforce permissive
```

- Secure the installation of database

```
# mysql_secure_installation
```

- Set up the database

```
# cloudstack-setup-databases cloud:<dbpassword>@localhost
--deploy-as=root:<password>
```

- After the database is set up, finish configuring the OS for the Management Server.

```
#cloudstack-setup-management
```

This command will set up iptables, and start the Management Server.

CloudStack Storage

Setup Primary Storage server

- Install the NFS package

```
# yum install nfs -y
```

➢ Stop iptables

```
#/etc/init.d/iptables stop
#chkconfig iptables off
```

Secondary storage

Setup Secondary Storage Server

➢ Install the NFS package

```
# yum install nfs -y
```

➢ Stop iptables

```
# /etc/init.d/iptables stop
#chkconfig iptables off
```

CloudStack Host server

Setup host server:

➢ Install next packages:

1- Network Timing Protocol

```
# yum install ntp -y
```

2- CLOUDSTACK-AGENT

```
# yum install cloudstack-agent –y
```

3- LIBVIRT

```
# yum install libvirt –y
```

➢ Stop iptables

```
#/etc/init.d/iptables stop
#chkconfig iptables off
```

➢ Configure Selinux

Change Selinux configuration to Permissive in /etc/selinux

```
#setenforce permissive
```

OpenDayLight controller

Setup OpenDayLight controller:

➢ Install ODL package in the management server.

```
#yum install odl -y
```

➢ Run the OpenDaylight SDN controller

```
# cd  /odl/opendaylight/
# /run.sh  -XX:MaxPermSize=384m  -virt ovsdb  -of13
```

HTTP service setup

➢ Install httpd packages

```
#Yum install http –y
```

- Check configuration file

```
#vi /etc/http/httpd.conf
```

DHCP service setup

- Install dhcpd packages

```
#Yum install dhcp -y
```

- Check configuration file

```
  #vi /etc/dhcp/dhcpd.conf
```

- Check next files existence in /etc/dhcp directory

  1- dhcpd.conf

  2- dhcpd.leases

  3- dhclient/dhclient.conf

- Configure rsyslog service

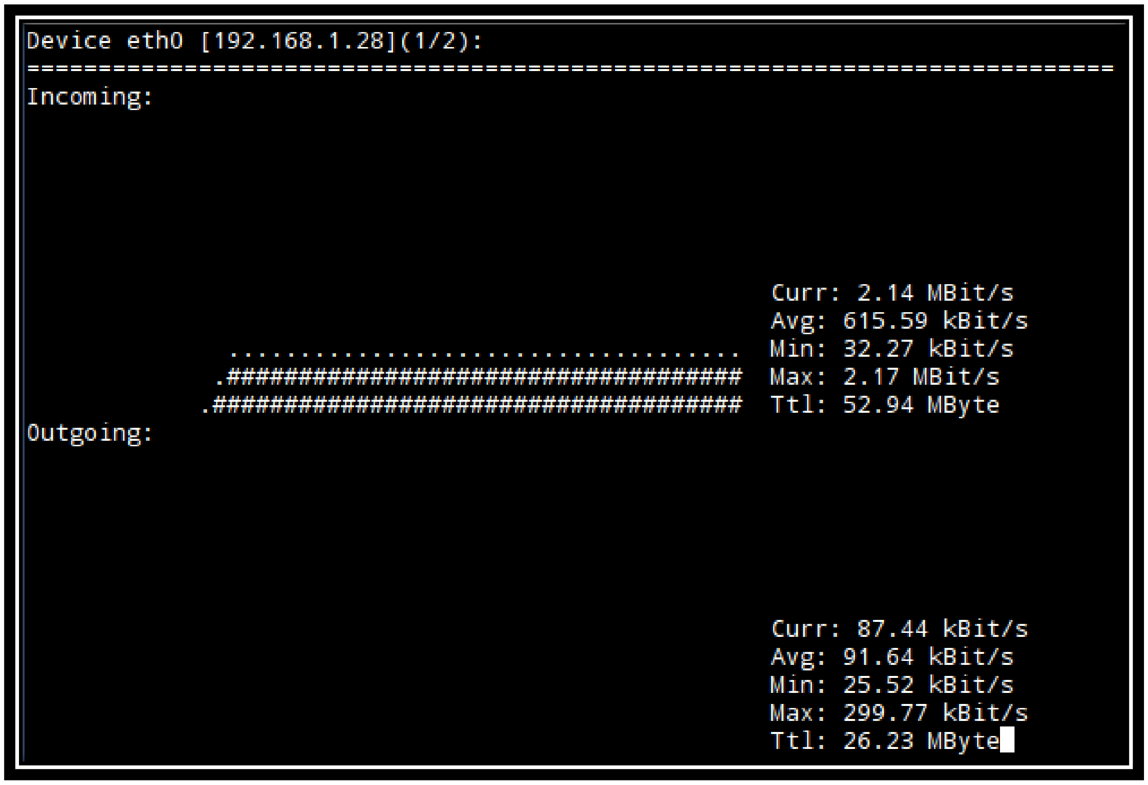
```
#vi /etc/rsyslog/rsyslog.conf
#service rsyslog restart
```

- Run DHCP service

```
#service dhcpd start
```

Testing Throughput

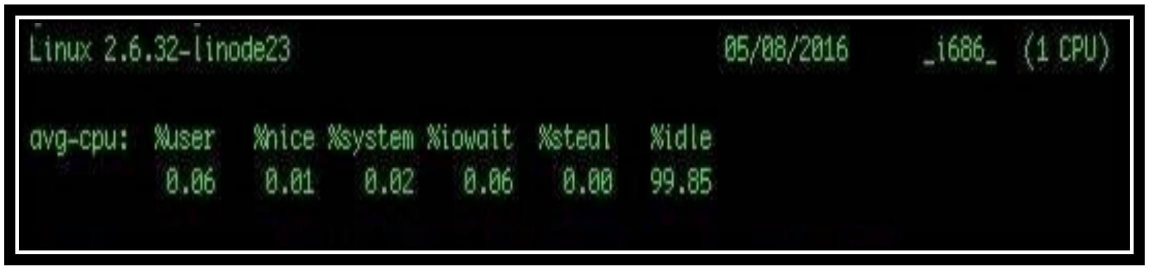
Using Nload to test the throughput.



```
Device eth0 [192.168.1.28](1/2):
================================================================================
Incoming:








                                                                      Curr: 2.14 MBit/s
                                                                      Avg: 615.59 kBit/s
              ....................................                    Min: 32.27 kBit/s
             .######################################                  Max: 2.17 MBit/s
            .#######################################                  Ttl: 52.94 MByte
Outgoing:







                                                                      Curr: 87.44 kBit/s
                                                                      Avg: 91.64 kBit/s
                                                                      Min: 25.52 kBit/s
                                                                      Max: 299.77 kBit/s
                                                                      Ttl: 26.23 MByte
```

Testing CPU Utilization

Using Iostat to test the CPU utilization.



```
Linux 2.6.32-linode23                          05/08/2016      _i686_  (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.06    0.01    0.02    0.06    0.00   99.85
```