Network Resource Management in Infrastructure-as-a-Service Clouds

Heli Amarasinghe

A Thesis submitted

in partial fulfillment of the requirements for the

Doctor of Philosophy Degree in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science

Faculty of Engineering

University of Ottawa

Abstract

Cloud Infrastructure-as-a-Service (IaaS) is a form of utility computing which has emerged with the recent innovations in the service computing and data communication technologies. Regardless of the fact that IaaS is attractive for application service providers, satisfying user requests while ensuring cloud operational objectives is a complicated task that raises several resource management challenges. Among these challenges, limited controllability over network services delivered to cloud consumers is prominent in single datacenter cloud environments. In addition, the lack of seamless service migration and optimization, poor infrastructure utilization, and unavailability of efficient fault tolerant techniques are noteworthy challenges in geographically distributed datacenter clouds.

Initially in this thesis, a datacenter resource management framework is presented to address the challenge of limited controllability over cloud network traffic. The proposed framework integrates network virtualization functionalities offered by software defined networking (SDN) into cloud ecosystem. To provide rich traffic control features to IaaS consumers, control plane virtualization capabilities offered by SDN have been employed. Secondly, a quality of service (QoS) aware seamless service migration and optimization framework has been proposed in the context of geo-distributed datacenters. Focus has been given to a mobile end-user scenario where frequent cloud service migrations are required to mitigate QoS violations. Finally, an SDN-based dynamic fault restoration scheme and a shared backup-based fault protection scheme have been proposed. The fault restoration has been achieved by introducing QoS-aware reactive and shared risk link group-aware proactive path computation algorithms. Shared backup protection has been achieved by optimizing virtual and backup link embedding through a novel integer linear programming approach. The proposed solutions significantly improve bandwidth utilization in inter-datacenter networks while recovering from substrate link failures.

Acknowledgements

I must first express my deepest gratitude to my supervisor, Professor Ahmed Karmouch, for giving me the opportunity to pursue my Ph.D. studies along with unconditional support and guidance throughout its course. I am honored to have learned from his experience and deep understanding in research. This thesis would have not been possible without his continuous trust, support, patience and encouragement. I would also like to thank Dr. Jarray Abdallah for productive collaboration and discussions that refined my knowledge and research skills. Also, I greatly appreciate his support in proofreading this dissertation and giving valuable suggestions and comments.

I wish to express my sincere gratitude to members of my Ph.D. committee, Dr. Shikharesh Majumdar, Dr. Nancy Samaan, Dr. Tet Yeap and Dr. Abdelhakim Hafid, who have regularly reviewed and constructively commented on the progress of my doctoral study. Their valuable feedback at different stages of my Ph.D. study greatly improved the quality of my thesis.

I would also like to thank Yousif Al Ridhawi, Imad Abdeljaouad, Ismaeel Al Ridhawi, Bassem Wanis, Ouassim Karrakchou, Wijaya Ekanayeka, Venkatraman Balasubramanian, Faisal Zaman, Jay Kumar Bhuchhada and other members of the Intelligence for Mobile Autonomic and Cognitive Networks (IMAGINE) Laboratory for their help and support during my research. Their companionship and friendship eased many tasks during times of difficulty. I am also thankful to Dr. Sanjeewa Herath for being an inspirational friend who continuously strengthened and helped me to stay focused. Moreover, I want to thank my parents and two brothers, Nethu and Dinu, who encouraged me with so much love and support over the years. I probably owe them much more than I think. Finally, I would like to thank my beloved wife Apsara, my daughter Ayushi and my son Denith for their unconditional love, sacrifice and encouragement.

Dedicated to my parents, wife Apsara and kids Ayushi & Denith

Table of Contents

List Of Acronyms

Acronym	Expansion
ARP	Address Resolution Protocol
API	Application Programming Interface
CPU	Central Processing Unit
CRM	Cloud Resource Manager
CSP	Cloud Service Provider
DC	Datacenter
DPID	Data Path ID
GUI	Graphical User Interface
IaaS	Infrastructure-as-a-Service
INDL	Infrastructure Description Language
ILP	Integer Linear Programming
IP	Internet Protocol
IT	Information Technology
ITRC	IT Resource Controller
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
MAC	Medium access control
MCC	Mobile Cloud Computing
MCSC	Mobile Cloud Service Controller
MEC	Mobile Edge Cloud

Acronym	Expansion
MIP	Mobile IP
MTTR	Mean Time To Repair/Recover
NIC	Network Interface Card
NRC	Netwok Resource Controller
NV	Network Virtualization
OVS	Open Virtual Switch
OWL	Web Ontology Language
PR	Physical Resources
PIP	Physical Information Pool
QoS	Quality of Service
RARP	Reverse Address Resolution Protocol
RDFS	Resource Description Framework Schema
RDC	Regional Datacenter
RIC	RDC IaaS Controller
RSM	Regional access and cloud Service Manager
SDN	Software Defined Networking
SLA	Service Level Agreement
TCP/IP	Transmission Control Protocol/Internet Protocol
VLAN	Virtual LAN
VM	Virtual Machine
VN	Virtual Network
VPN	Virtual Private Network
VR	Virtual Resources
WAN	Wide Area Network

List of Figures

List of Tables

1 Introduction

1.1 Overview

The internet is one of the most influential innovations in the late 20th century. Its current popularity can be primarily attributed to low latency broadband network connectivity coupled with convenient access technologies such as wireless networks (LTE, WiFi, 5G etc.) and smart mobile devices. In addition, the growth of the internet has created a fertile ground for new research and innovations that has led to a number of revolutionary technological services including: social networking, electronic commerce (e-commerce), mobile applications, ultra-high definition streaming (UHD), massively multiplayer online games, and virtual/augmented reality. Growing demand for these services has created a technological pull for more flexible and affordable computing infrastructure, that delivers high quality of service with on-demand rapid deployment. This pull has catalyzed the birth of the cloud service paradigm which is capable of harnessing the massive computing capacities available in geographically distributed datacenters.

As a technology, cloud computing has introduced a new era of utility computing by promising flexible resource provisioning, scalable deployment, as well as reduced capital and operational expenditure (CAPEX and OPEX). This emergence of cloud paradigm and its wide adoption reflects the increasing demand and growth of computing, in general. A number of industrial and business communities [1, 2] have predicted that this growth will last for a few more years before reaching its plateau. According to Gartner's report published in September 2017 [2], the worldwide public cloud services revenue reached

260.2 billion US dollars by the end 2017 and is predicted to further grow to 411.4 billion by the end of 2020. However, in contrast to similar revolutionary technologies such as television and the internet, the success of cloud computing cannot be attributed to a single ground-breaking technology or innovation. At its core, it has merely created a paradigm shift by introducing the notion of utility computing, that bounds some existing technologies in a more economical and user friendly manner. Among these technologies, virtualization can be regarded as the single most influential technology that catalyzed this paradigm shift.

Owing to its unique characteristics, cloud computing has been widely regarded as a successful technology capable of taking on the burden for business of managing and operating computing infrastructure. According to the National Institute of Standards and Technology (NIST), cloud computing is identified by five essential characteristics as follows: (1) on-demand self-service, (2) broad network access, (3) resource pooling, (4) rapid elasticity, and (5) measured service. These characteristics are particularly attractive for small and medium-scale businesses as they are not required to invest upfront in specialized computing equipment, infrastructure facilities and expertise upfront. As a result, cloud computing allows small and medium-scale businesses to cater to global markets by flexibly leasing computing infrastructure services as a pay-as-you-go utility. Since highly complex servers and networking equipment are managed and operated by experts in cloud infrastructure companies, businesses can focus on the core operations that directly affect their profits and productivity. On the other hand, computing infrastructure companies also benefit as they can take advantage of economies of scale through providing computing services to multiple businesses.

Even though the concentration of computing infrastructure resources in large datacenters allows infrastructure operators to reap benefits of economies of scale, it increases the complexity in cloud resource management. These complexities further intensify as cloud resource management frameworks are expected to satisfy essential cloud service characteristics, while guaranteeing higher levels of quality of service (QoS). These re-

quirements demand highly efficient and complex resource management capabilities.

This thesis addresses common IaaS resource management challenges that can be seen in large centralized datacenter cloud environments and mobile cloud environments. This chapter briefly describes the importance of resource management in cloud environments along with resource management functions, objectives and challenges. As well the role of virtualization technologies and advantages of novel Software Defined Networking (SDN) paradigm in overcoming cloud network resource management challenges is detailed. In addition, the motivations, objectives and contributions of this thesis are outlined. Finally, the organization of the remainder of the dissertation is presented.

1.2 Resource Management in Clouds

Cloud services are commonly classified into software-as-a-service (SaaS), platform-as-a-service (PaaS) and infrastructure-as-a-service (IaaS). Cloud resource management is the process of managing computing, storage and networking resource to efficiently deliver cloud services to satisfy business objectives. Since SaaS and PaaS are rely on IaaS, having a strong and powerful IaaS resource management framework allows Cloud Service Providers (CSPs) to deliver services effectively [3, 4]. Hence in this work, a focus has been given to manage cloud resources efficiently to deliver IaaS, while satisfying business objectives of CSPs.

In addition to managing compute, storage and network resources, the cloud management platform needs to improve the efficiency of resources including: energy, physical space, fault tolerance and human involvement. These tasks are necessary because they directly affect CAPEX and OPEX, as well as the natural environment. As a result, the major contributing factor to CSPs profits is the ability of the cloud management platform to perform tasks such as: dynamically power-up/down servers, automate service delivery, service chaining, use better virtualization to maximize physical space utilization, and apply efficient fault tolerance.

1.2.1 Resource Management Functions

Resource management involves three functions, namely: resource discovery, selection/allocation and operation. The cloud architecture allows CSPs to deliver cloud services through pooling resources from multiple infrastructure providers (e.g. datacenter owners). Hence, resource discovery is the process of CSP determining resources provided by different infrastructure providers in order to satisfy client requests. Also in a multi-provider federated cloud environment, resource discovery is how CSPs expose their services and resources to other CSPs in order to automate service provisioning [5]. Resource selection is the process of finding a set of resource configurations that satisfy user requirements while at the same time improving infrastructure utilization in order to serve a maximum number of clients with available resources. After selecting, resource allocation reserves a chosen resource configuration for a given request. Resource operation represents a broad range of functions performed by the cloud resource manager to meet operational objectives, after allocating specific resource configuration to client's request. These functions generally serve to guarantee service level agreements (SLAs) while improving CSP's profits.

Cloud service requirements are typically defined in SLAs that act as binding contracts between CSPs and clients. SLAs may specify management and administration policies such as billing and pricing models, QoS requirements, penalties for violations and termination conditions. Hence cloud resource operation involves conducting functionalities such as metering, monitoring and fault tolerance to ensure that SLA specifications are being met [6]. In addition to satisfying SLAs, there are several resource management objectives that affect CSP profits.

1.2.2 Resource Management Objectives

Since cloud echo-systems are driven by profit motives, operational decisions are typically multi-objective in nature. Some of these objectives can include:

- Satisfy user SLA

- Maximize resource utilization
- Apply efficient and cost effective fault tolerance
- Increase request acceptance ratio
- Provide better interoperability between heterogeneous resources
- Guarantee higher QoS levels
- Minimize service delivery time
- Reduce human involvement

Cloud resource management frameworks are designed to achieve these objectives while successfully overcoming challenges associated with heterogeneous infrastructure and dynamic user demands.

1.2.3 Resource Management Challenges

Cloud infrastructure, regardless of whether hosted in a single datacenter or a set of geographically distributed datacenters, is a complicated distributed system composed of multiple heterogeneous compute and network resources. These heterogeneous infrastructure resources are expected to handle highly unpredictable client requests, in addition to external events such as natural disasters and failures, that are beyond the control of the user and the system administrator. Hence, as the central coordinating entity in a cloud framework, the resource manager plays a critical role in determining the functionality, QoS, performance and associated costs of the cloud services. Furthermore, the cloud resource manager is required to make complex decisions according to SLAs, operational policies and multi-objective optimization algorithms. Thus resource management is considered to be one of the most important and challenging functionalities in centralized and geographically distributed cloud environments.

Due to its complex nature, there are several challenges associated with cloud resource management, corresponding to resource discovery, allocation and operation functions. There exists a number of prominent research works that further elaborate challenges in

Table 1.1: Challenges Associated With Cloud Resource Management

Functionality	Major Challenges
Resource Discovery	Inefficient resource modeling and description
	Limited resource disclosure between CSPs due to competition
	Lack of abstractions to handle heterogeneity
	Poor interoperability between vendor specific technologies
	Lack of real-time updates on resource availability
Resource Selection, Allocation and Elasticity	Unpredictability in workload characteristics
	Complexities in network resource sharing and middle boxes
	Vendor specific configuration interfaces
	Highly complex and CPU intensive resource allocation algorithms
	High cost of maintaining backups and fault tolerant schemes
	Lack of support for rapid elasticity
Monitoring and Operation	Handling the trade-off between accuracy and cost in monitoring schemes
	Insufficient control over virtual networks
	Limited support for virtual machine/network migration within and between datacenters
	Resource re-allocation and re-optimization, Handling user movements
	Load balancing and hot-spot avoidance
	Maintaining high QoS
	Guaranteeing privacy, security and resource isolation
	Fast failure detection, isolation and restoration

cloud resource management [7–10]. Table 1.1 summarizes major resource management challenges associated with cloud eco-system in general.

In order to achieve operational objectives while overcoming above challenges, cloud resource management platforms employ technologies and tools such as infrastructure scaling, virtualization, virtual machine (VM) migration, and equipment power state adjustment [7]. Among these technologies, virtualization is considered as the key enabling technology, which plays a major role in cloud echo-systems.

1.2.4 The Role of Virtualization in Cloud Environments

Virtualization is the process of emulating (i.e. creating a logical representation of) a hardware or software environment to users as they interact with complete instances of that environment [7]. Logical instances are created by partitioning and aggregating underlying hardware (or software) and masking configuration, implementation and other vendor specific details from the user. The ability to partition larger resources for smaller demands and to aggregate smaller resources for larger demands improves cloud infrastructure utilization and request acceptance. Abstraction of vendor specific and implementation details reduces complexity and improves interoperability. In addition to creating logical instances and abstracting implementation and vendor specific details of underlying layers, virtualization software provides resource and data isolation [11]. Furthermore, virtualization enables faster and automatic resource management due to virtual instances being deployed, decommissioned and migrated dynamically to meet variations in user demands. In addition, virtualization can satisfy operational objectives such as resource re-optimization, reduce idle power consumption and human involvement.

It is evident that virtualization technologies enhance resource management in cloud environments. However, several limitations in cloud network resource management can be identified. When a comparison is made between the virtualization of compute and storage and traditional network virtualization technologies, the later are limited in abstraction and isolation capabilities [8]. Traditionally network virtualization functional-

ities have been achieved at different levels and capacities while employing a number of technologies. These include virtual local area networks (VLAN), virtual private networks (VPN), active and programmable networks as well as overlay networks. These technologies are incapable of providing a comprehensive and automated network virtualization as they do not define a single unifying abstraction to configure, and operate a network as a whole [8]. As a result, virtual network (VN) provisioning and modifications may take several months compared to a few minutes used to deploy a set of VMs that satisfy client requests [12, 13].

1.2.5 Network Virtualization in Software Defined Networking

SDN is considered as a technology capable of addressing aforementioned limitations in existing virtualization technologies. SDN improves the network programmability and manageability by allowing infrastructure owners to dynamically establish traffic flows through a centralized controller. Fundamentally, SDN separate network control functionalities from packet forwarding functionalities in traditional L2/L3 switches. Traffic control responsibilities, traditionally taken care of by routing and switching protocols, are transferred to a logically centralized control server. The communication between SDN agents located at packet forwarding devices (SDN switches) and the network controller, is handled by a well defined communication protocol such as OpenFlow [14]. Traffic forwarding decisions are made at the controller by network control applications, and decisions (flow rules) are programmed into SDN switches through the same communication protocol.

Logical centralization of network control allows network operators to identify hot/cold spots and failures in different network components. Moreover, logical centralization allows operators to apply dynamic traffic engineering policies to minimize bottle-necks, avoid failures, and maintain high QoS and resource utilization levels. In addition, separation of data and control planes with a well-defined control protocol such as OpenFlow allows SDN to access powerful network virtualization capabilities such as data traffic

isolation, abstract implementation details and configure heterogeneous devices with common interface. However, most importantly, SDN is capable of providing network control virtualization, which is an additional layer of virtualization compared to traditional virtualization technologies.

Regardless of the fact that SDN-based network virtualization has the capacity to enhance cloud resource management and deliver IaaS with rich network services, existing cloud management frameworks lack the ability to make use of such functionalities. The major proposals of this dissertation have been motivated by these limitations visible in existing resource management platforms.

1.3 Motivations

The overall performance of a cloud resource management framework depends on how efficiently it achieves operational objectives while overcoming challenges associated with resource discovery, selection/allocation and operation functions. Complexities of these functions can be escalated if the resource management framework is not equipped with comprehensive *resource description and modeling scheme*. In addition, interoperability of different CSPs as well as resource management platforms, highly depends on resource description and modeling languages used by each of them. Hence resource description language that is capable of characterizing heterogeneous physical and virtual resources is an essential part of the cloud resource management [15]. Further, appropriate level of granularity of descriptions need to be identified to include details such as switch interfaces, control channels capacities and virtual link/switch characteristics. To this end, commonly used resource description schemes such as Resource Description Framework (RDF) and Network Description Language (NDL) fall short and need further improvements to effectively describe and model cloud resources [16].

Regardless of the fact that the, number of businesses that rely on cloud services to obtain computing as a utility is rising, the lack of control over infrastructure resources

remains as one of the strongest opposing forces to this trend. Particularly, enterprise customers expect cloud to take on the burden for businesses of managing and operating computing infrastructure, while providing the same level of control and management features as in-site infrastructure resources [17–19]. The state-of-the-art compute and storage virtualization technologies support delegation of virtual resource configuration and control features to cloud consumers through APIs and web interfaces. However, the same cannot be seen with cloud network services [20]. Furthermore, traditional virtualization technologies such as VLAN and VPN lack virtualization capabilities needed to deliver such rich network control features and high QoS, along with IaaS. Hence, there is a need for *better network virtualization technologies* that are capable of offering feature rich network services to clients while satisfying resource management objectives of CSPs.

Furthermore, cloud resource management frameworks are expected to serve client IaaS requests while guaranteeing efficient use of infrastructure resources. Proper use of infrastructure allows CSPs to serve a larger number of IaaS requests while minimizing CAPEX and OPEX. This is necessary to improve CSP's profits and satisfy operational objectives. Hence the task of selecting best resource configurations and reserving them to client IaaS requests is central to cloud resource management. This is known as the cloud resource selection/allocation problem. Due to the unpredictable nature of application demands, heterogeneity of resources, and the large number of associated variables (constraints), selecting the best resource configurations for a set of IaaS requests is a complex process that demands high computation power. Cloud resource management frameworks generally use Integer Liner Programming (ILP) based optimization approaches to solve the resource selection/allocation problem. Regardless of the fact that optimization algorithms are capable of providing better resource configurations, high algorithm complexities often lead to longer resource allocation delays. On the other hand, IaaS requests may arrive in a purely random manner with stringent QoS requirements and service delivery deadlines. For such scenarios, it is not always effective to employ optimization techniques to perform cloud resource allocation. Hence it is vital for CSPs to adopt a

resource allocation mechanism that effectively satisfies operational objectives as well as application demands within a short time span.

Resource selection/allocation algorithms are designed to accept most the number of requests while improving datacenter resource utilization. However these IaaS requests usually arrive and depart at different times resulting in high variations in request lifetimes. Also, planned and unplanned infrastructure maintenance, failure recovery, datacenter capacity upgrades and disaster recovery measures may require migrating VMs within as well as between datacenters, resulting in further disruptions in efficient resource allocation states. Such variations can affect QoS provided to cloud consumers, and may result in sub-optimal resource allocation states, which reduce request acceptance and resource utilization. To address these requirements, cloud resource management frameworks need to be equipped with seamless service migration and relocation techniques that can also improve resource utilization. This problem is further intensified in mobile cloud environments, where frequent cloud service migrations between datacenters might be required, in order to guarantee QoS requirements of mobile cloud applications.

Performance of mobile cloud services depends on a number of factors, including QoS support of the resource management platform as well as latency, bandwidth and reliability of the communication path between the datacenter and the mobile device. Although traditional large datacenter clouds process powerful computing and storage capacities, they tend to be located far from population centers. The geographical distance can cause extended delays, jitters and packet-losses that might diminish the advantages of using cloud services to boost mobile application performance. In addition, the use of wide area links can limit the CSP's and mobile network operators ability to control traffic paths and reduce communication overhead through traffic localization. Geographical distribution of cloud resources among multiple smaller datacenters situated close to mobile users in each region is one of the potential solutions to this problem. Such distribution of cloud resources close to the edge of the network allow CSPs to effectively overcome aforementioned limitations and deliver low latency high bandwidth cloud services to

mobile applications [21, 22]. However, guaranteeing QoS with regional edge datacenter clouds can still be quite challenging due to the mobility nature of users. Hence, *seamless cloud service migration and QoS aware relocation* capabilities play a crucial role in mobile cloud resource management frameworks.

Despite the fact that virtualization technologies employed in cloud resource management frameworks improve resource utilization, they also increase the risk of interrupting operation of multiple cloud tenants at the failure of the hosted hardware. Compared to individual server failures, network failures affect more cloud users as they can cause service interruptions in multiple VNs as well as cutting access to one or more servers. Similarly, in comparison to network link failures within a datacenter, inter-datacenter link failures are highly disastrous to cloud operation as such failure can completely disrupt a large number of client applications. Also, costs of inter-datacenter link failures can be significantly increased with high recovery times common in wide area networks (WANs). One of the main challenges associated with network link failures is the lack of means to detect failures quickly to apply fault tolerance measures. This is because failure of a network element usually breaks the communication path between the sender and the receiver, reducing their ability to identify and localize failures.

Network operators usually rely on distributed switching and routing protocols to identify network failures and reroute packets in alternative paths. However, for large scale networks such as inter-datacenter WANs, convergence time of network failures can last for tens of seconds. During this time the network would experience complete breakdown or serious deterioration of quality [23]. As a result, for delay sensitive applications, additional network resources (e.g. bandwidth, physical links, interfaces) are reserved as backups to protect each network element individually or end-to-end network path as a whole. Although such reservations can effectively improve fault tolerance, they may bring the network resource utilization to lower levels, causing loss of profits to service providers. In a context where application service providers obtain cloud services from networked cloud infrastructure that connects geographically distributed datacenters by

WAN, the cost of maintaining under-utilized WAN links can be detrimental to infrastructure operators [24]. Therefore *QoS aware dynamic fault restoration schemes* are vital to deliver cloud services with better QoS while maintaining high substrate resource resource utilization and service request acceptance.

QoS aware fault restoration techniques provide a powerful means of achieving fault tolerance by dynamically recalculating alternative routes after detecting failures. Even though dynamic path re-computation can potentially recover from failures of multiple substrate nodes and links, it can only provide best effort QoS. This is because availability of additional bandwidth, reliable and low latency links might not be available between required end-points. Also path restoration increases mean time to recover (MTTR) as dynamic route calculation is done after detecting failures. As a result, dynamic restoration techniques are suitable for delay tolerant, elastic traffic such as VM migration, routine database backup and content distribution. For delay, bandwidth and packet loss sensitive cloud applications such as VOIP, HD streaming and interactive gaming, faster fault tolerance techniques are required to guarantee higher levels of QoS and lower recovery delays. However, such techniques are typically involved with reserving additional link bandwidth in advance and tend to reduce overall resource utilization. In this context, *shared backup based path protection approaches* give promising solutions as they have the potential to guarantee QoS requirements while maintaining high average substrate resource utilization levels.

1.4 Thesis Objectives

Based on the aforementioned challenges and motivational aspects, there are four main research objectives in this work.

Resource Management Framework for Datacenters

The first objective of this thesis is to introduce an efficient resource management framework capable of allocating resources to IaaS requests with rich VN control features, while improving datacenter resource utilization. To perform resource management functions smoothly and efficiently, a well defined resource description and modeling schema which is capable of identifying and describing VN elements and sub-components needs to be integrated into the framework. Efficient resource allocation implies satisfaction of user QoS requirements as well as CSP's operational objectives with low computation overhead and allocation time.

Seamless Service Migration and Relocation in Mobile Cloud Environments

Although bringing cloud resources to the edge of the mobile network provides a number of benefits, it poses a number of resource management challenges due to user mobility between regions. As the second objective of this work, a resource management framework that is capable of providing cloud services from small-scale regional datacenters (RDCs) to mobile applications will be introduced. In the event of a user movement between regions, the resource management framework must evaluate whether the level of QoS satisfy SLA requirements and performs service migrations to appropriate regional datacenters as necessary. Virtualization and flow-control capabilities provided by SDN has been regarded as suitable technologies to conduct service migrations transparently to the end user.

QoS-aware Fault Tolerant Cloud Services with Better Resource Utilization

One of the main challenges faced by CSPs in providing IaaS using networked cloud infrastructure that interconnect geographically-distributed datacenters, is the operational costs incurred from under-utilized inter-datacenter WAN links, which reserve large amounts of additional backup bandwidth to protect each other. Hence the third objective of this work is to evaluate dynamic virtual link restoration capabilities of SDN and how it can

satisfy QoS requirements of cloud users as well as operational objectives of cloud service providers, during substrate link failures.

Guaranteeing QoS and Fast Recovery from Substrate Link Failures

Even though SDN based dynamic restoration promises higher resource utilization levels, it imposes additional delays to the recovery process by direct controller involvement in path computation and installation. Path protection relies on reserving additional backup bandwidth in advance and hence delivering QoS guaranteed faster recovery. However it is necessary to explore opportunities to minimize additional bandwidth reservation to deliver profitable cloud services. Thus, the final objective of this thesis is to explore the ability of backup sharing techniques to optimize substrate resource utilization while providing QoS guaranteed fast recovery from any single link failure.

1.5 Contributions

The motivations and objectives of this thesis have resulted in four major contributions. These key contributions can be highlighted as follows:

An efficient cloud datacenter infrastructure management framework capable of delivering rich network control features through SDN virtualization

Performance of the cloud resource management framework depends on how efficiently, accurately and timely it delivers infrastructure to satisfy a maximum number of cloud service requests. Therefore, as the first major contribution of this work, a cloud datacenter infrastructure management framework is presented, that is capable of delivering rich network control features to IaaS consumers. In order to achieve this, a datacenter resource management architecture has been designed to facilitate automated and on-demand provisioning of compute and network infrastructure. The proposed framework is equipped with an improved Infrastructure Description Language (INDL) schema to

describe resource attributes required to model OpenFlow based SDN with virtualization support. An Ontology-based resource composition approach has been adopted to perform VN resource allocation. The composition approach is capable of allocating resource with less computational overhead and time. User experience has been improved by providing a high degree of control over network connectivity and QoS parameters. Performance of the proposed cloud resource management framework has been evaluated by implementing on a real-physical test-bed as well as a mininet emulated test-bed.

Regional cloud service framework that allocate IaaS to mobile applications and guarantees QoS through Seamless service migration

As the second major contribution, a mobile cloud resource management framework that allocates cloud resources from RDC to mobile applications has been proposed. This work has been conducted in collaboration with a former M.ASc. student named Wijaya Ekanayake, and published in [25]. Multi-tenancy within an RDC is achieved by virtualizing compute and network resources, by leveraging the architecture presented in the first contribution. The framework for mobile applications creates infrastructure on demand and provides IaaS for each mobile device.The framework measures real-time QoS parameters during mobility, and then reactively relocates cloud resources to a RDC to improve QoS and service provider profits while maintaining end-to-end transparency. The framework benefits from the network control plane programmability offered by SDN. OpenFlow is used to perform traffic flow path creation, flow path update, layer 2 packet creation together with IP mapping and rerouting for the seamless and transparent cloud access even during inter-regional service mobility. In order to evaluate the performance of the framework with respect to existing scenarios, a prototype has been developed on Mininet emulation platform and a physical test-bed.

QoS aware fault tolerant cloud resource management framework with SDN-based dynamic restoration to ensure high substrate resource utilization

An SDN based fault tolerant IaaS resource management framework for networked cloud infrastructure (NCI) has been introduced as the third main contribution of this thesis. The NCI is composed of geo-distributed cloud datacenters interconnected with inter-datacenter WAN. The framework performs IaaS resource allocation by formulating an integer linear programming (ILP) problem and solving it to determine optimum reservation combinations for a set of IaaS requests arrived within a given time interval. Two SDN-traffic engineering (SDN-TE) based virtual link restoration algorithms have been proposed that perform proactive and reactive path computation to address the trade-off between QoS awareness and faster recovery. The framework employs SDN based real-time monitoring to detect network failures and trigger SDN-TE algorithms to dynamically reroute traffic through alternative paths.

QoS guaranteed fault tolerance from any single link failures through shared backup protection

A novel ILP based path protection scheme has been introduced for SDN-based NCI as the final major contribution of this thesis. The proposed shared backup path protection scheme is designed with the objective of ensuring faster recovery from any single link failures. Backup paths which are capable of providing full bandwidth as well as other QoS parameters of the primary paths are computed along with the primary paths to jointly optimize the bandwidth sharing and resource utilization. Both primary and backup paths are configured in the substrate network at the resource allocation phase with the help of a centralized SDN controller. The protection mechanism relies on switching to pre-installed backup paths in the event of failure. This is done without the direct involvement of the SDN controller to avoid communication delays. The adverse effects on substrate bandwidth utilization have been reduced by maximizing backup bandwidth sharing. Thus the proposed approach improves overall substrate resource utilization, while guarantee-

ing QoS in any single link failure. The performance of the proposed shared-backup link embedding approach is evaluated through emulated test-bed experiments.

1.6 Overview and Organization of the Dissertation

The remainder of this thesis has been organized as follows:

- Chapter 2: Presents a comprehensive background study on cloud computing, focusing more on centralized network management aspects. First a formal definition of cloud computing is provided with elaboration of the basic cloud architecture, service models and different types of cloud deployments. Second, the role of virtualization in cloud resource management is presented, followed by an explanation of how resource discovery, allocation and operation are done in network virtualization. Third, some background to SDN and applications of SDN-based technologies in cloud environments have been provided as they have used to achieve cloud resource management objectives. Finally, details on cloud resource management platforms and mobile cloud computing frameworks have been presented.

- Chapter 3: Introduces an SDN based IaaS resource management framework that is capable of providing rich network control features to cloud consumers. This chapter is organized into three parts. The first part details limitations in existing SDN adoptions in cloud resource management frameworks. A prototype OpenStack cloud test-bed has been deployed with OpenDaylight SDN controller to experimentally evaluate these limitations. In the second part, the proposed cloud IaaS resource management framework that integrates control-plane virtualization capabilities to overcome those limitations has been presented. Through a set of experiments on physical and Mininet test-beds, ability of the proposed framework to deliver cloud IaaS services with a high degree of control over traffic has been demonstrated.

- Chapter 4: Presents a regional cloud framework with SDN to provide QoS-aware IaaS for mobile users. With the proposed distributed RDC architecture, it is possible to allocate virtual computing resources for mobile applications. During user mobility between regions, the proposed framework measures QoS parameters of cloud services. If expected QoS cannot be provided to the user at the user's destination region, a service relocation algorithm is executed, in order to identify suitable RDCs for cloud service relocation. In such scenarios, the framework minimizes the service cost and maintains QoS by seamlessly relocating cloud resources to a RDC by using dynamic network configuration capabilities of SDN.

- Chapter 5: Illustrates the dynamic restoration-based fault tolerant cloud resource management framework designed for NCI. Initially the chapter provides background information related to inter-datacenter network failures and distinguishing characteristics of path protection and restoration approaches in providing fault tolerance. Then the framework architecture and the ILP formulation required to address IaaS resource allocation problem are described. Subsequently, the proposed QoS aware reactive path computation and shared risk link group (SRLG) aware proactive path computation algorithms have been introduced. Finally SDN based implementation and performance evaluation are presented.

- Chapter 6: Describes the shared backup based path protection approach which provides QoS guaranteed faster recovery from any single link failure. The chapter initially highlights some prominent work in the domain of virtual link protection and shared backup techniques. Next the proposed path-embedding approach has been described with an example scenario. Finally implementation details and performance evaluation have been presented.

- Chapter 7: provides the conclusion by summarizing research that has been conducted in the domain of cloud resource management. It also describes identified limitations of work conducted in the scope of this thesis and highlights several

future research directions.

2 Background

The cloud service model introduces three layers of services. Among these, IaaS has attracted significant attention from cloud research communities. While the compute and storage resource management has been developed and studied to a greater extent, the use of network resource management in the context of IaaS is still in its early stages. There is a need for a comprehensive virtualization framework capable of providing users with low-level network and compute resource control while improving underlying resource utilization.

In this chapter, background details relevant to the main proposals of this work are presented. Initially, the cloud service model is defined and its architecture is presented. This is followed by an explanation on the role of virtualization in cloud, and details on how SDN based virtualization can improve network management. Subsequently, attention is given towards existing cloud resource management platforms. Out of existing cloud platforms, focus is placed on OpenStack architecture since in this thesis, experiments have been conducted on OpenStack to evaluate network virtualization capabilities in existing clouds.

One of the major challenges in cloud resource management is satisfying CSP's operational objectives in the run-time, while guaranteeing QoS for IaaS allocated for user applications. This is especially challenging in mobile cloud platforms where user movements often violate QoS and result in additional costs to cloud service providers. Hence in the latter part of this chapter, background details required to address mobile cloud resource management problem are provided.

2.1 Introduction to Cloud Computing

Wide industrial adoption of cloud computing can be attributed to a number of direct and indirect factors. Listed below are four of the key driving forces behind the evolution of the cloud paradigm:

1. *Popularity of wired and wireless internet infrastructure and smart portable devices.* A main contributing factor to the popularity of the cloud model is recent advancement of internet related technologies and infrastructure such as high speed broadband networks, wireless access points, smart phones, tablets, sensors and wearable computing devices (apple-watch, fit-bit, Google glass etc.). This popularity has resulted in high demand for convenient internet access and remote computing infrastructure services capable of performing resource intensive processing tasks.

2. *Growth in video streaming, social media and Massively Multi-player Online Games (MMOG).* Popularity of large scale multimedia applications including video streaming (YouTube, Netflix), social media (Facebook, twitter) and online gaming (World of War-craft, Final Fantasy) has increased the demand for low latency high bandwidth communication with processing servers and content storage. Maintaining multiple Geo-distributed datacenters to serve end-users with low latency is not cost effective for application service providers.

3. *Wide use of big data and data processing technologies.* Collection and processing of vast amounts of data received through social networks, scientific applications (genome research, astrophysics), advance sensor networks etc. usually generate highly varying compute and storage demands.

4. *Ability to harness advantages of economies of scale.* It is more economically viable for small and medium businesses to acquire compute services from existing infrastructure (datacenter) owners rather than building everything from scratch. Also,

datacenter owners have the opportunity to maximize their profits through providing infrastructure services to multiple businesses, while harnessing the benefits of economies of scale.

Regardless of the fact that economic and profit driven motives are viewed as major forces behind its inception, the cloud model has emerged to fill a void caused by rapid technological growth in the IT industry over the last decade.

2.1.1 Definition and Essential Characteristics

For several years, *cloud computing* has remained an abstract technological term due to lack of a standard definition. In 2011, a widely accepted definition for cloud computing was given by the NIST as follows: "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [26]

The NIST definition of cloud computing is further elaborated by the following five essential characteristics:

1. *On-demand self-service*: Cloud service users can flexibly obtain computing, network and storage resources (e.g. through VM) with minimal human intervention.

2. *Broad network access*: Acquired resources are accessible remotely over the internet through heterogeneous platforms such as work stations and mobile phones.

3. *Resource pooling*: Cloud resources are abstracted and pooled (through technologies such as virtualization) and shared by multiple consumers. Resources can be dynamically assigned and released based on consumer demand.

4. *Rapid elasticity*: Resources can be allocated and released rapidly with minimal human interaction.

5. *Measured service*: Metering and resource optimization facilities are automated and integrated with the cloud platform allowing users to measure their usage and benefit from the pay-as-you-go service model.

2.1.2 Architecture and Service Models

The cloud business model follows the service oriented architecture (SOA), where discrete units of functionalities interact to serve consumers [27]. Hence application software, platform and hardware resources are designed as layers of services and each layer interacts with other layers through services.

Architecture

Architecture of a cloud computing environment can be divided into three main layers [28]: the physical resource layer, the resource abstraction/control layer, and the service layer. These layers and their main components are illustrated in Figure 2.1.

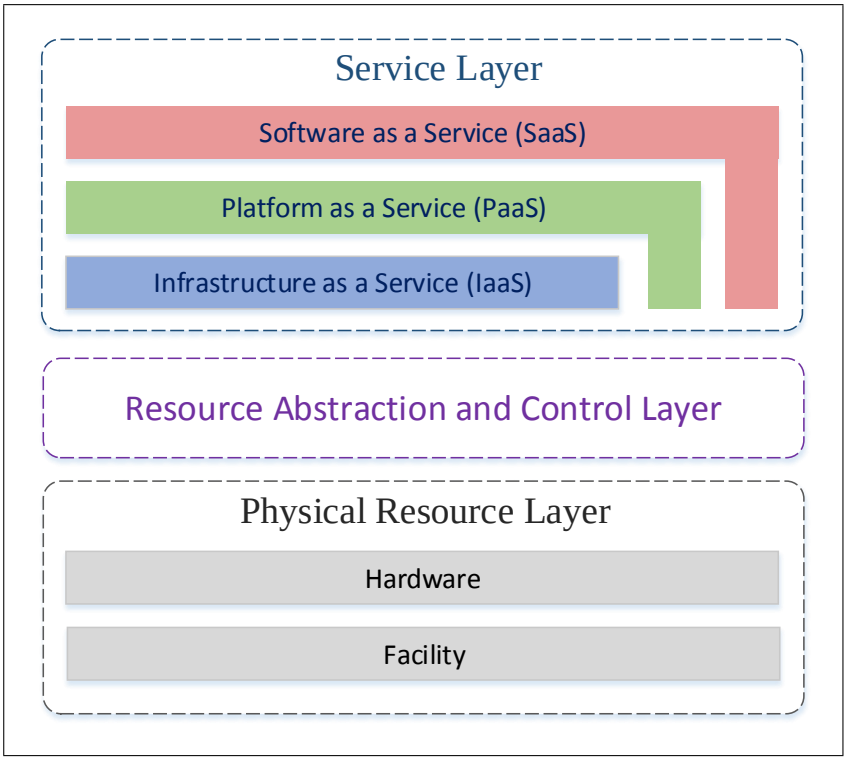


Figure 2.1: Layered Cloud Architecture [28]

1. *The Physical Resource Layer*: This is the lowest layer in the layered architecture that is composed of facility (datacenter) infrastructure and hardware resources. The facility resources include buildings, server racks, power supplies, auxiliary power batteries, heating, ventilation and cooling equipment. The hardware resources include computing resources (CPU, Memory, GPU), storage resources (hard disks, backup storage media) and networking resources (switches, routers, firewalls, links and interfaces). Increasing and decreasing capacity in the physical layer requires adding (connecting, powering up) or removing (disconnecting, shutting down) servers and network equipment to the facility.

2. *The Resource Abstraction and Control Layer*: This layer provides one of the most important functionalities in cloud architecture. Allocating physical resources directly for application services does not provide sufficient flexibility and controllability to support dynamic service provisioning. Hence virtualization technologies are used to logically partition and aggregate hardware resources for service deployment through abstraction. Beside masking hardware implementation-specific details from service layers, virtualization also guarantees isolation between logical resources. To achieve this, the resource abstraction and control layer is composed of special software, commonly identified as hypervisors. These hypervisors create virtualized elements from compute, storage and network hardware resources. Compute hypervisors abstract physical CPU and memory resources into VM or containers (e.g. Docker). As well, storage virtualization technologies are used to create storage blocks and virtual drives. In addition, network virtualization technologies such as VPN, SDN and network function vitalization (NFV) are used to dynamically deploy VNs and virtualized network functions.

3. *Service Layer*: The top layer of the cloud architecture is known as the Service Layer and it contains interfaces corresponding to infrastructure, platform and software services. Infrastructure service composition interface contains virtualized hard-

ware resource elements (e.g. VMs, storage blocks, VNs) and provides IaaS to consumers. Similarly, platform and software service interfaces deliver PaaS and SaaS to end users. IaaS, PaaS and SaaS are the three main service models in the cloud eco-system. In a cloud environment where IaaS, PaaS and SaaS are provided to consumers, platforms services are typically implemented on IaaS, and software services are implemented on PaaS. These basic service models are further detailed below.

Basic Service Models

Cloud service models have evolved into a variety of specific services such as Network as a Service (NaaS), Firewall as a Service (FWaaS), GPU as a Service and Database as a Service (DBaaS). However, this *everything-as-a-service* model, also known as XaaS, has evolved from three basic service models: SaaS, PaaS and IaaS. Three basic cloud service models are further elaborated as follows:

1. *SaaS*: Layer of service where consumers gain access to software and application interfaces that are hosted on cloud providers infrastructure. SaaS provides the highest level of abstraction and keeps unnecessary details such as execution platforms, operating systems, firmware and physical infrastructure details hidden from the consumer. Software installation, upgrading, patching and licensing etc., as well as maintaining operating platforms and infrastructure management, are responsibilities of the SaaS provider.

2. *PaaS*: Provides the consumer with the ability to select preferred platforms, operating systems, development and execution environments, database systems etc. to deploy and operate their specific applications on cloud infrastructure. Cloud users do not manage or have control over underlying computing and network infrastructure, but have full control over applications they deploy.

3. *IaaS*: Layer of service where the cloud user is given the ability to deploy comput-

ing resources and networking resources as necessary over the physical infrastructure (servers, network equipment etc). Infrastructure services are delivered as virtualized service units of hardware resource, which abstract hardware specific and implementation details. For example, CPU and memory resources are abstracted into VMs and compute containers. As well, Storage resources are abstracted into blocks and virtual drives and network resources are abstracted into virtual switches, links, networks and VN functions.

2.1.3 Cloud Deployment Types

The cloud service model gives consumers the freedom to select a deployment strategy based on their business needs. NIST has identified four deployment types characterized by factors such as privacy, security and infrastructure management:

1. *Private Cloud*: Cloud infrastructure is dedicated to a single organization. The organization itself or a third party may own, manage and operate infrastructure located on or off premises.

2. *Community Cloud*: Cloud infrastructure is dedicated to a single community comprised of consumers with shared interests or concerns (e.g. similar security requirements, access rights). Infrastructure may exist on or off premises.

3. *Public Cloud*: Cloud infrastructure is provided for the general public. It is owned and operated by CSPs and located at CSP's premises.

4. *Hybrid Cloud*: Comprised of properties of two or more of above deployment models and bounded by a standard or proprietary technology that enables data and application portability operations.

2.2 Virtualization in Cloud

Cloud computing is often highlighted for its flexible resource management capabilities. It is also recognized for its distinguishing characteristics such as shared pool of resources that allow multi-tenancy, dynamic resource provisioning, elasticity (increase and decrease allocated resources at run-time), and pay-as-you-go billing. The main enabler technology that facilitates almost all of these characteristics is virtualization. Notably, virtualization is employed to partition, aggregate, isolate and abstract datacenter and inter-datacenter network infrastructure resources. Resource pooling allows cloud service providers to create logical instances of virtual infrastructure out of heterogeneous physical infrastructure. These logical instances are performance-isolated and constructed by partitioning and aggregating physical resources. A layer of abstraction is also used to mask hardware specific information and implementation details of the physical infrastructure from virtual instances. These virtual instances can be dynamically provisioned to be used by multiple cloud consumers, also known as tenants. In addition, the resource abstraction allows cloud providers to more flexibly add and remove physical infrastructure from the virtual resource pool, as well as dynamically increase and decrease capacities of virtual resources. Dynamic and flexible resource provisioning and releasing also allow cloud service providers to maintain better infrastructure utilization and also do pay-as-you-go billing, realizing the long speculated utility computing notion [29].

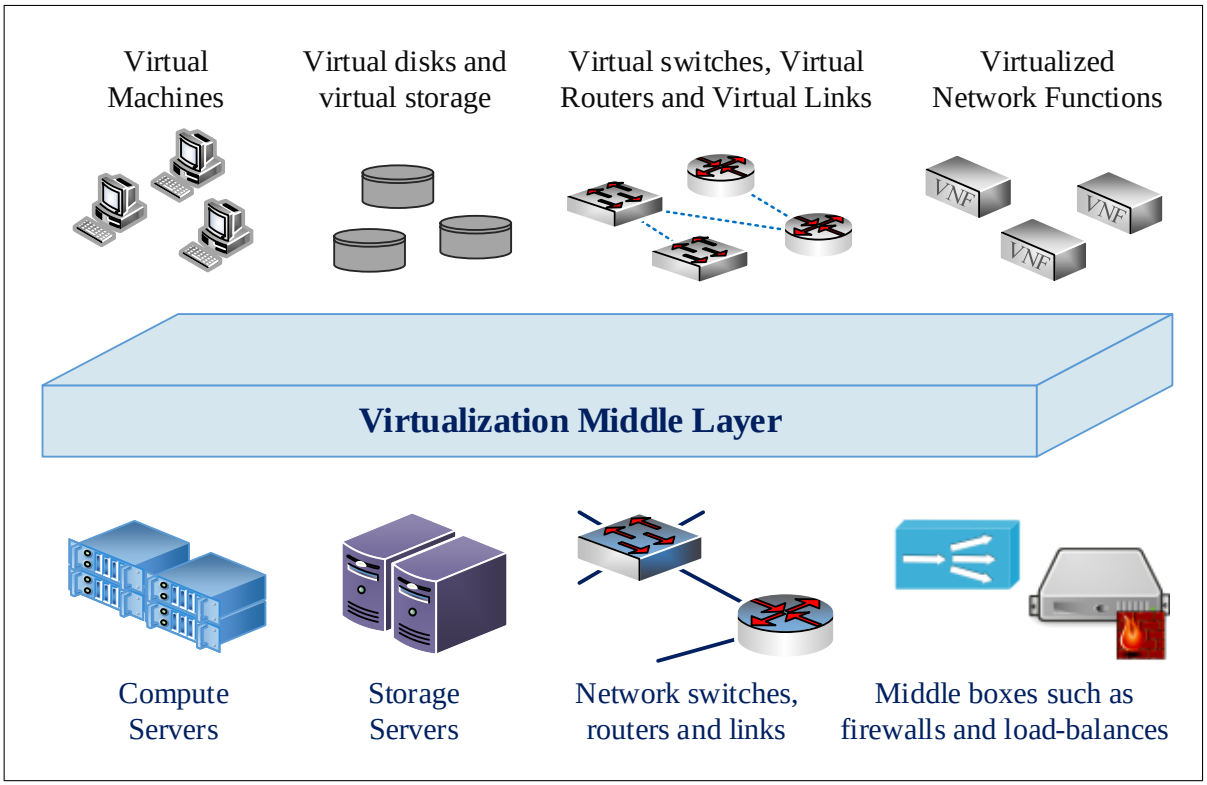


Figure 2.2: Creating Logical Instances of Datacenter Resources by Virtualization Layer

In cloud datacenter environments, virtualization software is used to create VMs from computing servers, virtual disks and block storage from storage resources, virtual switches/routers from network devices, and virtualized network functions from middle-boxes. This is illustrated in Figure 2.2. These virtual elements can be dynamically provisioned and can be logically interconnected to form feature-rich VNs to satisfy client IaaS requests. Even though concepts of virtualization apply to compute, storage and network infrastructure alike, the abstraction methods, partition/aggregation and isolation technologies vary due to unique characteristics in each resource type. In this sections, virtualization technologies are categorized and detailed with respect to three main infrastructure types, namely: storage, compute and network.

2.2.1 Virtualization of Storage Resources

One of the most widely used and well studied areas of cloud services is data storage [30, 31]. In traditional datacenters and computing environments, local physical storage disks and network storage solutions such as storage area networks (SAN) and network attached storage (NAS) were directly used to store data. SANs and NASs introduce a level of virtualization to pool storage resources. However, they are highly expensive to maintain as they require special equipment, dedicated servers, as well as high levels of technical expertise. Since cloud architecture offer an additional level of abstraction, infrastructure providers are able to use other none-specialized heterogeneous storage infrastructure along with traditional network storage in a common pool of resources to be shared among multiple consumers. This reduces the cost of storage as well as improves the resource utilization in datacenters.

There are three main data storage abstractions used in clouds:

1. *Database level*: In the database level, virtualized storage resources are offered along with platform services to meet business and organizational demands for relational and semantic (e.g. ontology, triple-store) databases.

2. *Object storage*: Compared to traditional hierarchical file-systems such as NAS and SAN, virtualized object storage systems do not maintain tree-like structure to organize data. Instead they maintain a flat layout to store files and call by unique identifier. This eliminates the need of specialized hardware and allows cloud storage systems to be composed of diverse hardware resources and abstracted into a single storage pool.

3. *Block storage*: Block storage is the lowest level of storage. It ignores data semantics and is generally combined into volumes. These physical storage volumes are virtualized in cloud and virtual volumes are dynamically assigned to VMs as disks.

Both object and block storage play a vital role in clouds by facilitating storage level elas-

ticity. Furthermore, they provide storage isolation and makes physical location, physical disk and partition details transparent to cloud users [31].

2.2.2 Virtualization of Compute Resources

Recent advancements in compute virtualization can be considered as one of the main driving forces behind inception of the cloud service model. Compute resource virtualization allows server computers to support the concurrent execution of multiple operating systems while granting resource/performance isolation, better application performance and enhanced security. Compute resources such as CPU and memory are virtualized by a special layer of software known as hypervisor [32]. Hypervisors (e.g. VMWare, Xen, KVM, virtual-box) virtualize server resources through resource partition/aggregation, abstraction and isolation. Partition of compute resources involves creating multiple VMs on a physical server. Aggregation combines CPU and memory resources from multiple servers and provides a logical view of a single powerful computer.

Hypervisors use several abstraction technologies to hide hardware specific and implementation level details from applications, as well as to combine resources from heterogeneous server hardware into a single resource pool. Abstractions and isolation further enable VMs to be dynamically instantiated, suspended, terminated, duplicated and migrated to different locations without disrupting operations of applications executed on top of them. Typically these features are achieved through platform independent application programming interfaces (APIs). Further resource isolation allows increasing and decreasing VM capacities (CPU cores, memory) during run-time. In comparison to conventional server-based compute resource management, VMs provide flexibility and enhance cloud service providers ability to operate compute resource efficiently with improved resource utilization [33].

2.2.3 Virtualization of Network Resources

In a traditional datacenter environment, computing and storage infrastructure are interconnected with switching and routing networks. Intra-datacenter networks are typically switched networks composed of top-of-the-rack, aggregation and core switches connected with physical wires (e.g. Ethernet, optical). As well, datacenters are interconnected with wide area routed networks. Similar to compute and storage virtualization, virtualizing network resources involve adding a layer of abstraction to make underlying network hardware details transparent to users. In addition, virtualization allows partitioning and aggregating network resources and guarantees resource and performance isolation. Traditional network virtualization technologies such as VLAN and VPN fall short in facilitating these requirements as they were designed for different purposes such as segmenting traffic at lower levels (e.g. switch ports) and providing end-to-end secure tunnels. Hence, recent advancements in network virtualization and SDN are employed to virtualize cloud network infrastructure as they provide more flexibility for dynamic network management. Elaboration on virtualization of network resources is presented below.

2.3 Network Virtualization and Software Defined Networking

For over a decade, both industrial and academic communities have predicted an inevitable stagnation in the internet that will hinder its performance and further growth. This has been termed as internet ossification [34–36]. Network virtualization was initially proposed as a solution to this ossification problem caused in the public internet [37–40].

2.3.1 Internet Ossification and Other Catalysts Behind Network Virtualization

The internet is composed of multiple infrastructure provider networks with various organizational objectives. Due to the multi-provider nature of the internet, making major architectural changes or adding new technologies requires agreements between multiple stakeholders, which has proved to be impractical. As a result, a number of favourable technologies such as differentiated services and active/programmable networks failed to gain sufficient attention to deliver their intended benefits [38]. Furthermore, internet protocol (IP) based networks suffer from several drawbacks. First, traffic and bandwidth isolation cannot be easily achieved to guarantee QoS and security requirements of users. Second, since IP addresses are not limited to the networking layer, higher level abstractions are not achievable. Third, management complexities are significantly high due to a multitude of distributed protocols. In particular, designing networks, identifying the operating state, detecting errors, and reasoning and correcting these errors has become an arduous task that requires specialized knowledge. Finally, lack of isolation and the high number of stake holders has resulted in resistance to experimentation and further growth in traditional networks. These resisting forces have long been identified by researches and known as internet ossification.

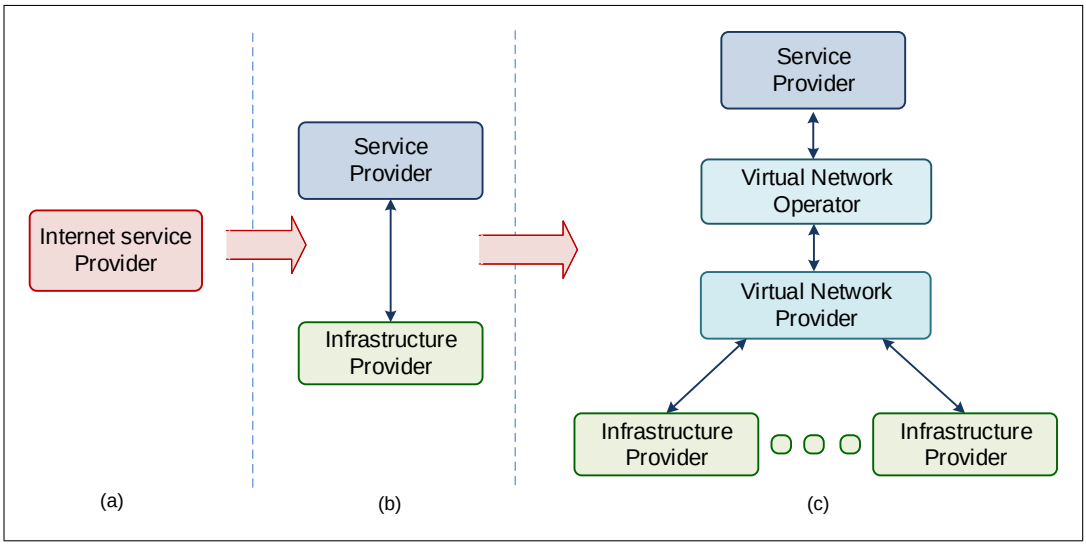


Figure 2.3: (a) Traditional internet service provider, (b) decoupling roles to infrastructure provider and service provider (c) VN provider and operator to provide network services [41]

Network virtualization is considered a promising solution to overcome these ossifying forces by allowing several logical networks to coexist in a shared physical infrastructure composed of multiple infrastructure provider (InP) networks [35,38–40]. It introduces the virtualization as a intermediary function by initially decoupling the traditional internet service provider's (ISP's) role into a provider of service and a provider of infrastructure [41] as shown in Figure 2.3(a) and (b). VN provider(VNP) and VN operator (VNO) roles are introduced between service provider and InP as shown in Figure 2.3(c), masking InPs from the service providers. This layer of virtualiztion, composed of VNO and VNP, allows composition of VNs for service providers combining physical infrastructure from multiple InPs. Moreover, abstractions and isolation provided through virtualization allow coexistence of multiple VNs on the same physical infrastructure, as illustrated in Figure 2.4.

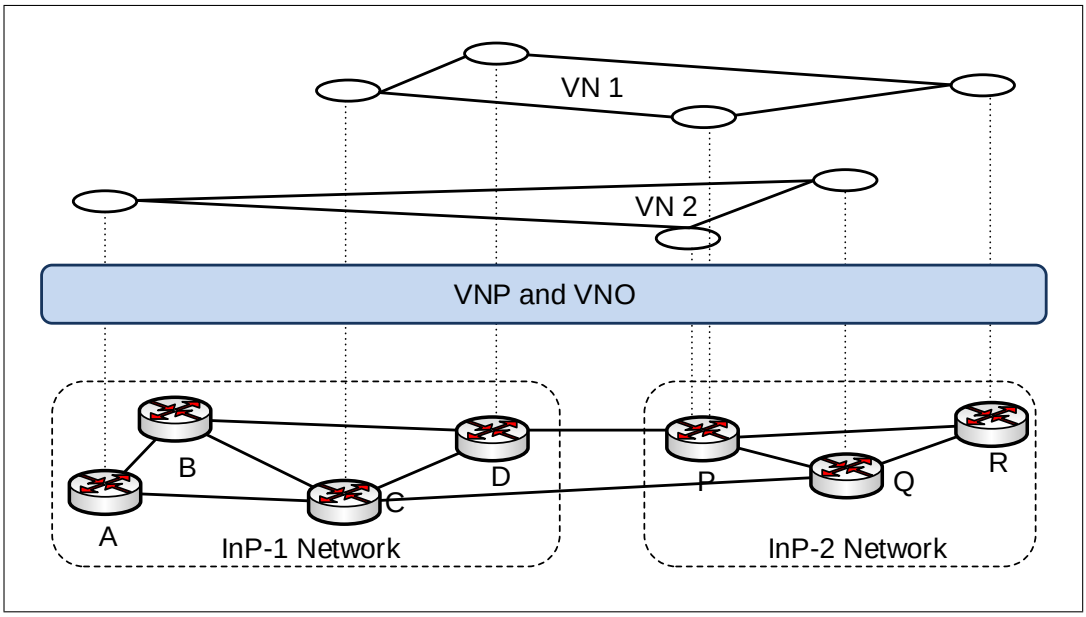


Figure 2.4: Coexistence of multiple VNs in a network virtualization environment

2.3.2 Advantages of Network Virtualization

From an architectural perspective, network virtualization provides several advantages due to this separation of functions. These advantages include:

- *Coexistence*: Multiple VNs can coexist in same physical infrastructure
- *Recursion*: One VN can act as InP for another VN built on top (nested)
- *Inheritance*: Upper-level VN can inherit properties from Lower-level VN
- *Revisitation*: Each server/link can host multiple virtual instances
- *Isolation*: Traffic and performance isolation to guarantee security and fault-tolerance
- *Flexibility*: Independence facilitated by isolation allows changes to VNs less complicated
- *Manageability*: Each VN can be created, operated, migrated and terminated independently

- *Scalability*: Flexibility to add/remove infrastructure based on demand
- *Heterogeneity*: Both underlying infrastructure and constructed VNs can be composed of different technologies

2.3.3 Software Defined Networking

SDN [42, 43] is a novel network management paradigm that brings several innovations into network management [44]. These include:

- Separation of the control and data planes
- Control plane centralization
- Control plane progammability
- Standardizing communication between control and data planes through APIs

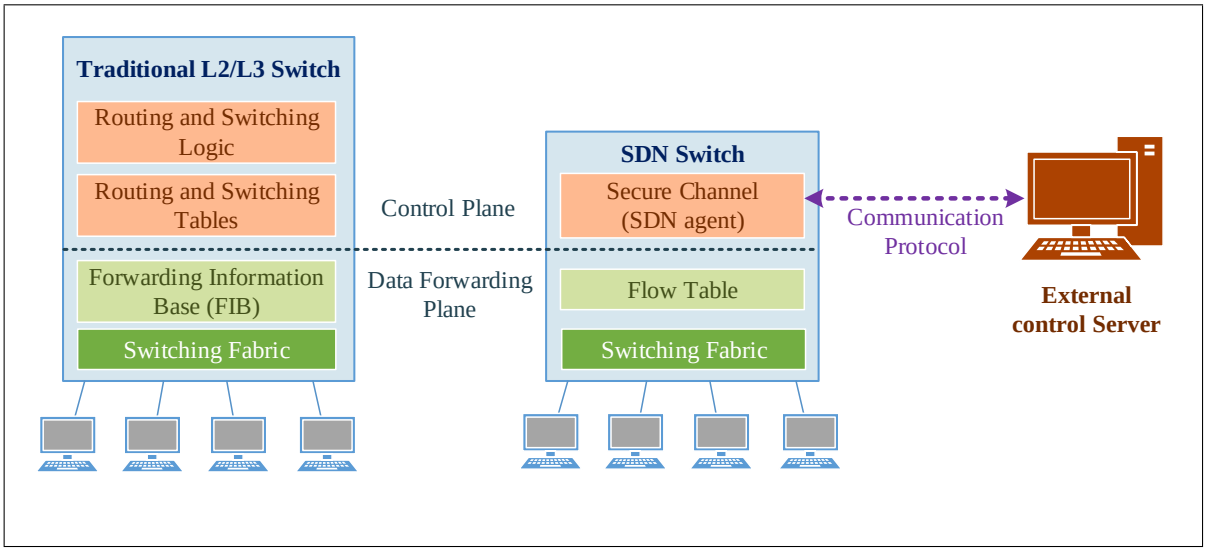


Figure 2.5: Traditional switch vs software defined networking switch

SDN improves the network programmability and manageability by allowing infrastructure owners to dynamically establish traffic flow through a centralized controller. At its core, SDN separates network control functionalities from packet forwarding functionalities in traditional L2/L3 switches, as shown in Figure 2.5. After separation, the

control functions are moved out of the network devices to a logically centralized server. As a result, network equipment such as switches and routers act as simple forwarding elements and compose the data-forwarding plane. Control logic of all switches and routers composes the control plain. This separation of planes and formation of a centralized control plane is illustrated in Figure 2.6.

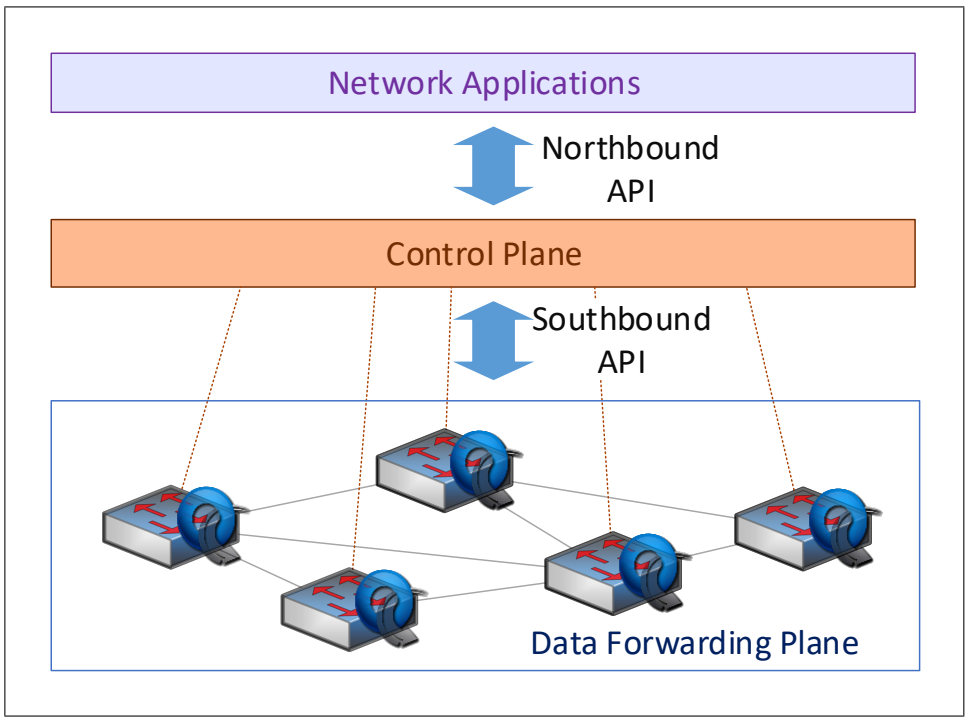


Figure 2.6: Centralized control plane and the data plane in software defined networking environment

The communication between data and control planes, as well as control plane and applications is implemented through APIs. As well, the communication between control plane and applications is defined by north-bound API. The South-bound API defines communication between the control plane and the data plane. In addition to North-bound and South-bound APIs, East-bound and West-bound APIs can exist to horizontally connect different control plane instances. APIs allow customized but strictly defined communication policies between applications, forwarding elements and other controllers.

Table 2.1: Major Fields of an OpenFlow Rule (version 1.0)

Header Match Fields	Counters	Actions	Priority
Ingress Port Ethernet Source/Dest Addr Ethernet Type VLAN id VLAN Priority IP Source/Dest Addr IP Protocol IP ToS ICMP type ICMP code	*Per Flow Counters* Received Packets Received Bytes Duration seconds Duration nanosec.	Forward (All,Controller, Local, Table, IN port, Port num, Normal, Flood) Enqueue Drop Modify	Lower value represents high priority and vice-versa.

2.3.4 OpenFlow

OpenFlow [14] standardized by open networking foundation is one of the most studied and well-known southbound APIs. Other south-bound APIs exist, such as Cisco proprietary OnePK [45]. However, Open-flow has been widely adopted by industrial, academic and open source communities. OpenFlow defines a set of packet handling rules. Each rule contains a match, counter, action and priority fields to select a sub-set of traffic, and to perform a specified action such as drop, forward and modify. Table 2.1 illustrates some header match fields, counters and actions that are used in OpenFlow 1.0 [46] rules.

The control-plane of an OpenFlow based SDN environment can be implemented with several OpenFlow SDN controllers such as NOX [47], POX [48], Floodlight [49], ONOS [50] and OpenDalight [51]. OpenFlow protocol allows controllers to add, update and delete flow entries in switch flow tables. Typically with the arrival of a new packet at one of its switch ports, an OpenFlow switch checks its internal flow table to determine how to forward the packet. Forwarding decisions are made by identifying appropriate flow rules after matching ingress port and/or packet header fields with match fields of available flow rules. If multiple flow rules match the incoming packet, priority value in each flow rule is used to select the suitable rule. If a matching rule is not available in the flow table, the packet is dropped. However, in most cases, a default rule with least priority is used to forward the packet to the controller.

OpenFlow controllers use two approaches to add flow entries, namely reactive and proactive. In the reactive mode, the controller installs new flow rules after receiving new packets that don't match with previously installed specific flow rules. In such cases, default forward-to-controller rules are triggered at the switch and a new packet is forwarded to the controller. Based on packet processing policies imposed by north-bound network control applications, the controller may determine how to forward the packet. It may also install a new specific flow rule to forward subsequent packets with similar match fields. In the proactive mode, the controller installs flow rules statically on switches before they are needed. In some cases, a hybrid approach that combines proactive and reactive flow generation may have been used by statically placing several rules in advance and dynamically installing rules during operation based on new packet arrivals.

2.3.5 Achieving Network Virtualization Through Software Defined Networking

Inherent support for virtualization can be considered as one of the most important advantages of SDN [52]. Centralizing control functions in a server allows applications to benefit from end-to-end view over the network. With the end-to-end view over the network, network managers are able to provide better QoS guarantees to application service providers by forming VNs for each IaaS request. Creating independent VNs allow applications to deploy different policies and protocols without interfering with the operation of other applications running on the same physical infrastructure. Hence, beside SDN controllers, the control-plane is composed of network virtualization hypervisors such as FlowVisor [53], AdVisor [54] and CoVisor [55]. Figure 2.7 illustrates a typical network virtualization platform with SDN-based network hypervisor (e.g. FlowVisor).

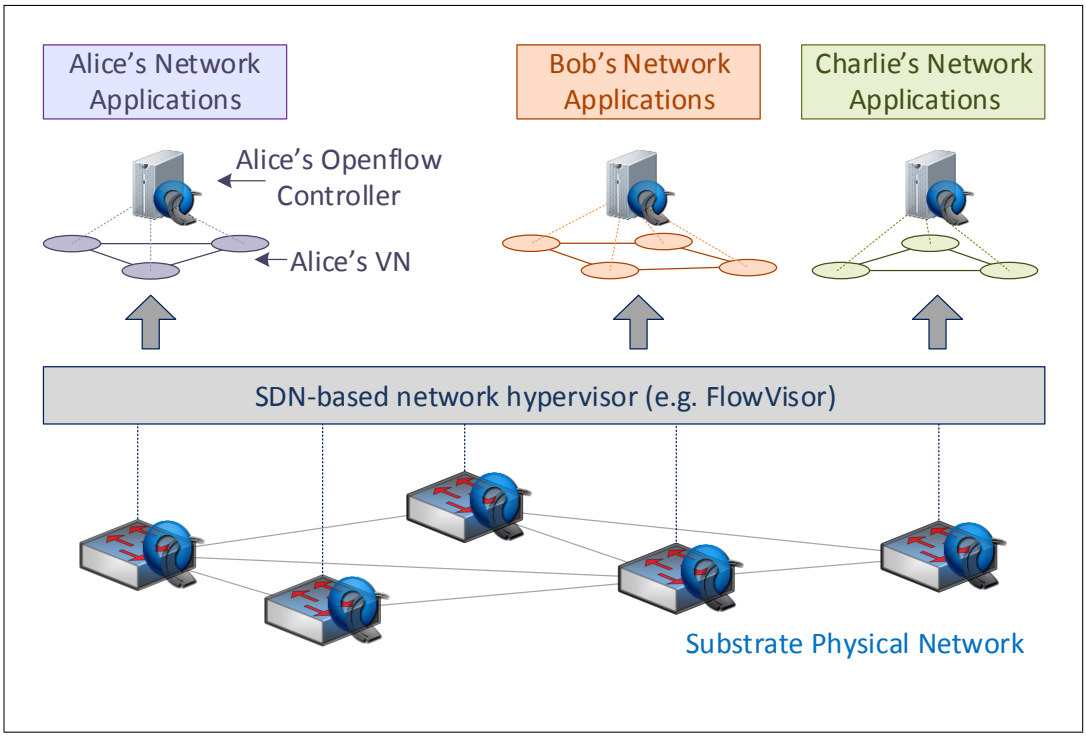


Figure 2.7: Software defined networking-based network virtualization

2.3.6 FlowVisor: FlowSpace Segregation and Abstraction

FlowVisor controls underlying physical network through OpenFlow protocol. It introduces the notion of network slicing by partitioning the entire geometric space composed of packet header fields accessible through OpenFlow protocol. Thus, each network slice is carved as a geometric shape out of multi-dimensional space composed of ten packet header fields and ingress physical switch ports [56]. These slices can be subsequently associated to SDN controllers which would see them as entire physical networks. A simplified example of defining a FlowSpace out of three dimensional space (MAC address, IP address and TCP port) is illustrated in Figure 2.8 (a). FlowVisor can also use wildcards to include ranges as well as negation (e.g. all packets except source IP equal to 137.122.88.97), unions (e.g. ethertype is ARP or destination MAC FF:FF:FF:FF:FF:FF) and intersections (e.g. TCP port is 22 and ingress switch port 4) to define slices.

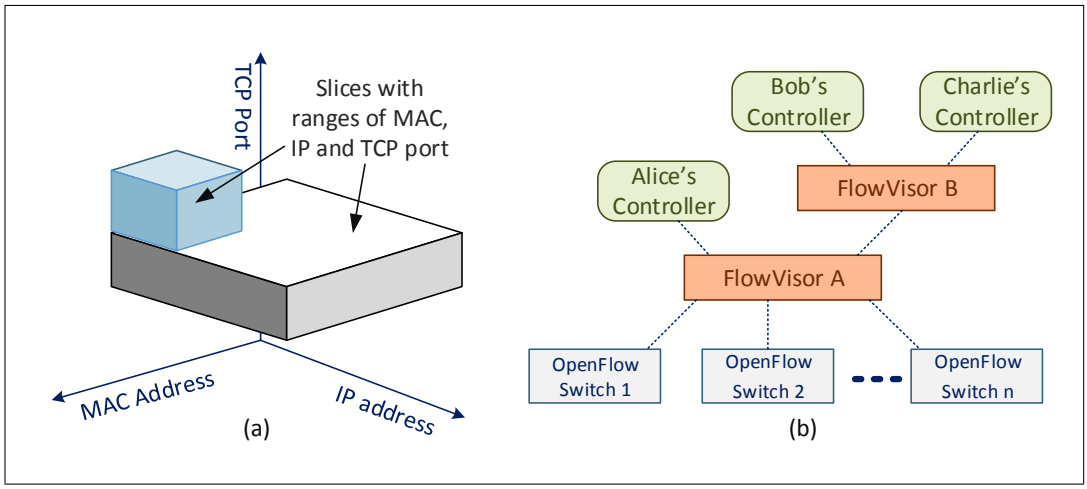


Figure 2.8: (a) Simplified illustration of slicing in 3-dimensional FlowSpace. (b) FlowVisor acts as a transparent proxy between software defined networking controllers and switches allowing nested FlowSpace partitioning [57]

Since FlowVisor uses OpenFlow protocol to control substrate network switches and communicate with OpenFlow controllers, it acts as a transparent proxy between SDN switches and controllers. It maintains the consistency in network state by intercepting control messages sent by each controller and modifying them to avoid conflicts and interference with other slices. Hence, as shown in Figure 2.8(b), FlowVisors can be nested to construct fine-grained slices without compromising traffic isolation characteristics.

2.4 Cloud Resource Management Platforms

With the popularity of the cloud service model, several open-source and commercial cloud service management platforms have emerged to address growing demands. Amazon Web Services (AWS) [58], Google App Engine [59] Microsoft Azure [60] are some of the major enterprises that offer commercial cloud services. OpenStack [61], Nimbus [62] and Eucalyptus [63] are some of the popular open source cloud management platforms within academic and industrial communities. Among major commercial cloud platforms, Amazon has the highest market share (47.1%) followed by Microsoft (10.0%) and Google (3.95%). Out of open source cloud management platforms, OpenStack has gained con-

siderable attention and has been widely adopted by cloud service providers. Reasons for this include an explanation of OpenStack architecture, and some of its major functional modules are presented in following section.

2.4.1 OpenStack Architecture

OpenStack is a combination of open source software modules, developed as individual projects originally by Rackspace and NASA. Configurable modular implementation allows entry level and professional implementations to have a preferred number of compute nodes with varying hardware capacities. Figure 2.9 illustrates the basic building blocks of OpenStack cloud management framework and their interactions.

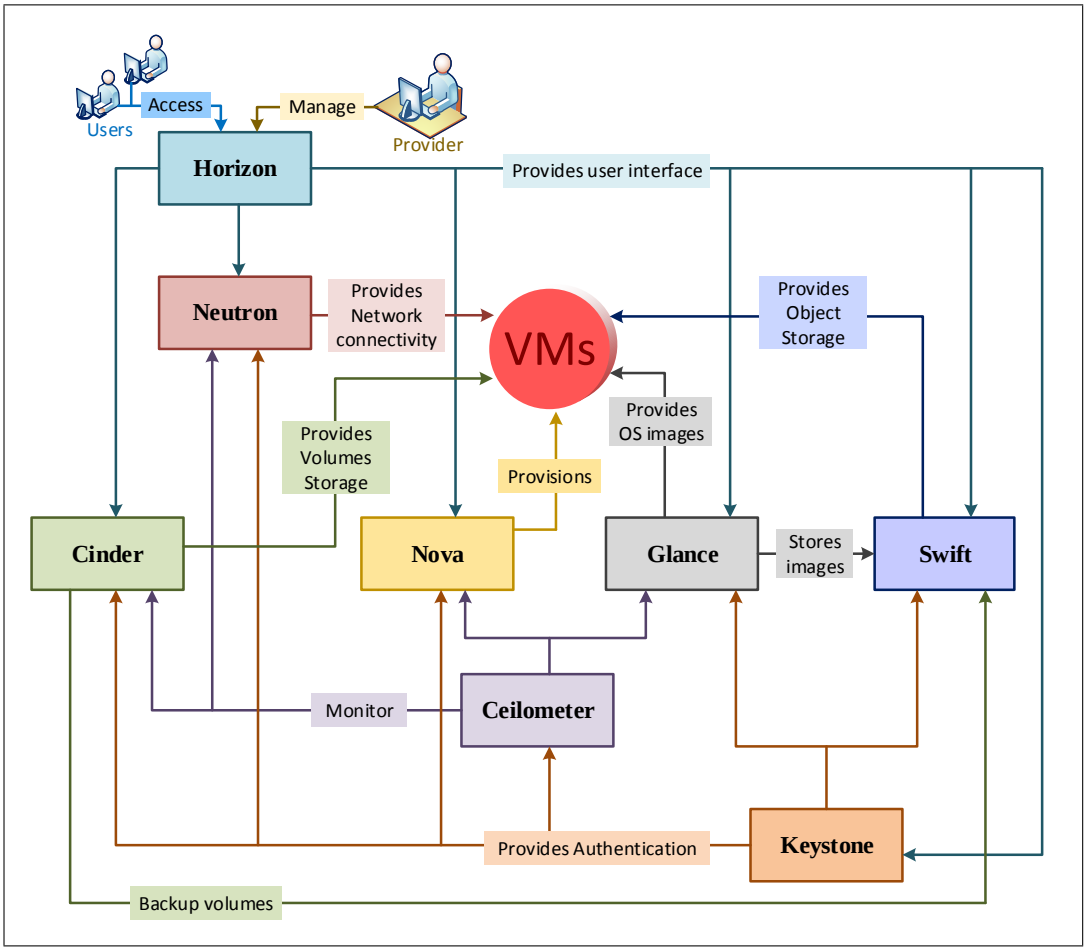


Figure 2.9: Functional modules and their interactions in OpenStack [64]

These functional components can be mainly divided into four categories: compute,

network, storage and shared services. Nova and Glance are associated with compute functions, Neutron handles network functions, Swift and Cinder manage storage resource and Keystone, Cinder and Horizon provide shared services. In addition, MySQL is used as the configuration and management database and RabbitMQ is used as the inter-process message passing fabric. Functions of each of these components can be summarized as follows:

- *Nova* interacts with compute hypervisors such as KVM [65], XEN [66] and VmWare [67] installed in physical servers through APIs such as Libvirt [68] and create VMs based on user demands. Nova is also capable of providing basic network functions such as internet connectivity and exposes an API to user applications.

- *Glance* is the VM image service that manages a library of operating system images to use with newly created VMs. It provides an API to discover, register and retrieve images.

- *Neutron* is a configurable and extensible network service that is capable of providing network as a service. Neutron also provides tools for dynamic host configuration (DHCP), subnetting and routing, virtual LANs (VLANs) and supports integration with open-source and vendor specific tools such as OpenFlow controllers and middle-boxes.

- *Swift* is a distributed and highly available object storage.

- *Cinder* is a block/volume storage that provides virtual hard-drives to VMs. Cinder also supports volume snapshots and backups required for services that demand high fault tolerance.

- *Keystone* plays a major role in OpenStack security and modular integration by maintaining authentication, authorization and access policies for roles (e.g. service provider, users, operators) and services. User/applications are required to obtain access tokens from keystones to interact with APIs.

- *Horizon* is the dashboard that provides a user interface for virtual resource and infrastructure management.

- *Ceilometer* is a usage measurement tool that provides metering and statistical data for billing and analysis.

Beside these main modules that are present in early versions of OpenStack, recent releases (e.g. OpenStack Kilo) contain software components such as: telemetry service called Ceilometer for monitoring and measuring cloud usage for of billing, orchestration service called Heat, bare-metal provisioning toolkit named Ironic, Big Data as a Service named Sahara and Database as a service known as Trove. Compared to other open source cloud management platforms, OpenStack provides a highly configurable modular framework. However, it has several limitations in delivering network as a service and VN resource management. In this thesis, a cloud management platform has been proposed that demonstrates how SDN-based advance virtualization can be integrated with cloud management frameworks to provide better control over traffic flows to users.

2.5 Mobile Cloud Computing

With the popularity of mobile applications and innovations in mobile and wireless technologies, use of smart mobile and wearable devices (e.g. phones, tablets, smart watches, wearable fitness trackers, augmented reality glasses) are becoming a normal part of our day-to-day activities. However, limitations in mobile devices call for alternative solutions to execute compute intensive tasks [69]. Some of these challenges in mobile devices can be summarized as follows;

- *Lack of resources:* Mobile devices have limited CPU, memory and disk capacity compared to static computing devices.

- *Finite source of energy:* Mobile devices have limited battery power. With the increased use of applications that use high CPU/GPU, sensors, cameras and wireless

technologies, mobile phones draw high current from batteries. Therefore power limitation is an important factor to consider.

- *Unreliable connectivity:* Mobile devices use wireless technologies to connect to the internet. Highly varying bandwidth, delays and packet losses as well as error rates are inherent challenges in wireless networks, resulting in additional unreliability in services.

- *Heat and other physical limitations:* Due to form factor limitations, mobile devices typically do not contain fans, and larger heat dissipation surfaces. Also, to improve water and dust resistance, they are designed without ventilation openings. As a result, long lasting processing tasks can easily overheat mobile devices. These mobile CPUs are typically designed for shorter bursty work loads and require external processing for such tasks.

Mobile cloud computing (MCC) [70] emerged to address these limitations by providing computing resources outside the device. It allows mobile users to dynamically request infrastructure, platform and software services from service providers in pay-as-you-go manner over the internet. By offloading compute intensive tasks to mobile cloud, mobile users can experience reduced mobile network and battery consumption, less overheating, less slowing down of the device and reduced use of storage and memory. Beside these direct advantages, applications can maintain device hardware independent backend, perform aggregated data analysis (e.g. big-data processing), and provide better interaction with other application services (e.g. social media) and technologies.

2.5.1 General Architecture of Mobile Cloud Computing

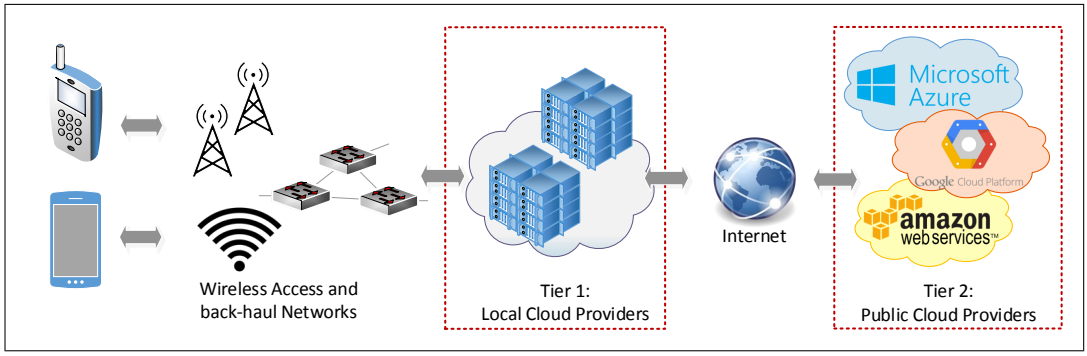


Figure 2.10: Functional modules and their interactions in OpenStack

Figure 2.10 shows the high-level two-tier MCC architecture [71]. Mobile devices can obtain rich services through the LTE network or Wifi. There are two types of cloud services, the cloud services offered by local cloud providers (Tier 1), and public cloud providers (Tier 2). Tier 2 public clouds typically have a larger pool of computing resources as they are built to reap benefits from economies of scale. However, they are typically built far from populated areas and thus may have high communication latency. On the other hand, relatively smaller Tier 1 clouds are placed within close proximity to mobile end users. These clouds are typically operated by wireless service providers and may have access to radio network information (e.g. signal strength), location and bandwidth. The main advantage of acquiring cloud services from Tier 1 is lower latency. However due to small scale datacenters and facility infrastructure, Tier 1 clouds generally have limited computing resources. As well, due to the mobility of end users, these clouds can result in highly varying latency based on network distance. The following section presents some notable research in MMC domain that focuses on resource placement QoS and service mobility aspects.

2.5.2 Notable Mobile Cloud Architectures

In the recent literature of MCC, several cloud resource placement strategies have been proposed with different objectives. Among them, accessing cloud services with minimal latency is central to most mobile applications as it has high impact on Quality of Experience (QoE) for end users. However applications that do not mainly depend on round trip time to deliver high QoE (e.g. cloud storage, email, mobile commerce) are able to rely on large centralized datacenters. Most of the existing research work that proposes to use centralized cloud data centers to offload and execute processing tasks is designed to save battery power and reduce execution time. For example, recent work such as MAUI [72] and CloneCloud [73] propose offloading certain parts of applications to clouds to conserve battery power in mobile devices. Besides inexpensive computing resources, large centralized datacenters offer a sizable pool of highly available resources with a wide range of infrastructure, platform and software services. Also high bandwidth available in inter-datacenter networks allows application service providers to deploy services covering wider geographical areas and integration with other application services.

On the contrary, mobile ad-hoc clouds [74–76] deliver a plausible alternative for high latency-related issues caused by large centralized datacenters. Additionally, cloud connectivity over WAN imposes a number of communication bottlenecks for interactive, real-time and multimedia applications [77]. The concept of mobile ad-hoc clouds is motivated by the fact that the majority of personal as well as organizational mobile IT infrastructure and devices (e.g. laptops, tablets, smart phones) are not always fully utilized and have idle resources that can be shared with other devices while in areas where internet connectivity is of poor quality or expensive. Short range wireless technologies such as WiFi direct, Bluetooth and device-to-device in mobile cellular are typically used to form an ad-hoc network. In an ad-hoc cloud where a client-server architecture is applied, participating mobile devices may act as client nodes and server nodes. Once compute resources of participating service nodes are discovered, the client node can of-

fload tasks fully or partially to service nodes for processing. Server nodes may provide cloud service in return for incentives pre-defined by the ad-hoc cloud platform.

Ad-hoc cloud computing is characterized by its unique features such as: lack of trust, lower interference, volunteering resources, and diverse workload [78]. Despite reduced communication delays, ad-hoc cloud suffers from limited resources and unreliable resource availability durations due to abrupt joining and leaving of nodes from the ad-hoc network. Thus ad-hoc clouds prove to be less effective with heavy and complex work loads. In addition, in certain cases, task break down and distribution adds considerable overhead to client devices. In [78], authors highlight several limitations in mobile ad-hoc clouds such as:

- Heterogeneity in wireless mobile devices
- Risk of losing data and partially processed tasks due to node churn
- Highly dynamic connectivity(varying signal strength, error rates and bandwidth)

Among MCC proposals, Mobile Edge Clouds(MEC), Fog computing, and Cloudlets are considered promising solutions to overcome aforementioned limitations in centralized and ad-hoc clouds.

Cloudlets

Cloudlets [79, 80] introduced a cloud resource placement strategy with the objective of minimizing communication latency by placing resources close to the user. Also placing cloud resources close to user allos providing highly available cloud services over high bandwidth access network. Cloudlets can be placed at wireless access points, base stations and aggregation points as shown in Figure 2.11. The size and resource capacity of each cloudlet can vary from a couple of servers to several racks, based on demand and selected placement location. Thus both location and capacity planning are done while considering factors such as number of users in the covered area, variations in demand and bandwidth and latency requirements [81].

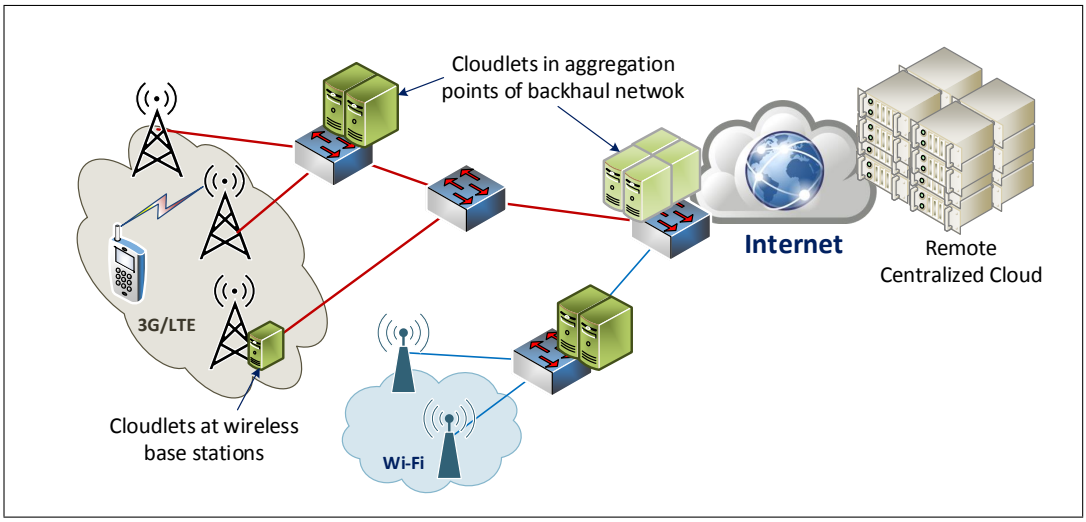


Figure 2.11: Cloudlet: cloud resource placement close to mobile user

One of the main advantages of cloudlet is the hierarchical design which allows heavy tasks to be pushed into larger cloudlets at aggregation points or directly to public clouds. However, in comparison to centralized clouds, recent research [82] demonstrates that cloudlets deliver better performance results only when the maximum number of cloudlet hops is two. Otherwise, it cannot be guaranteed that cloudlets outperform large centralized clouds. Also as argued by the authors in [22, 83], cloudlets are owned and managed by mobile end users. Thus they do not have access to radio network parameters such as signal strength, bandwidth and other network conditions that interconnect wireless links.

Fog computing

Fog computing was initially proposed by Cisco [84], which termed it as a new cloud computing paradigm that extends to edge of the network. With the wide popularity of Mobile devices and where-able devices (smart watch etc.), there is a significant growth in number of devices connected to internet. Further more, researches predict that the numbers will grow exponentially with the increasing use of sensors and actuators [85, 86]. Fog computing allows computing at the edge by putting cloud servers along with edge routers (or by replacing with servers with routing functionality). Computing and storage

resources from these edge servers provide cloud services to resource poor edge devices such as wireless access points, set-top-boxes, sensors, actuators, smart home devices and wearable computing.

Mobile Edge Computing

Mobile Edge Computing (MEC) [22,83,87,88] has received wide recognition as a technology that can effectively bring the benefits of the cloud service model to mobile devices. MEC is capable of satisfying stringent QoS requirements of highly interactive and real-time mobile applications. MEC proposes to place miniature datacenters comprised of one to a few servers at the radio access network (RAN). These datacenters can be placed directly at the cellular base-station. Beside delivering cloud services to delay sensitive mobile applications, MECs can improve content delivery by caching remote server data at the receiver edge.

From the perspective of mobile network operators, MEC servers can reduce the burden of increasing network traffic over back-haul (transport) and core networks by moving services to the network edge. Mobile application service providers gain access to close proximity computing resources with reduced latency, high bandwidth and more reliability. On the other hand, mobile end users benefit from richer applications and high QoE. Finally, cloud service providers are able to reduce workload at remote centralized clouds and also reduce notwork utilization costs.

European Telecommunication Standards Institute (ETSI) act as the standardizing body for MEC and has defined a set of distinguishing characteristics for MEC framework [89]. These includes

- Computing resource that can be accessed locally
- Close proximity to users
- Reduced response time due to low latency, which lead to better user QoE

- Applications that can gain access to user location details to provide customized services

- Availability of RAN information in real-time

ETSI further exemplifies several deployment models of MEC servers along with different mobile service provider infrastructure, as shown in Figure 2.12.

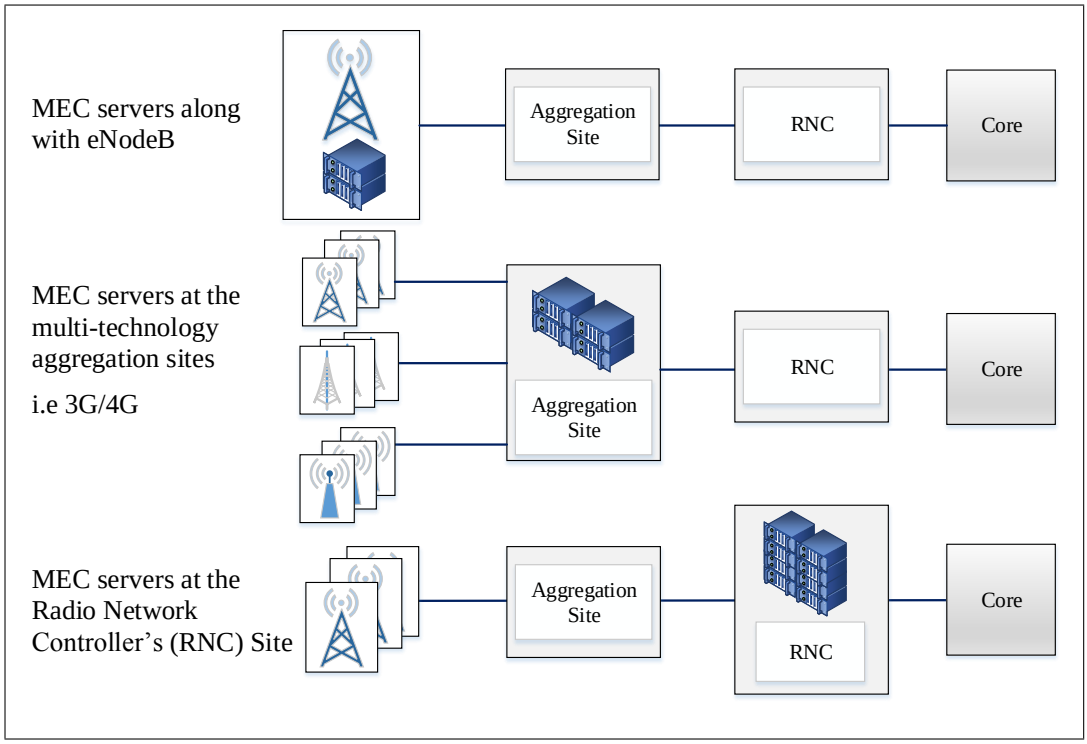


Figure 2.12: Mobile edge computing server placement options in different mobile network infrastructure sites

As illustrated in Figure 2.12, MEC servers can be vertically positioned in different layers in wireless back-end network hierarchy such as at eNodeBs (base-stations), aggregation sites and/or radio network controller (RNC) sites. Capacity planning and suitable location of MEC servers for each segment of the network and each set of users need to be determined considering a number of factors including infrastructure costs specific to each site (e.g. cost of space, electricity and cooling expenses, security), connectivity (e.g. latency, available bandwidth, packet drop-rates), accessibility (distance and connectivity

to other MEC servers and centralized clouds), number of mobile users and application demand predictions.

As mentioned previously, placing cloud resources closer to mobile users does provide a number of advantages. However, the inherent mobility nature of mobile users can result in a higher variation of QoS parameters. Thus further on in this thesis, detailed analysis is given on how existing how existing mobile cloud service frameworks attempt to overcome this challenge. Furthermore, by identifying several limitations in existing approaches, we have proposed an SDN based seamless service migration technique is proposed. It minimizes QoS interruptions while improving resource utilization as well as cloud provider revenue.

2.6 Chapter Summary

In this chapter a comprehensive study of cloud computing has been presented, with the main focus on various aspects of centralized network management. Initially a formal definition of cloud is given, along with attention to the basic cloud architectures, service models, and different types of cloud deployments. Of the three cloud service models presented, focus is directed towards IaaS, particularly the network connectivity of cloud model. In addition, details are given on the role of virtualization as key-enabling technology in cloud computing. Attention is also given to how resource discovery, allocation, and operation are done in network virtualization. Currently research in the area of network virtualization has escalated due to the introduction of SDN paradigm. Background on SDN is given as the capabilities of SDN are used extensively throughout this thesis to perform cloud network resource allocation, operation, mobility management and fault tolerance. As well in this chapter, relevant information on cloud resource management platforms and mobile cloud frameworks is provided.

3 Resource Management Framework for Cloud Datacenters

With the popularity of the cloud model, the number of businesses that rely on cloud infrastructure services for their compute, storage and network resource requirements has steadily grown during the last decade. However, the lack of control over infrastructure resources remain as one of the strongest opposing forces to this trend. In this chapter, a novel cloud resource management framework capable of allocating resources to IaaS requests with rich VN control features has been proposed. The framework integrates network virtualization functionalities offered by SDN to provide flow level traffic control to IaaS consumers while improving datacenter resource utilization.

First, an analysis is given to network virtualization capabilities of existing cloud management platforms, by conducting a test-bed implementation of the widely used OpenStack cloud management system. OpenStack is one of the well designed and popular open source cloud resource management platforms available. Second, through the implementation of OpenStack, several limitations have been identified in existing network resource management strategies which are used in IaaS clouds. Finally, to address these limitations, a cloud datacenter resource management framework, which integrates network virtualization functionalities offered by SDN into cloud echo-system is presented.

3.1 Deploying Virtual Networks with OpenStack

3.1.1 Software Defined Networking-based Virtualization in Cloud Management Systems

Within the three layers of the cloud service model, IaaS has become highly attractive for application service providers due to the degree of freedom and the control it provides to fine-tune computing infrastructure based on application needs. Delivering such customized service without interfering with coexisting users was entirely possible due to recent advances in computing and networking infrastructure virtualization. Virtualization allows infrastructure providers to partition as well as combine physical resources to construct logically-isolated custom resource units. Such custom resource allocation which is permitted by virtualization can potentially improve datacenter resource utilization. Also, virtualization is generally used to conceal complexities of implementation in the underlying layers. Thus, in cloud resource management platforms such as OpenStack, hypervisor-based virtualization technologies are widely adopted to manage compute resources.

The success of compute resource virtualization can be mainly attributed to the operational simplicity provided by well-defined abstraction mechanisms. However, computer networks have evolved by introducing new protocols to deliver new service, thereby increasing design and operational complexity. A number of researchers have pointed out that the lack of flexibility, dynamicity and controllability of interconnecting VNs impede the ability of business to rely on cloud services. Network virtualization therefore demands powerful abstraction capabilities to continue advancements while keeping the operational complexities hidden. Existing network virtualization technologies alone, such as VLAN and VPN, do not provide adequate solutions.

SDN, with its de-facto standard application OpenFlow, is seen as a promising technology capable of addressing these needs. SDN serves to configure/route traffic flows

dynamically and flexibly to satisfy user application requirements. OpenFlow's success is largely due to its wide adoption by industry and network equipment manufacturers. OpenFlow is capable of abstracting underlying hardware and allows fine-grained control over network traffic paths. Hence, particularly in an IaaS cloud environment, OpenFlow can deliver highly customizable virtualized network infrastructure. It is logical to envision that a comprehensive engineering solution that automates on-demand provisioning of virtual infrastructure with flow-level control over network traffic will attract more businesses towards cloud services. A framework with such characteristics is capable of not only enhancing user application performance but also improving resource utilization for IaaS providers. Existing commercial IaaS platforms such as Amazone AWS, Google App service and Microsoft Azur do not provide automated IaaS provisioning capabilities with flow-level control over network traffic to VN users. However, open source cloud management platforms such as OpenStack, support integration of SDN technologies into cloud echo-system. Thus as the initial step, we evaluate the support for SDN-based network virtualization in the OpenStack cloud management system through test-bed deployment.

3.1.2 Limitations in OpenStack Neutron

Cloud networks must support dynamic and efficient deployment of VMs to satisfy varying demands and elasticity requirements. Dynamic nature imposes significant stress on network management as it has to perform fast and robust deployment of network services to satisfy cloud user/application requirements. OpenStack exposes direct open networking API through Neutron networking module to cloud provider for the purpose of creating and managing VNs dynamically.

Before introducing Nutron (also known as Quantum in its early days) OpenStack had a simple flat networking structure, provided by Nova network module. Through such primitive networking, it was next to impossible to satisfy growing application QoS demands. With Neutron, OpenStack was able to extend its network capabilities to functionalities such as adding interfaces, assigning ip addresses, sub-nets and deploying basic

gateways/routers. Neutron uses three software agents to perform basic network functions: (i) neutron-dhcp-agent to provide DHCP services to tenant networks, (ii) Neutron L3-agent to provide layer 3 network address translation (NAT) forwarding required for internet connectivity to VMs and (iii) plug-in-specific agents to configure virtual switches in each hypervisor. Besides the above basic functions carried out by agents, Neutron is composed of plugin modules to provide additional networking services. For example, Multi-Layer 2 (ML-2) plugin is designed to implement some of Layer 2 switch functions.

However multiple plugins make installation and modular integration of OpenStack Neutron a tedious task. Also, as pointed out by a number of web surveys [91,92], Neutron presents a number of obstacles to resiliency and scalability of OpenStack deployments. This is mainly because Neutron has only limited support to layer 3 routing and relies extensively on routing capabilities of Linux kernel. Even in a broad scale OpenStack deployment with a large number of applications, VNs and tenants, only a single L3 agent is expected to handle multiple services including floating IPs management, traffic routing and network address translation. Though it is possible to use multiple L3 agents, its highly complicated. Therefore the L3 agent becomes a bottleneck in Neutron network service. Furthermore, even with the support of ML-2, Neutron still supports only a limited set of switches. Due to these reasons, Neutron alone is not sophisticated enough to perform advance network services such as dynamic deployment of virtualized network functions (VNF) and traffic engineering.

3.1.3 OpenStack Implementation with Software Defined Networking Controller

Industrial and academic communities who in the past adopted OpenStack and other cloud management systems were quick to realize that limitations in cloud networking services can hinder application performance and deployment capabilities. More sophisticated network traffic control strategies were needed to dynamically deliver virtualized services such as Network Function Vitalization (NFV) and service function chaining,

along with traffic isolation and inter-domain networking. Vendor specific technologies and protocols that provide traffic management functionalities have proven to be ineffective as they limit interoperability and migration capabilities between cloud providers as well as enterprise and cloud provider sites. For these reasons, OpenStack has provided integration capabilities to several SDN projects. From this group of supported projects, open daylight(ODL) [51] is one of the most popular SDN controllers with wide industrial and open source community support.

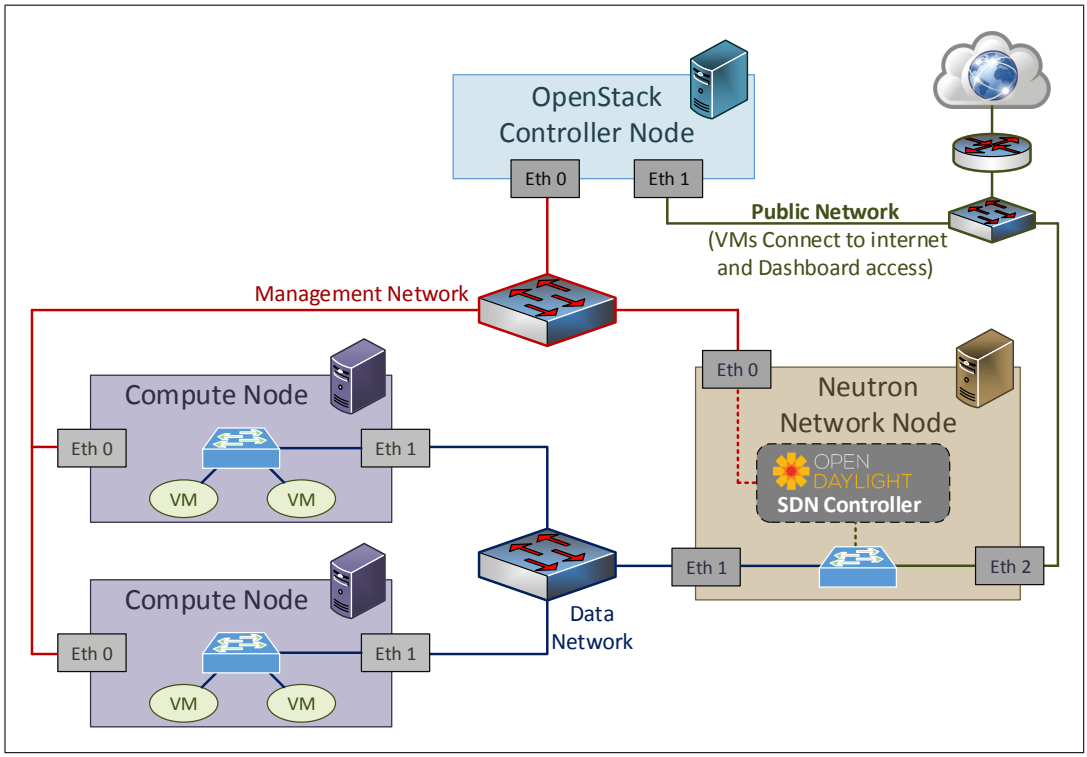


Figure 3.1: OpenStack test-bed to designed to evaluate virtualization capabilities

In this thesis, OpenStack cloud with ODL integration has been deployed to study network virtualization capabilities in existing CMS, and identify how each VN resembles in-site enterprise network. To support this, a test-bed composed of four servers interconnected with three traditional switches was constructed. One server was used as OpenStack controller and two servers were used as OpenStack compute nodes. In the fourth server, an OpenStack Neutron with ODL controller was installed, in which Neutron is connected to ODL through ML-2 plugin. Management and data networks are

constructed with two Layer 2 switches and the third switch is used with a traditional router to provide internet access to VMs deployed in compute nodes as shown in Figure 3.1.

In the above deployment, OpenStack controller executes most of the core modules including Nova, Horizon, Keystone and Glance along with supplementary services such as database and messaging queue. Horizon provide access to the cloud management system (through a graphical web dashboard and an API) to the cloud manager as well as consumers. Cloud manager can set resource quotas for each consumer to create VMs based on their requirements. Compute nodes run mainly KVM hypervisors and contain Neutron and Nova client modules. VMs are instantiated in compute nodes as per OpenStack controller instructions. Neutron network controller manage virtual switches in compute nodes to implement access police for VM interconnections and internet access.

With the integration of ODL with OpenStack, changes to network elements and traffic flows are triggered by cloud users and user applications. Such requirements are translated into Neutron API calls, which are transferred to OpenDaylight controller through Neutron agent and REST API. OpenDaylight maintains a Service Abstraction Layer (SAL) and communicates with both virtual and physical switches through OpenFlow southbound protocol. Figure 3.2 illustrates ODL integrated architecture of OpenStack Neutron [93, 94].

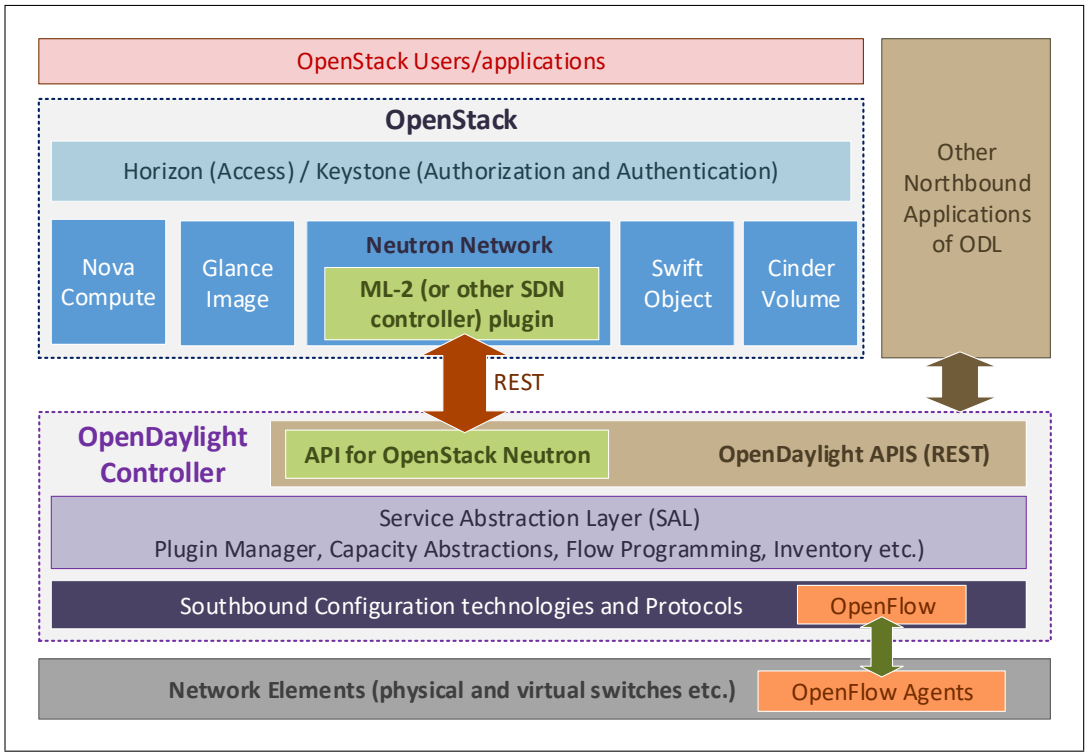


Figure 3.2: OpenStack as a northbound application to OpenDaylight controller

Integration of ODL SDN controller into OpenStack Neutron excludes the need of managing a distributed network protocol stack to provide connectivity and facilitates centralized management. It eliminates Neutron L3 agent bottleneck by abstracting routing and switching operations into much simpler flow rules that can be installed in forwarding plane devices. By translating traffic management policies of each cloud tenant into flow rules that can be installed proactively on forwarding devices, load on centralized ODL controller can be significantly reduced. Due to centralized management, monitoring, configuration and virtual/physical network management tasks are simplified. SDN also supports dynamic placement of middle boxes and other VNFs as flows can be redirected through them without tearing down or suspending VNs. Furthermore, OpenFlow based southbound API eliminates the need of using vendor specific APIs and configuration channels.

3.1.4 Limitations of Current Software Defined Networking Integration and Benefits of Network Control Virtualization

SDN controller such as OpenDaylight does provide a number of advantages in cloud network services. However, based on the experiences of OpenStack/ODL implementation as previously described, several limitations have been identified in the current Openstack/SDN integration approach. Particularly, it has been observed that even with ODL controller integration, Nutron does not provide sufficient abstractions for cloud consumers to independently operate their VNs as their own enterprise networks. Some of the current enterprise networks are quite complex and employ multiple technologies to manage bandwidth, switching/routing, security, QoS and other communication requirements coming from application and organizational objectives. Hence cloud consumers as well as cloud service providers can benefit form more powerful virtualization capabilities that SDN can offer through network hypervisors. Potential benefits of SDN-based cloud network virtualization can be summarized as follows:

- Allow cloud users to manage traffic through their own SDN controllers during runtime
- Improved application performance through application driven dynamic traffic policies
- Reduced network management overhead on the cloud management system from application driven networking
- Better traffic and performance isolation
- Improved infrastructure utilization through resource sharing and aggregation

From the perspective of application service providers and cloud consumers, more dynamic application triggered traffic reshaping can significantly improve application perfor-

mance. As well, for enterprise customers, more granular control over traffic can increase their odds on relying on cloud network services for their enterprise network needs.

3.2 State of the Art in Cloud Virtual Network Services

In this section, some comparable research on cloud network IaaS is evaluated. Focus is on the traffic-flow control features available for users in each approach and the measures taken to guarantee isolation and QoS.

Meridian [96] introduces a network service model that allows cloud users to create and manage virtual topologies in order to execute complex workloads. Users are able to construct logical topologies that suit their infrastructure needs using the API and GUI tools provided. An underlying network orchestration platform performs logical-to-physical translation and converts API calls into appropriate device configuration commands. It also provides network-wide services to applications, and coordination and mapping requests in the network. However, the authors do not attempt to give users a high degree of control over network traffic. Even simple traffic-flow modification requests need to go through cloud controllers, network controllers and planner modules before they can be physically configured.

Network Reservation System [97] uses a central network controller approach to construct on-demand VNs. The authors attempt to address both intra-datacenter and inter-datacenter connectivity requirements by employing OpenFlow and overlay technologies respectively. A layered architecture and a topology model facilitate the construction of automated VNs. Isolation between VNs within a datacenter is achieved using VLANs while VPNs are used for inter-datacenter communication. The authors highlight the importance of a QoS guarantee and consider a possible future extension. However, they do not explore a way to grant flow-level control over the VNs.

J. A. Wickboldt et al. [98] emphasize that existing cloud management platforms

lack the network management support and flexibility to express and control resource allocation requirements. The authors propose HyFS, a solution that explicitly considers network resources to be a part of a cloud platform that includes computing and storage resources. Their proposed framework allows users to demand a complex, feature-rich, virtual infrastructure through a flexible and programmable API. SDN/OpenFlow is used to implement robust networking functionalities within the cloud environment. Compared to existing frameworks, HyFS is distinguished by its tight integration of VN support. Nevertheless, HyFS does not grant users with traffic-flow control over their networks or illustrate how traffic isolation is guaranteed.

CloudNaaS [99] highlights the importance of providing users with a high degree of control over their VNs. It allows users to deploy applications with rich network functions such as custom addressing, service differentiation and flexible middle-box interposition. The authors introduce a network policy language and a network controller that respectively specify VN requirements and configure the VNs on a physical substrate. Tenant networks are isolated by creating VLANs. Bandwidth QoS is assured by pre-configuring forwarding queues in switches and dynamically selecting them based on Type-of-Service (ToS) bits in IP packet headers.

While this thesis shares similar motives, the work differs from that of CloudNaaS in several ways. First, CloudNaaS only provides users with a set of high-level policy constructs to define network requirements. These high-level policies do not provide capabilities to manipulate traffic flows dynamically between end-points. Second, this thesis' proposed architecture employs multiple layers of abstractions. This allows the IaaS provider to improve resource utilization using resource management techniques such as VM migration and over-subscription.

3.3 Proposed Software Defined Networking-based Cloud Resource Management Framework

In This section, an SDN-based infrastructure management framework for IaaS cloud platforms have been introduced. The framework employs benefits of SDN based network virtualization to overcome limitations of existing cloud network services. The proposed framework has four broad design considerations. First, deliver a higher degree of control over traffic on composed VNs to consumers and application service providers. Thus cloud applications are able to dynamically adjust network connectivity and QoS parameters to suit varying application requirements and user demands. Second, the management complexities were reduced by automating service delivery and explicitly integrating network resources, along with computing and storage resources, into the cloud platform. Third, management overheads were reduced and focus was placed on fast service delivery. Fourth, resource utilization was improved through composed resource allocation.

To meet these objectives, following were carried out:

- Designed a datacenter resource management architecture that facilitate automated and on-demand provisioning of compute and network infrastructure.
- Integrated SDN based network virtualization into cloud IaaS resource provisioning, by abstracting datacenter infrastructure into a unified pool of virtual resources.
- Employed Ontology reasoning to improve efficiency of resource discovery process and adopted semantic-similarity and closeness-centrality based on composition technique to reduce complexity of VN embedding process while improving substrate resource utilization.
- Implemented the proposed framework on (a) physical and (b) emulated test-beds and evaluated performance and efficiency of proposed composition technique

The composition algorithm was used to select virtual resources to satisfy on-demand IaaS requests. In the prototype testbed, OpenFlow protocol was used as the SDN hardware abstraction layer. In addition, FlowVisor [53], OpenFlow controllers and related tools were used to virtualize network infrastructure in a datacenter. After a successful resource allocation, users gain access to a compute resource controller and a subset of OpenFlow controller functions, allowing them to manage their virtual compute and network resources.

3.3.1 Datacenter Resource Abstraction

The proposed framework employs multiple abstraction layers to facilitate virtualization and mask complexities in physical compute and networking infrastructure as illustrated in Figure 3.3.

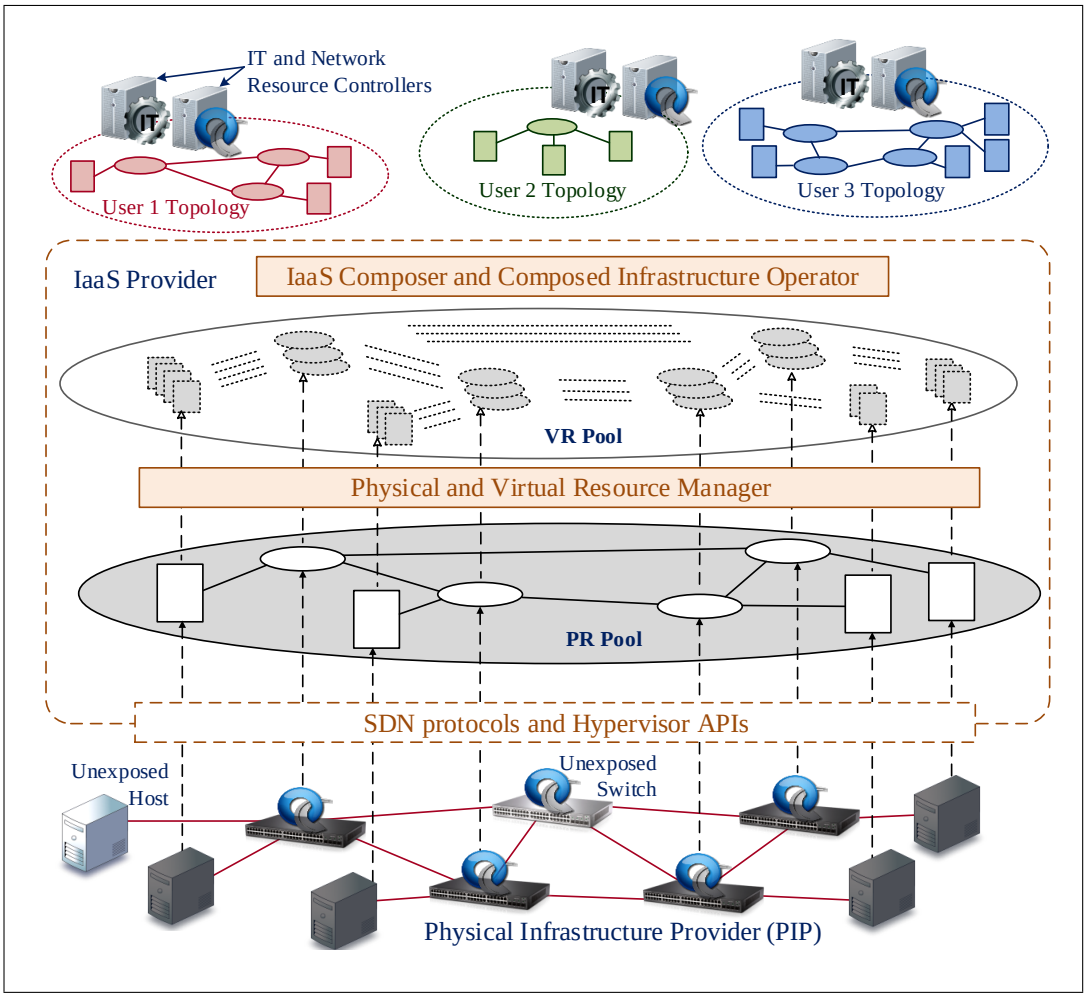


Figure 3.3: Abstraction of Datacenter Resources

Physical Infrastructure Provider (PIP) manages the physical datacenter resources and is responsible for maintaining connectivity at the physical level. PIP provides the IaaS provider with access to physical resources through an abstraction layer formed with OpenFlow protocol and hypervisor APIs. With available information, IaaS provider maintains a logical view of the physical infrastructure in a local ontology known as physical resource pool (PR pool). IaaS provider also manages a second ontology, virtual resource pool (VR pool), by partitioning compute and network resources in the PR pool. The consistency between PR and VR pools are maintained by the physical and virtual resource manager. Finally the IaaS composer and composed infrastructure operator jointly act as a layer of abstraction between user requests and virtual resources.

3.3.2 Virtual Resource Composition

Several existing proposals that address resource allocation for IaaS [100, 104] have paid more attention towards guaranteeing bandwidth than improving underlying resource utilization. Thus they result in less efficient use of datacenter resources and high blocking of IaaS requests. To overcome these limitations, a composition approach known as 2P-IaaS [101] proposed by another PhD student from our research lab has been used to allocate resources for IaaS requests. The proposed composition approach employs a unified ontology model along with reasoning capabilities to semantically represent diverse datacenter infrastructure and to discover suitable resources for each request. After discovering suitable resources, individual resource matching values are calculated for each discovered resource using a technique known as semantic similarity evaluation [102]. Semantic similarity evaluation increases the opportunity to find matching resources for IaaS requests.

Minimizing the cost of interconnecting matching substrate resource, in order to satisfy IaaS requests, is regarded as one of the key operational objectives of IaaS providers. Closeness centrality [103] is a technique which measures the topological distance between a specific node and all other nodes in a network by associating a relative numerical value based on measured distances. Higher closeness centrality value infers the node in consideration is in the center of the network and smaller values infers the node is in the edge. Hence closeness centrality values are employed for efficient virtual path composition, ensuring minimum proximity between discovered resources.

3.3.3 System Architecture

The main functional components of our system architecture and the ways in which they interact are shown in Figure 3.4. The framework seeks to improve IaaS request acceptance latency by discovering available physical resources and populating physical and virtual resource pools in advance. Before processing requests, IaaS provider reads avail-

able physical infrastructure information from PIP, through compute virtualizer and the network virtualizer modules. These two modules employ abstract communication protocols and APIs to retrieve limited but sufficient information from PIP. The limits guarantee the privacy of PIP and mask vendor-specific information and other physical layer complexities.

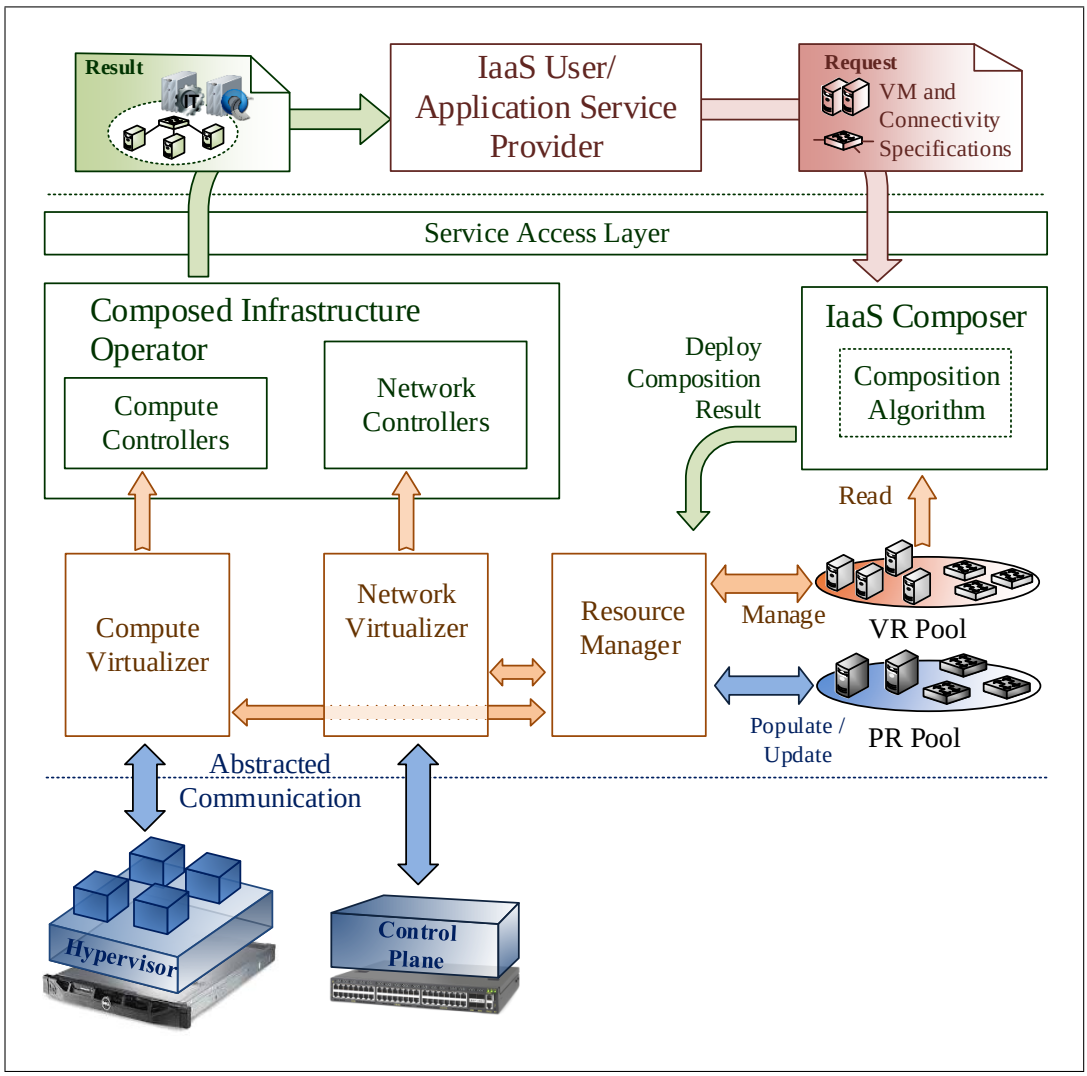


Figure 3.4: System architecture of the proposed infrastructure-as-a-service management framework

Two ontology-based resource pools maintained by IaaS provider, the PR pool and the VR pool, aid to sustain consistency during physical to virtual resource mapping. Initially, the PR pool is populated with the physical resource information discovered by

compute virtualizer and network virtualizer.

These two repositories are dynamically updated by:

(a) acceptance and allocation of new requests

(b) de-allocation of resources when requests are successfully completed, and

(c) modification or failure of physical infrastructure

The implementation and operation of the two pools are further discussed in the next section. When physical compute and network infrastructure specifications are retrieved, a resource manager module processes connectivity details and generates topology, identifying interconnections and relationships in the physical infrastructure. The topology and resource availability is stored in the PR pool. A virtualization algorithm in resource manager then queries the PR pool and populates the VR pool by partitioning physical resources into composable units.

Users can make a request from IaaS provider through a service access layer. It allows users to:

(a) monitor the operation of the composed infrastructure

(b) interact with compute resource controllers to start, suspend and shut down VMs and initiate VN computing (VNC) sessions

(c) communicate with network resource controllers to install, change or delete flow rules and controller applications, and

(d) request VM migration and modify resource allocations.

When a new IaaS request arrives, the service access layer forwards the extracted information to the infrastructure composer. The infrastructure composer then queries the VR pool and searches for resources that match the request criteria. The composition algorithm is then executed, connecting resources that satisfy the IaaS request. Once the

request has been successfully composed, the composer forwards it to resource manager for deployment.

To commence the deployment, the resource manager performs three distinct, parallel tasks:

(a) changes the state of corresponding composable units in the VR pool to represent reservation

(b) requests the compute virtualizer to create VMs, and

(c) requests network virtualizer to create interconnections between the VMs

Resource manager verifies the deployment by signaling compute virtualizer and network virtualizer to update the PR pool. If the deployment is successful, it delivers accessibility information to the composed infrastructure operator. Subsequently, the composed infrastructure operator creates a compute controller and a network controller to provide user applications with a set of API methods through the service access layer in order to manage the infrastructure. These API methods are translated into traffic-flow modification rules and VM management commands at the network controller and compute controller respectively. Thus the end result of a successful allocation is a set of API methods that allow the user applications to manage VMs and dynamically manipulate traffic flows between them. One instance of network controller and one instance of compute controller is assigned to each accepted IaaS request. If the request is rejected, the user is given a response through the service access layer.

3.4 OpenFlow-based Implementation of the Proposed Framework

To verify the operation and evaluate the performance of the proposed framework, a prototype test-bed with OpenFlow supported switches was constructed. Java programming

language was used for the development of most of the management framework. Network virtualizer was implemented as a Java application module that uses FlowVisor API. Each network controller instance allocated to each request is implemented as a python application running atop dedicated POX [48] controller. The framework was tested on both a physical test-bed and Mininet [105] emulated environment. Each VM host in the system was configured with a Kernel-based Virtual Machine (KVM) [65] hypervisor and an open virtual switch (OVS) [106] over a Linux platform. An open source API toolkit known as Libvirt [68] was used for management. Libvirt is also appropriate as an abstraction layer since it conceals the implementation complexities and sensitive security information.

The operation of the framework can be divided into five phases:

1. resource discovery and construction of the PR pool
2. virtualization and construction of the VR pool
3. VM mapping and connectivity composition
4. resource reservation and deployment
5. composed infrastructure operation

This section outlines the techniques and tools used to carry out these phases.

3.4.1 Resource Discovery and Construction of Physical Resource Pool

During resource discovery, resource manager makes two parallel requests to network virtualizer and compute virtualizer, inquiring about the current state and availability of network and compute resources respectively. Switches are uniquely identified from their data-path identifiers and links are identified by end-point switches. Each physical VM host contains an OVS to allow internal VMs to connect with the external data network, thereby forming a virtual access layer of switches in the topology. Thus OVSs act as network endpoints and are abstracted in the same way as physical switches.

The PR pool is designed to maintain a logical replica of the datacenter topology constructed by resource manager. When an OVS is connected to a physical network interface card (NIC) of a host, OVS adopts MAC of the NIC as its DPID. Resource manager uses this property of OVS to map network information obtained from network virtualizer to host information obtained from compute virtualizer.

To construct the PR pool, in this thesis W3C standard OWL-2 ontology was employed since its semantic graph structure is suitable for describing computing infrastructure. An overview of the PR Pool Ontology structure is illustrated in Figure 3.5.

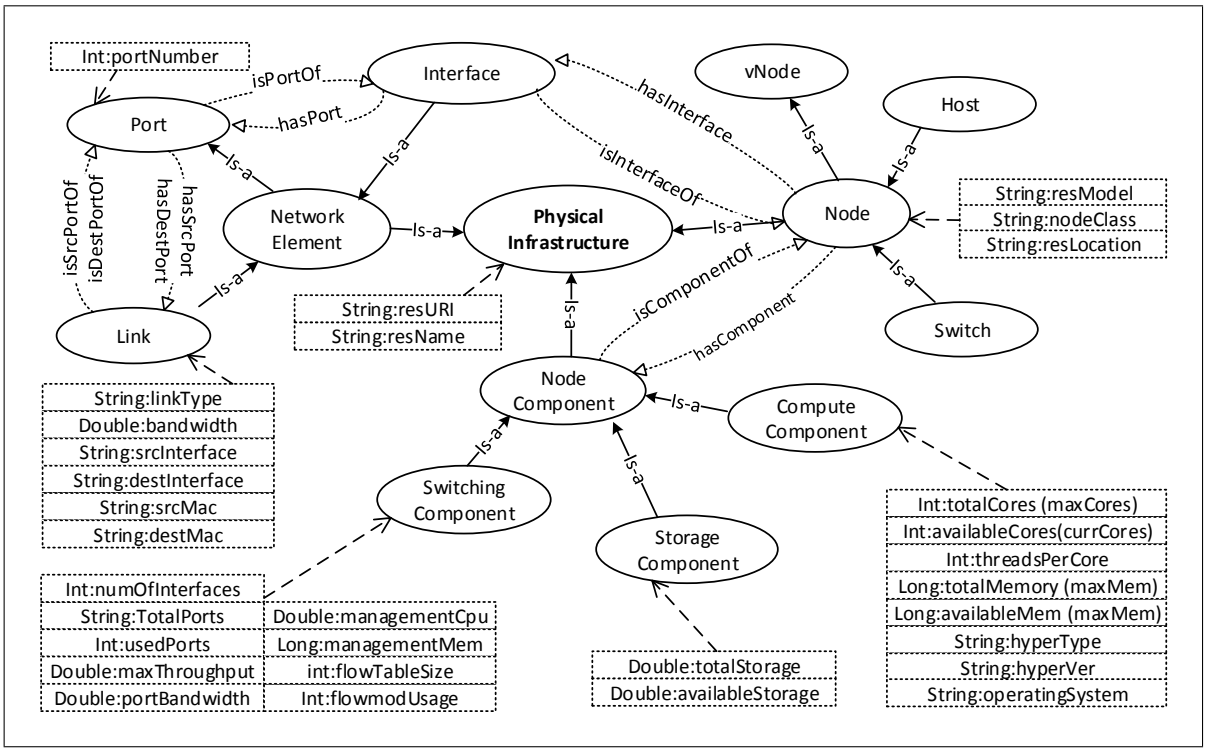


Figure 3.5: Ontology diagram of the physical resource pool

Infrastructure Description Language (INDL) [107], type schema with several modifications has been used to model SDN-based cloud infrastructure. Specifically, port attributes were added with appropriate relations to store OpenFlow port-level information. Also processor and memory components were abstracted into a single ontology class and identified as a compute component, to facilitate cloud integration. INDL is powerful descriptive language that defines resources with properties and restrictions. It also pro-

vides a platform to describe relationships between each resource type as class relations. Furthermore ontologies provide a convenient way to validate and derive relationships automatically through reasoning algorithms. However, as discussed later in this chapter, reasoning operations can be highly expensive in terms of compute resource usage and can delay resource composition. Hence in this research, INDL was used mainly to model resource pools and basic reasoning capabilities were only used to validate data through relationships. Specifically, *OWL_LITE_MEM_RDFS_INF* was used as the ontology model descriptor attribute to specify the use of OWL-Lite modeling with Resource Description Framework Schema (RDFS) inferencer for additional entailment [108].

3.4.2 Virtualization and Construction of Virtual Resource Pool

After the PR pool is updated with physical topology information, the virtualization algorithm in resource manager queries the PR pool and creates composable VR units through partitioning. The VR pool therefore maintains a set of virtual machines, virtual links and virtual switches to be used during VM mapping and composition. These virtual resources were created with the intention of satisfying the input requirements of the composition algorithm. Similar to PR pool, INDL schema was used to model VR pool with resource descriptions and relationships. Physical VM host resources are logically partitioned into VMs with three classes (A, B and C). This ensures that all VMs in the same class contain an equal amount of resources. Class A VMs have the most resources and those in class C have the fewest.

For this work, only static partitioning was employed to decide the number of VMs and resources in each category. Partitioning and aggregation to maximize resource use is a subject for future work. Link bandwidth, switch CPU and switch flow table are considered as critical network resources. Each resource in the VR pool contains a state, together with functional and non-functional characteristics associated with computing, network or storage services. Interconnecting network link information, including bandwidth and delay parameters, is also stored, along with other network device properties.

3.4.3 Virtual Machine Mapping and Connectivity Composition

Once an IaaS request has been received, the composer module extracts the requested end-point and QoS information and executes the composition algorithm. The 2P-IaaS composition algorithm works in two phases: (i) a VM mapping phase, and (ii) a connectivity composition phase. During the VM mapping phase, VMs that satisfy the request are discovered from the VR pool. The objective of the mapping algorithm is to improve the datacenter host's resource utilization by minimizing excessive network use when VMs belonging to the same request are distributed. If insufficient VMs are discovered, the request is rejected without proceeding to the connectivity composition phase. Otherwise, the connectivity composition algorithm is executed. If suitable switches and links are available and sufficient QoS can be ensured, the request is accepted and resources are reserved through resource manager.

3.4.4 Resource Reservation and Deployment

When an IaaS request is accepted, resource manager changes the state of reserved VMs from available to reserved. But for each virtual link reserved, a new virtual link is created with the required bandwidth and deducted from the original bandwidth. Similarly, new virtual switch are created on the original virtual switches by reserving switch CPU and flow table resources. After updating the VR pool, resource manager constructs a deployment plan by defining VMs and FlowVisor slices. Each slice created per request is assigned a VLAN identifier so that the traffic is isolated. Packets generated from VMs belonging to a particular request are tagged with a unique VLAN-ID and are assigned to a corresponding slice.

Bandwidth isolation is achieved by creating QoS queues for switch ports with rate limiters. When slices are created, flow-spaces are formed assigning queue numbers using force-enqueue in the add-flowspace method in the FlowVisor API. Also in this research, switch resources were isolated through the corresponding allocation of switch CPU and

flow-table to each slice. For this, initially the PR pool was populated with flow-table and management plane CPU information of each switch. During slice creation, the percentage of control plane CPU was assigned using rate-limit and the percentage of flow table using flowmod-limit.

3.4.5 Composed Infrastructure Operation

Once the deployment has commenced, the system architecture allows the PR pool to be updated to verify the deployment and to deliver accessibility information to composed infrastructure operator. It in turn allows compute controller and network controller to manage VMs and traffic flows respectively. However, in the prototype, since network controller information must be given when slices are created, a pool of network controller and compute controller instances were maintained by the composed infrastructure operator. The final output of a successful IaaS request is therefore a VM management interface and a set of network virtualizer API methods that manages traffic flows in the VN.

3.5 Performance Evaluation

To evaluate the framework's performance, set of experiments were carried out using both a physical test-bed and Mininet. For the physical experiments, our test-bed consisted of 9 physical VM hosts, 3 physical access layer switches, 2 core layer switches and a gateway, as illustrated in the topology diagram shown in Figure 3.6(a).

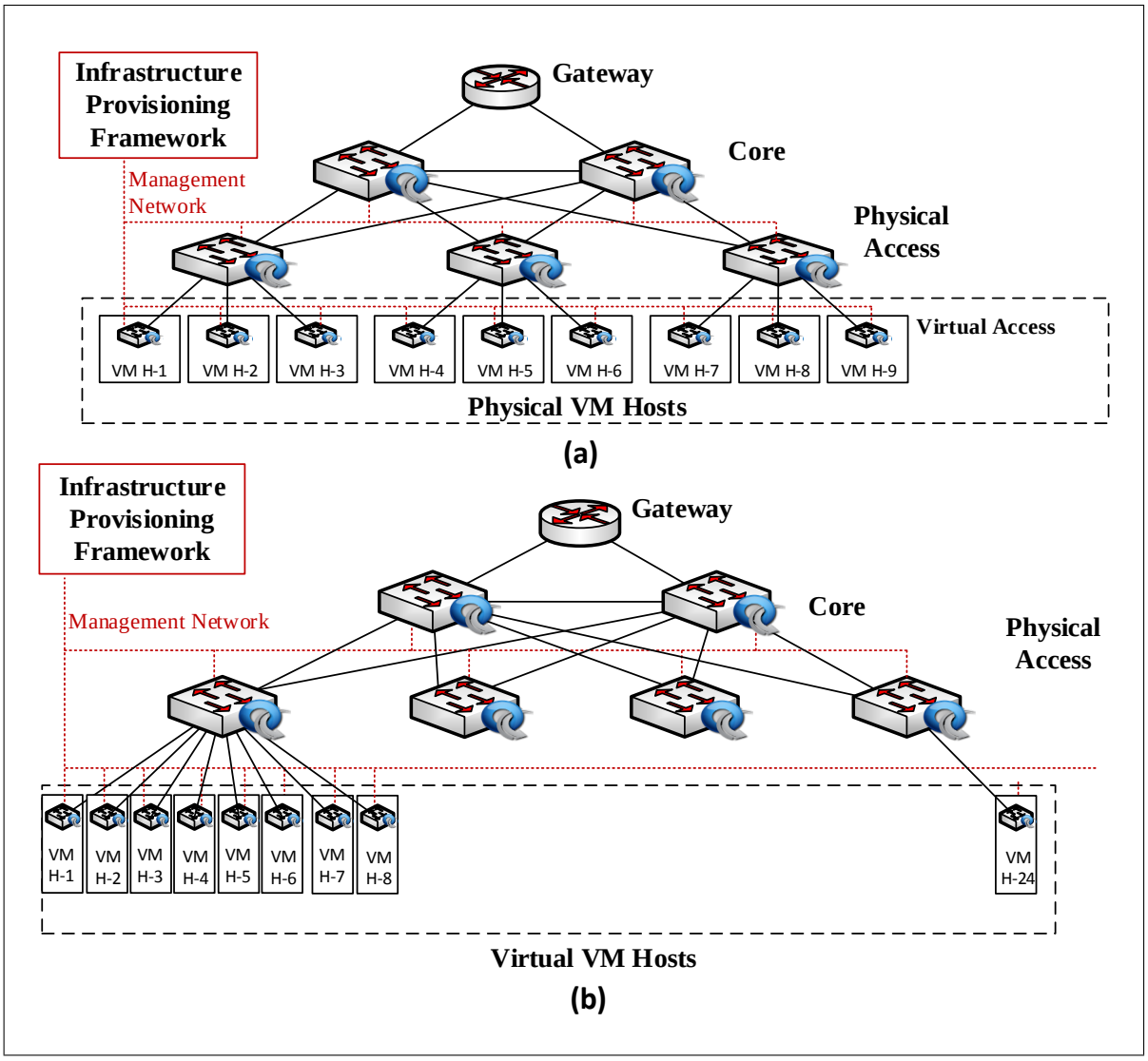


Figure 3.6: Network topology of (a) physical and (b) virtual testbeds

Servers and switches were interconnected to form a fat-tree topology with 1Gbps blue and white colored Ethernet cables. A photograph of stacked switches of in-lab prototype test-bed is shown in Figure 3.7. For physical core and access switches, OpenFlow-supported Pica-8 P-3290 switches. An OpenFlow supported high-throughput HP-E5406 modular switch was used as the SDN supported gateway (also customer premises equipment) to the connect to internet through Cisco 2611XM legacy router. The router essentially isolates the test-bed from the campus network. All 5 Pica-8 switches (2-core and 3-aggregation) were physically stacked on top of the gateway SDN switch. Blue

color Ethernet cables were used for data network that interconnect these switches with each other and with VM host servers. White color Ethernet cables were used for the management network, that connects management ports of all switches and host servers. A legacy L2 switch was used to switch management network traffic.

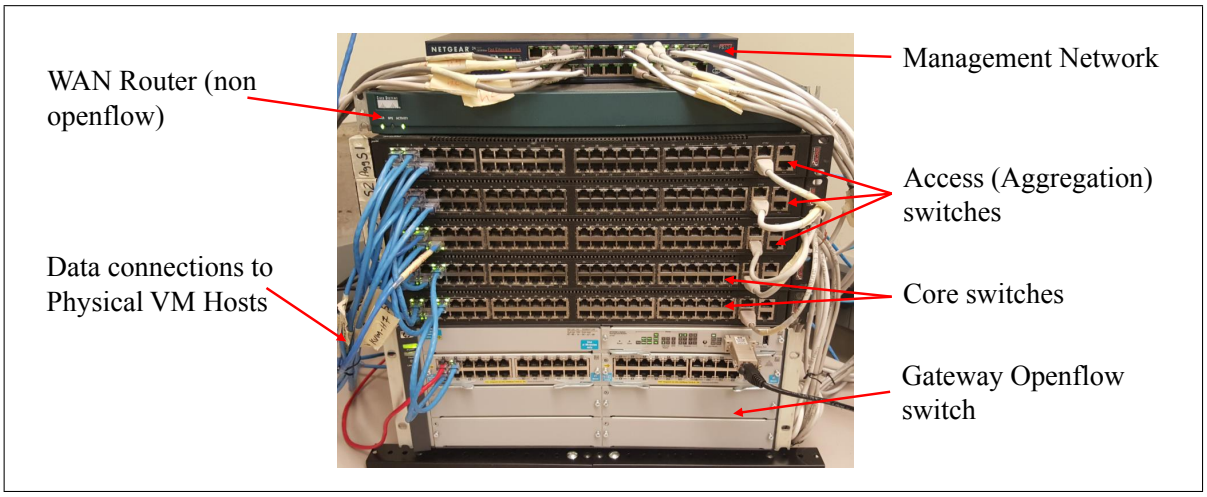


Figure 3.7: Switches and connections of the OpenFlow test-bed, physically deployed in the lab

The test-bed consisted of physical compute nodes with a range of processing, memory and storage resources. Ubuntu-server 14.04 LTE was used as the host operating system. For testing purposes, VMs created from the framework were booted from live versions of Lubuntu 14.04 and Tiny-core version 6.3 operating systems.

In the initial design, physical VM hosts were categorized into three resource classes loosely based on resource availability. When constructing the PR pool, these classes were used to partition physical resource into respective classes of VMs, to match composer requirements. To implement the management components of the framework, six work stations were employed. One was used to execute a Java-based cloud management application and another to run FlowVisor. The remaining four machines ran POX controllers. A hub-connected out-of-band management network was used for all configurations, monitoring and other management operations.

An emulated test environment was also implemented using Mininet to compare the performance of the physical test-bed and to analyze the scalability of the framework. It

consisted of 32 virtual hosts created with VirtualBox [109], with 6 switches, as shown in Figure 3.6(b). Mininet and the VirtualBox were both executed on a Dell PowerEdge R720 server equipped with 2 Intel Xeon E5-5910 processors and 196 GB of RAM. Performances of the main operational phases were evaluated using: (i) Management overhead, (ii) Scalability and (iii) Request service time. The performance of the resource discovery and request deployment phases was evaluated on both physical and Mininet test-beds. The performance of the virtualization and composition phases was analyzed using data collected for Mininet test-bed in order to satisfy scalability requirements. Performance results and the IaaS allocation framework presented in this chapter are also published in [90].

3.5.1 Resource Discovery and Construction of Physical Resource Pool

The scalability of the approach was initially quantified by measuring the management overhead of the PR pool population phase, according to the number of VM Hosts connected to the system. In this phase, resource manager signals compute virtualizer to discover VM host resources and FlowVisor to discover network resources over the management network.

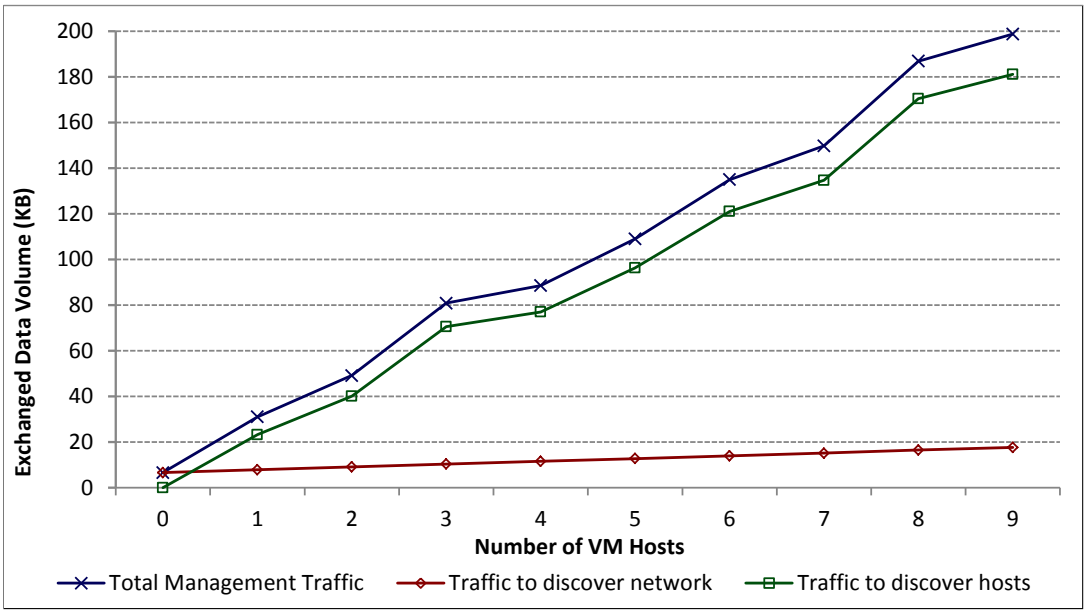


Figure 3.8: Exchanged data volume for resource discovery in management network

VM hosts were physically connected and disconnected to access switch ports and obtained traffic measurements for 10 iterations. The traffic volume was measured using the IPTraf tool [110] and separated into FlowVisor and hypervisor IP addresses. The average values appear in Figure 3.8.

Management network overhead varies linearly with the number of VM hosts. When a new host is added, an increase in traffic can be anticipated. An increase in overhead can also be expected since, with the new VM host, a hypervisor virtual switch is also added to the system. However, VM host discovery overhead varies in a higher gradient than the network overhead. This variation is also predictable since host resources have higher granularity than switch resources.

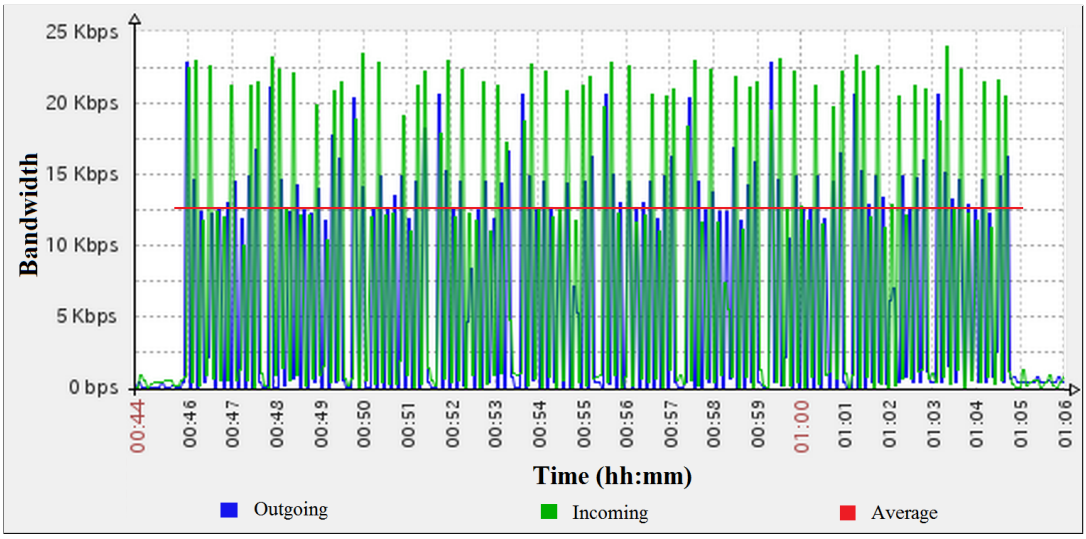


Figure 3.9: Average bandwidth usage during resource discovery in the management network

Bandwidth use was measured with a standard open-source resource monitoring toolkit known as Zabbix [111]. The bandwidth used to obtain resource information about the physical topology (9 hosts and 5 physical switches) was measured for 10 iterations. The incoming and outgoing bandwidth use, including average use, is shown in Figure 3.9. This graph shows that the framework needs on average less than 15 Kbps to discover physical resources. The results in Figures 3.8 and 3.9 demonstrate that management network overhead during the resource discovery phase is minimal.

PR pool construction involves three aspects: (i) compute resource discovery, (ii) network resource discovery, and (iii) ontology update. The time taken to obtain compute resource and network resource information were measured in the discovery process for both physical and Mininet topologies. Results are shown in Figure 3.10, in terms of the number of hypervisors added to the test-bed.

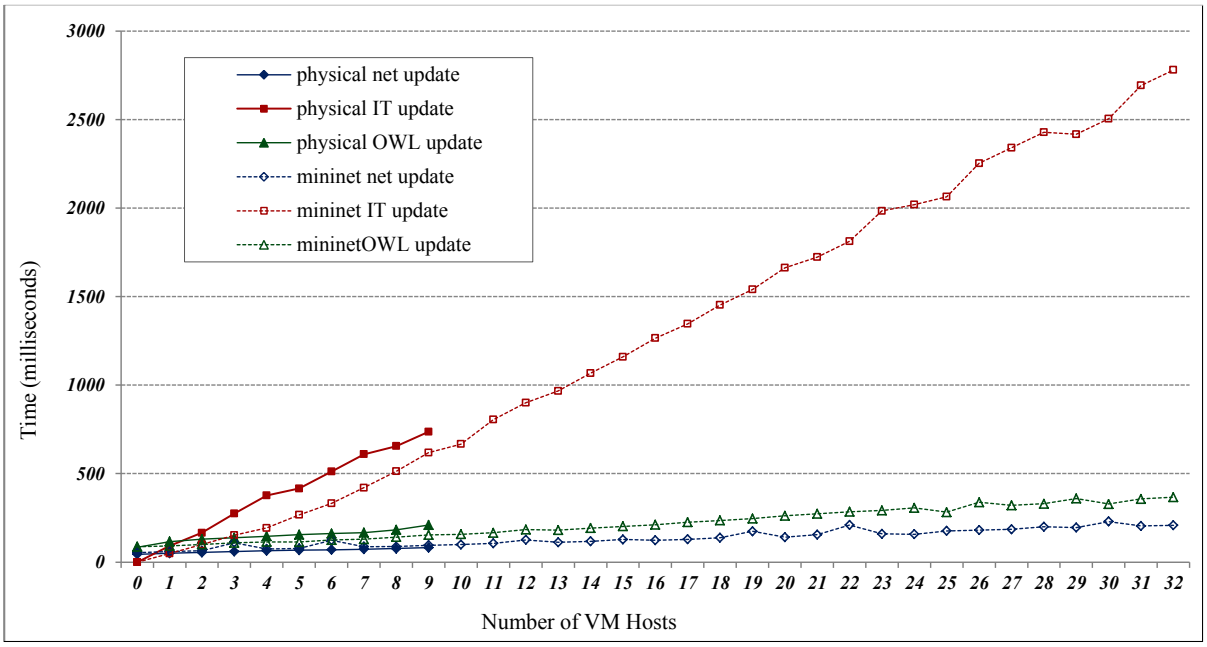


Figure 3.10: Physical resource discovery time

The graphs also show the times taken to update PR pool ontology for both test-beds. Since there are limited servers in physical test-bed (i.e. 9 VM hosts), physical result graphs are displayed for 9 VM hosts. Both topologies take a similar amount of time to obtain network resources and ontology update times are comparable in both cases. However, the physical resource update consumed relatively more time than Mininet update. It was determined that the difference was mainly due to delays added by the physical network interface cards. This was verified by an experiment comparing the time taken for a small Libvirt call between two virtual and physical hosts.

3.5.2 Virtualization and Construction of Virtual Resource Pool

For compute resources, the virtualization process involves partitioning physical host resources and creating VMs. For network resources, one virtual resource was created for each physical resource, to include all available resources. The time needed to virtualize both compute and network resources in terms of the number of virtual resources created was measured. The results are shown in Figure 3.11, together with the time to update the VR pool according to its size. The total time to construct and update the VR pool

can therefore be calculated from the graph by adding virtualization and OWL update times. For a large data-set, a PR pool populated by resources from Mininet topology was employed. This allowed analysis of the virtualization and composition algorithms.

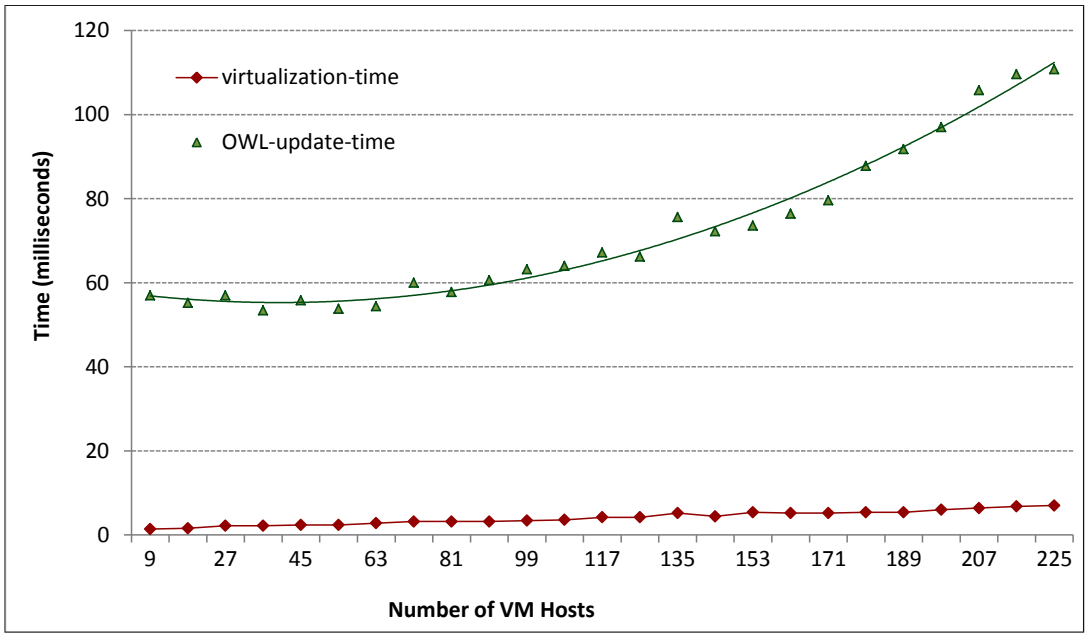


Figure 3.11: Virtualization and virtual resource pool update time

The graphs show that the time to virtualize resources varies linearly with a small gradient. However, ontology update time increases in polynomial time with respect to the number of elements in the VR pool. By extrapolation, it was observed that around 600 milliseconds were required to update 1000 virtual resources into VR pool. Hence for massive datacenters with a large amount of operational free resources, the ontology update will add considerable overhead. However, under normal operating conditions in small-scale datacenters, with moderate use of datacenter resources, the total time needed to update the VR pool will be tolerable. Moreover, since both the PR pool and the VR pool were updated after successful deployment of a user request, the time required to update the VR pool does not directly affect the user experience.

3.5.3 Virtual Machine Mapping and Connectivity Composition

Once an IaaS request arrives at the composer, the 2P-IaaS algorithm initially discovers and maps host resources (end-point VMs). Connectivity composition follows. The VM mapping function was evaluated and measured the time taken to calculate semantic similarity and closeness centrality scores for each requested resource. In order to analyze the performance of the VM mapping function under different load conditions, continuous high-life-time requests were made in order to saturate the system. Three sets of requests were created with 3, 5 and 9 VMs per request with 100 requests in each set. The measurements are shown in Figure 3.12, according to the sequential arrival of requests.

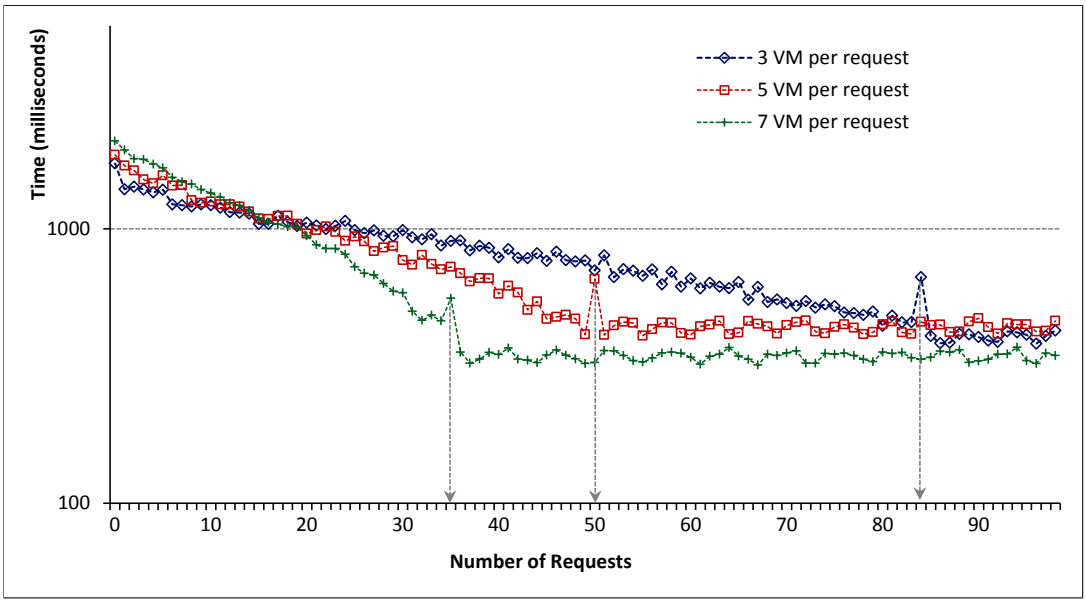


Figure 3.12: Time to discover suitable end-point virtual machines

The vertical axis of the graph is represented in logarithmic scale to highlight important observations in the results. It can be seen that the performance of the VM mapping function varies significantly with the number of available resources in the VR pool. Since the experiment was started with a large number of resources in the VR pool, the mapping function initially took more time to find the most suitable VMs. This is due to the processing time taken by large data-sets returned from ontology queries. When the number of available resources decreased, the time taken by the mapping function de-

creased logarithmically. This demonstrates the theoretical complexity values presented in [14], i.e. O(n log n) for querying and sorting resources, and O(n) for the evaluation of semantic similarity and closeness.

Once suitable resources are exhausted, the mapping function starts to reject incoming requests. Rejections were noticed for requests with 3 VMs from the 85th request and for requests with 5 VMs from the 51st request. A sudden increase was observed in the time consumed by the discovery algorithm at the point the rejections started (marked by a dotted-line arrow in the graph). This is due to additional exhaustive search queries made by the program with relaxed variables. The results of these additional search queries are kept in data structures to prevent repetition in all following requests with similar requirements. With this improvement, the time taken by the mapping function was minimal and roughly constant after the rejections started.

After mapping VMs, the 2P-IaaS algorithm builds interconnections between VMs, a process also known as connectivity composition. The time taken by the connectivity composition algorithm to progressively discover and compose network paths was measured and the results are shown in Figure 3.13.

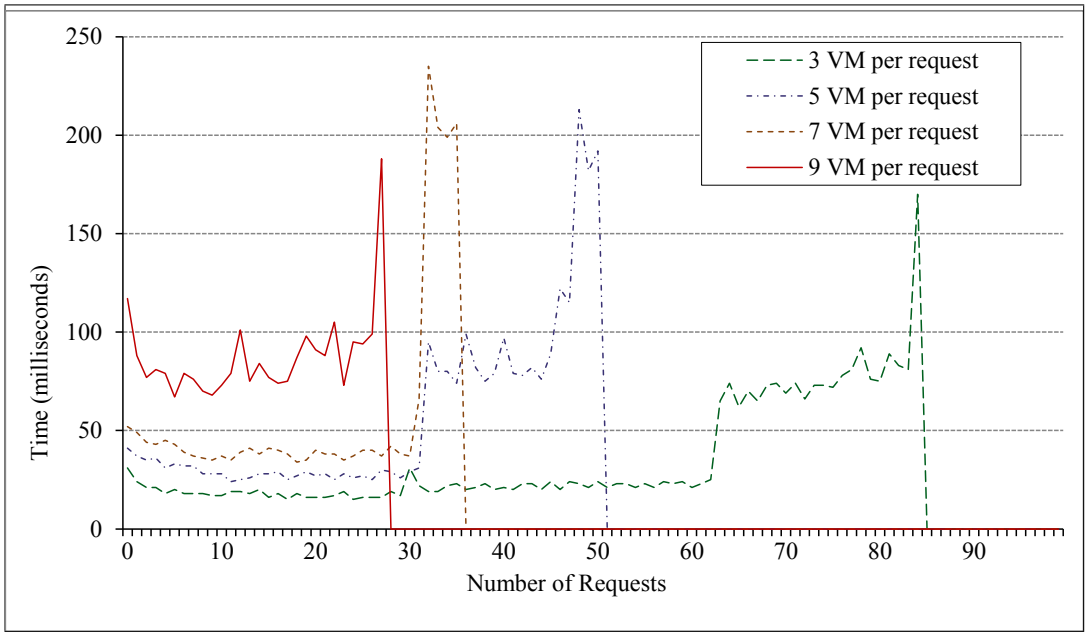


Figure 3.13: Time taken by composition algorithm to interconnect end-points

As previously noted, the objective of the VM mapping function is to improve host resource use while minimizing the resource use caused by the distribution of VMs belonging to the same request in different hosts. However, when sufficient VMs are not available in the same host, a topologically closer host is selected, resulting in an increased complexity of path computation. This can be seen in the graph as sudden increases in the composition time. Particularly for 5VM requests, initial composition time is constant around 28 milliseconds for the first 32 requests. Afterwards, composition time increments were observed in distinct stages roughly corresponding to 80mS, 12mS and 200ms until the rejections starting from the 51st request. Composition time was seen to increase when the number of VMs per request increased. This can be explained by the increase in processing requirements needed to interconnect a higher number of end-points.

3.5.4 Resource Reservation and Deployment

The deployment of successfully composed requests involves creating disk images, defining VMs and deriving flow rules to create slices and flow-spaces through FlowVisor. A comparision was made of the time required for each of these sub-tasks on both the physical and simulated test-beds. The simulated environment was constructed using virtual-box VMs with resources equivalent to the physical test-bed. These virtual hosts were interconnected through Mininet, which was topologically equivalent to the physical test-bed. The results were plotted according to the number of requests.

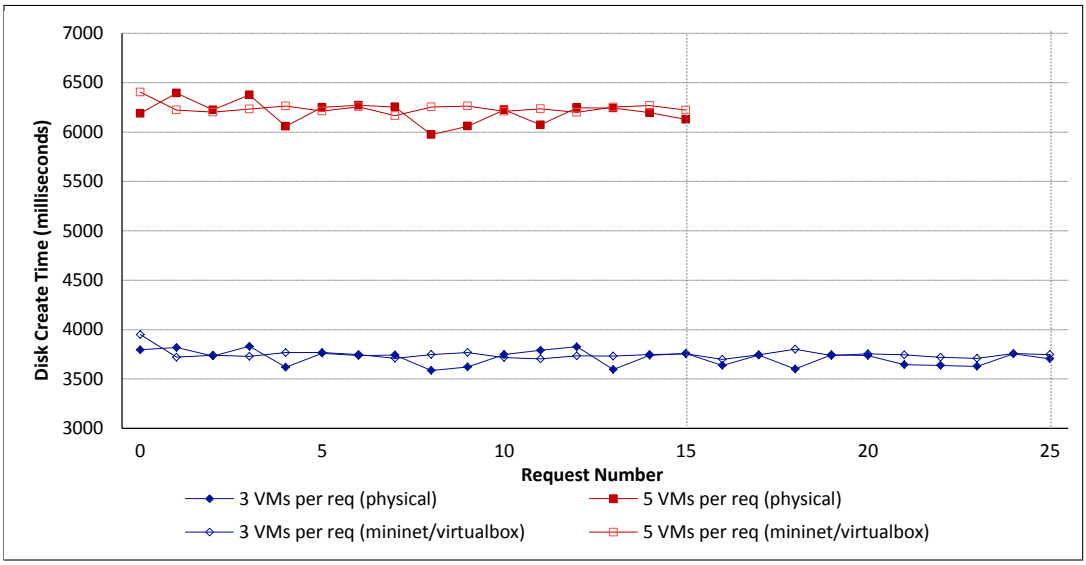


Figure 3.14: Time to create virtual hard disks

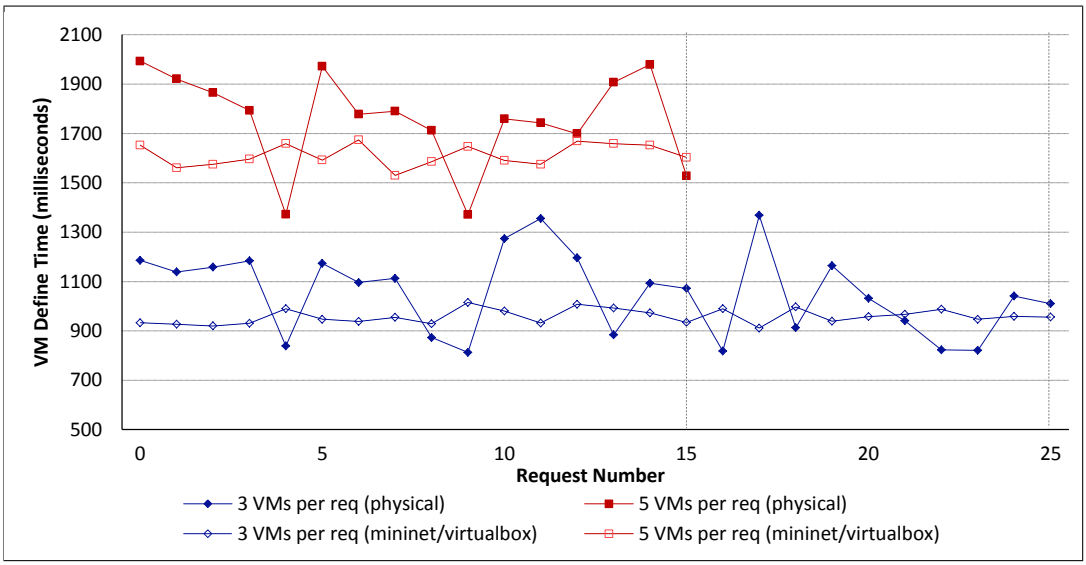


Figure 3.15: Time to discover suitable end-point virtual machines

A disk image for a VM is created by making an image creation request to the corresponding host's Libvirt agent over a secure shell. The time between request to response for each VM in accepted 3VM and 5VM requests was measured. The results in Figure 3.14 show that the time taken to create disk image files for both physical hosts and virtual hosts was equivalent. Although disk creation time is relatively independent of the underlying hardware platform, it ws discovered that the VM creation times are highly

hardware-dependent, as shown in Figure 3.15. As mentioned before, physical test-bed consisted of heterogeneous computing platforms with uneven processing power. Factors such as CPU technologies (Intel Xeon, i7 etc.) and memory speeds play a major role in the results. However, although the simulated platform represents an equivalent number of cores and an equivalent amount of memory to the physical platform, all virtual hosts share the same CPU technology and memory speed. Thus, only noticed minor variations in VM creation time were noticed for the simulated environment.

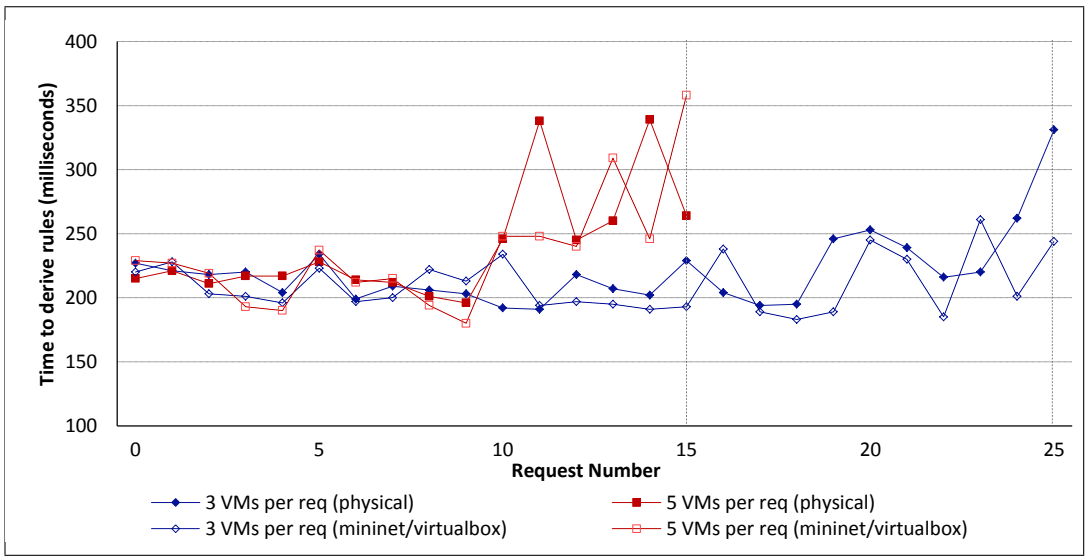


Figure 3.16: Time to create VN slices

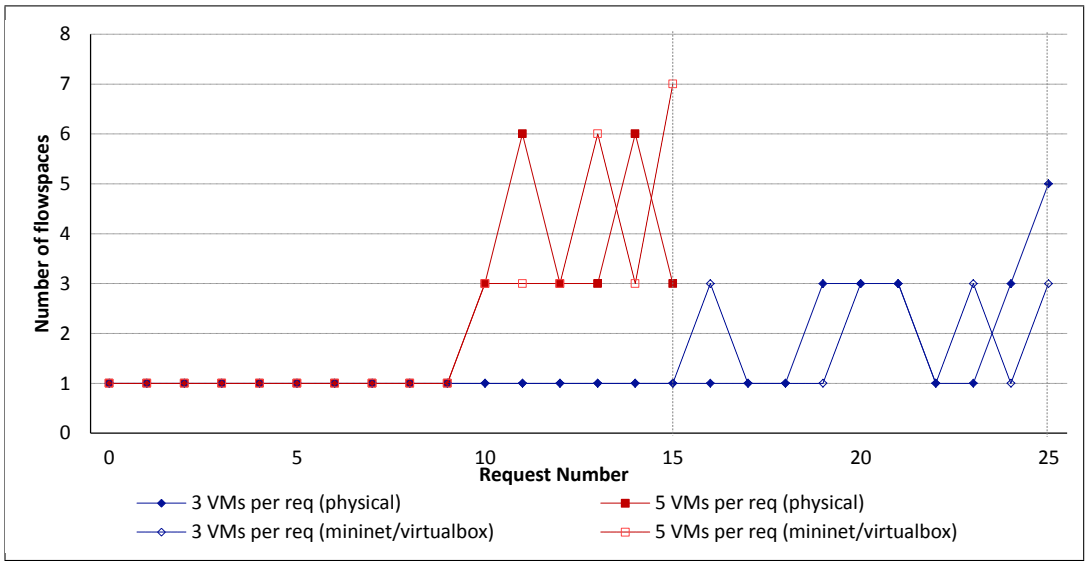


Figure 3.17: Number of flow-spaces created

After creating all end-point VMs in a request, a VN interconnecting end-points is created by deriving and making FlowVisor API calls for slice and flow-space creation. Time measurements for these operations are shown in Figure 3.16 and the number of flow-spaces required for each slice is shown in Figure 3.17. From these results, it is evident that the slice creation time is correlated to the number of flow-spaces. It can be concluded that allocating VMs close to each other for the same request during composition reduces the load over FlowVisor and minimizes delays in slice creation. It also improves resource use and decreases request service time.

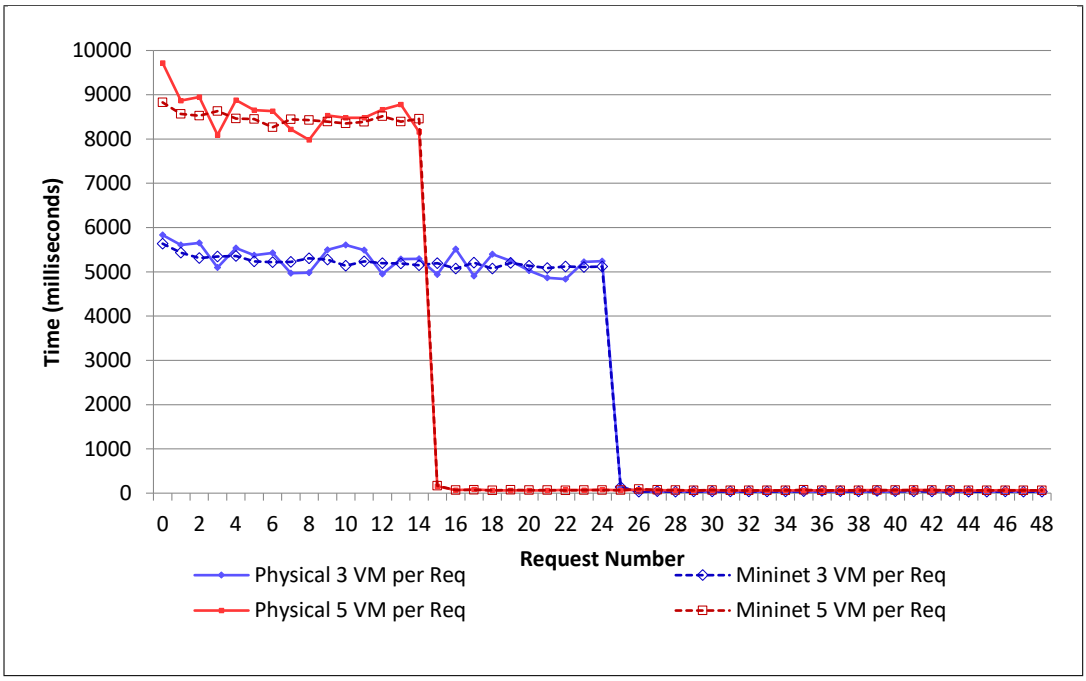


Figure 3.18: Total Time to serve a request from the time of arrival

Total time taken by the framework to serve an IaaS request after arrival was calculated and plotted in Figure 3.18. From this graph, it can be seen that an IaaS requests with a smaller number of VMs can be served faster in both Mininet and Physical test-bed environments. Furthermore, it is evident from the figure that the total service time is highly dependent on VM creation time rather than network slice creation time.

3.6 Chapter Summary

In this chapter, an IaaS resource allocation framework has been introduce that is capable of fully automated service provisioning, Ontology reasoning based VN composition and providing users with control over allocated infrastructure. Initially an evaluation was done on how existing cloud management systems make use of SDN technologies to provide network services to tenants by deploying OpenStack with OpenDaylight on a prototype test-bed. Through test-bed implementation, several limitations were identified in existing SDN integration in cloud management systems and elaborated on how virtualization capabilities of SDN can extend cloud network service offerings. Subsequently, a cloud resource management framework which is capable of delivering cloud IaaS services to users with a high degree of control has been presented. The operation of the proposed framework was demonstrated through a series of prototype experiments with both physical and Mininet test-beds. By comparing the results of these two environments, the efficiency and scalability of the presented framework was demonstrated. Also it was shown experimentally that, in an OpenFlow environment, selecting end-point VMs topologically close to each other can potentially reduce request service times and load.

4 Seamless Service Migration and QoS-aware Relocation for Regional Datacenter Clouds

4.1 Introduction

With the rapid growth in the number of mobile devices, rich mobile applications that are capable of providing a better quality of experience to the user are becoming increasingly popular [116]. However, execution of such applications within mobile devices is challenging due to their inherited resource limitations [117]. In this context, recent innovations in broadband and wireless technologies, as well as availability of flexible and on-demand access to large pools of resources in cloud datacenters has allowed mobile applications to expand their boundaries beyond the limited device resources. As a result, modern mobile applications are increasingly relying on IaaS obtained from cloud datacenters, relieving mobile devices from complex and resource-intensive tasks.

In this chapter, existing mobile cloud architectures have been evaluated and an SDN-based mobility aware cloud service framework is introduced. The proposed framework is designed to acquire IaaS resources from regional edge datacenters, and provide value added services to mobile users. These value added services include QoS guarantees and seamless dynamic service migration in the event of user mobility. From the perspective of cloud infrastructure providers, the proposed framework highlights the importance of efficient resource allocation as well as providing network and sever resource control as a service to mobile cloud service providers. Hence the proposed framework is built upon the composition based IaaS provisioning framework that leverages network programmability

provided by SDN to deliver higher degree of control over traffic, presented in Chapter 3. Also it is worth mentioning that the work presented in this chapter has been conducted in collaboration with Mr. Wijaya Ekanayake and presented at the 2017 IEEE 14th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, USA, [25].

The rest of this chapter is organized as follows. In section 4.2 design considerations and motivations of the proposed framework are presented. Then in section 4.3 similar approaches related to this this thesis are introduced. Section 4.4 elaborates on the system architecture and in section 4.5, improving resource reservation with user mobility across regions is analyzed. Section 4.6 evaluates the performance and analyzes the results and a summary and future research directions are discussed in section 4.7.

4.2 Design Considerations and Motivations of Proposed Mobile Cloud Framework

As mentioned previously in Chapter 2, there are several mobile cloud approaches based on the proximity and computing capabilities, namely: mobile ad-hoc clouds [78], cloudlets [80] , mobile edge computing [88] and traditional remote clouds. Although large remote clouds are cost effective, higher cloud access delay negatively affects the performance of mobile cloud service [118]. In contrast, ad-hoc clouds and cloudlets are placed in close proximity to user devices. Even though the access delay is lower, they are limited in computing capacity and also are highly sensitive to variations of relative distances between end points. Furthermore, a number of challenges in cloudlet deployment such as trust, security, placement [119] add limitations to using cloudlets for cost-efficient and QoS mobile cloud services. Another research work on near proximity clouds uses edge clouds. Instead of a larger centralized cloud, edge clouds are geo-distributed and are placed closer to user locations (cities, suburbs etc.) to provide a better QoS with minimal communication overhead [120]. Being placed closer to users as in a form of a small datacenter, distributed edge clouds are the most suitable alternative for remote

clouds to provide powerful cloud infrastructure at the network edge for mobile cloud computing.

Considering the benefits of edge cloud, a distributed mobile cloud architecture with regional edge datacenters (RDC) was introduced in this thesis. An RDC is able to host cloud infrastructure for mobile applications within a given geographical region. QoS requirements for IaaS-based mobile applications are considered at the time of an RDC placement. RDCs are interconnected over a WAN and each RDC serves its regional users. Discovery of the local regional RDC is carried out at the domain name resolution phase by the mobile device. As described in the previous work related to this research, [90], it is worth noting that dedicated resources are allocated for each device based on QoS and service provisioning cost. However, in distributed mobile clouds, service performance can be affected during a user mobility event. When a user moved to a remote region, user traffic had to be transferred over WAN to provide access to user dedicated cloud resources. Remotely accessing the cloud has several challenges such as (i) exceeding the maximum tolerable delay, (ii) risk of packet loss and (iii) imposing a higher network usage cost. Even if it may not be frequent, such violations need to be addressed in the IaaS mobile cloud in order to maintain QoS. Therefore, in regional IaaS clouds, challenges due to mobility can be alleviated by keeping the cloud as close as possible to the user during movement.

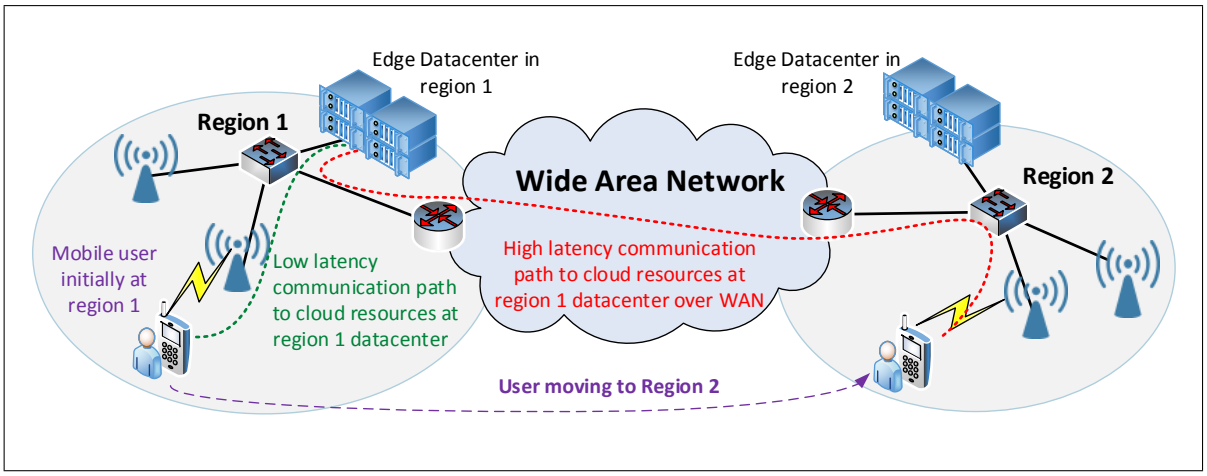


Figure 4.1: Application experienced delay variation in mobile clouds

Keeping the cloud services connected to client applications in user device during mobility is a complex task, which requires flexible and dynamic network resource management between the user and the cloud resource. Figure 4.1 illustrates a such challenging scenario where mobility of the user results in longer network distances and delay variations in mobile cloud services. However, it is observed that the current network architecture introduces a number of challenges and bottlenecks [121] in managing networks. To address these limitations, flexible and dynamic network management capabilities offered by SDN is employed to maintain necessary isolation and abstractions. Multi-tenancy within an RDC is achieved by virtualizing its compute and network resources. The proposed framework for mobile resource management creates infrastructure on demand and provides IaaS for each mobile device. Furthermore, it measures QoS requirements in an event of user mobility, and reactively relocates cloud resources to a selected RDC maintaining end-to-end transparency. The framework benefits from the network control plane programmability offered by SDN. In this research the OpenFlow [14] protocol, which is the most popular SDN implementation to date, was employed to manage OpenFlow switches at the edge of the wireless and RDC networks. OpenFlow enables cloud service traffic flow path creation, flow path update, layer 2 packet creation along with with IP mapping and rerouting for the seamless and transparent cloud access during inter-regional service mobility. In order to evaluate the performance of the framework with respect to existing scenarios, a prototype on Mininet [105] simulation platform and a physical test-bed were developed.

4.3 Related Work

In this section, a description is given of a few comparable approaches for the IaaS for mobile cloud services. The focus is mainly on the edge cloud approaches, resource allocation, mobility management with SDN and cross-subnet service mobility.

In order to make use of underutilized compute resources, authors of [78] propose an

ad-Hoc cloud architecture by forming application-dedicated cloudlets with a set of trustworthy pool of computing devices within an organization. This mobile cloud architecture reflects a datacenter for mobile users which can host different application services. However, this approach does not consider potential QoS violations that can be generated with end user's geographical movement. In [88], mobile edge computing (MEC) concept is introduced where compute servers are placed within the mobile network. Similarly, authors of [122] propose eNodeB-cloud architecture for mobile operator networks. However, due to the service provider-locked nature and difficulties in providing a common pool of resources for any public mobile cloud user, MEC or eNodeB cloud does not provide considerable openness and benefits over RDCs.

In [80] an intermediate entity named cloudlets is proposed to be placed between mobile devices and remote clouds closer to mobile users. The main benefit of cloudlets is that they provide consistent connectivity in the case of a remote cloud network failure. Similar to that, authors of [123] present hierarchical edge cloud architecture for mobile computing. However, considering the aforementioned limitations with remote clouds, and placing many cloudlets at lower tiers, cloudlet architecture does not provide significant cost benefits for IaaS clouds.

The previous work [120] proposed QoS-aware resource allocation mechanisms for networked edge datacenters. The ILP-based approaches were able to optimally locate cloud resources across edge datacenters. In this thesis, focus is on building a dynamic resource allocation and mobility-aware QoS management mechanism with SDN at RDCs.

In the literature, several approaches exist in order to achieve mobility across networks such as Mobile IP (MIP), MobileIPv6 and Hierarchical MIP. However, these solutions show a triangular flow of traffic adding extra latency for time critical applications [124]. Therefore, SDN-based approaches [125, 126] solve this issue by updating flow paths in OpenFlow switches. In [126], OpenFlow rules are updated at the wireless access edge switches to support mobility across networks. In addition to device mobility, in our approach, applying a similar concept with SDN, both mobile devices and cloud resources

are seamlessly and transparently moved across subnets.

4.4 Regional Infrastructure-as-a-Service Mobile Cloud with Software Defined Networking

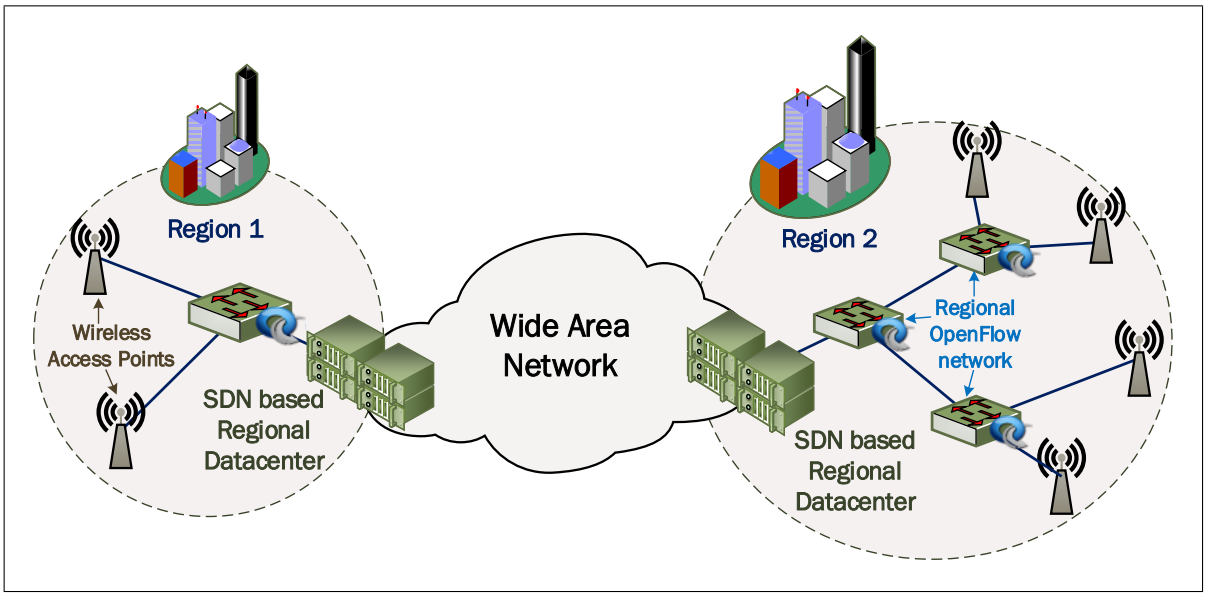


Figure 4.2: Software defined networking-enabled regional wireless networks

As shown in Figure 4.2, wireless networks in a region connect cloud users to the local RDC which are interconnected with a WAN. This inter-RDC WAN allows distribution of control information and user traffic among RDCs. RDCs are able to provide an abstraction of single centralized datacenter to mobile users, with the help of logically centralized resource management and seamless cloud service migrations. IaaS for a cloud application is acquired or released on-demand from the RDCs' virtualized resource pool. Such resource acquisition method minimizes the service provisioning cost for the mobile cloud provider.

In RDC formation, use of existing regional IaaS providers gives a number of benefits for mobile cloud providers. However, IaaS provider of a local region must be capable of providing a higher degree of control over compute and network resources for the mobile

cloud provider. Furthermore, user mobility across regions trigger rapid VM migrations between RDCs. Hence, IaaS providers must be capable of performing faster and low overhead resource allocation and release, rather than reaching optimum resource allocation levels in individual RDCs. Thus the composition based resource allocation framework presented in chapter 3 provides a comprehensive architecture for virtualized resource management in RDCs. With integration of this work, the system architecture of the mobile cloud framework can be illustrated in Figure 4.3.

4.4.1 System Architecture

The proposed mobile cloud service framework provides a logically centralized controller to manage the IaaS cloud. Figure 4.3 shows the RDC-based mobile cloud framework where local regional functions are carried out by regional wireless access and cloud service manager (RSM) modules. These modules are distributed in each region. Each RSM contains a wireless Network SDN controller and RDC IaaS controller (RIC).

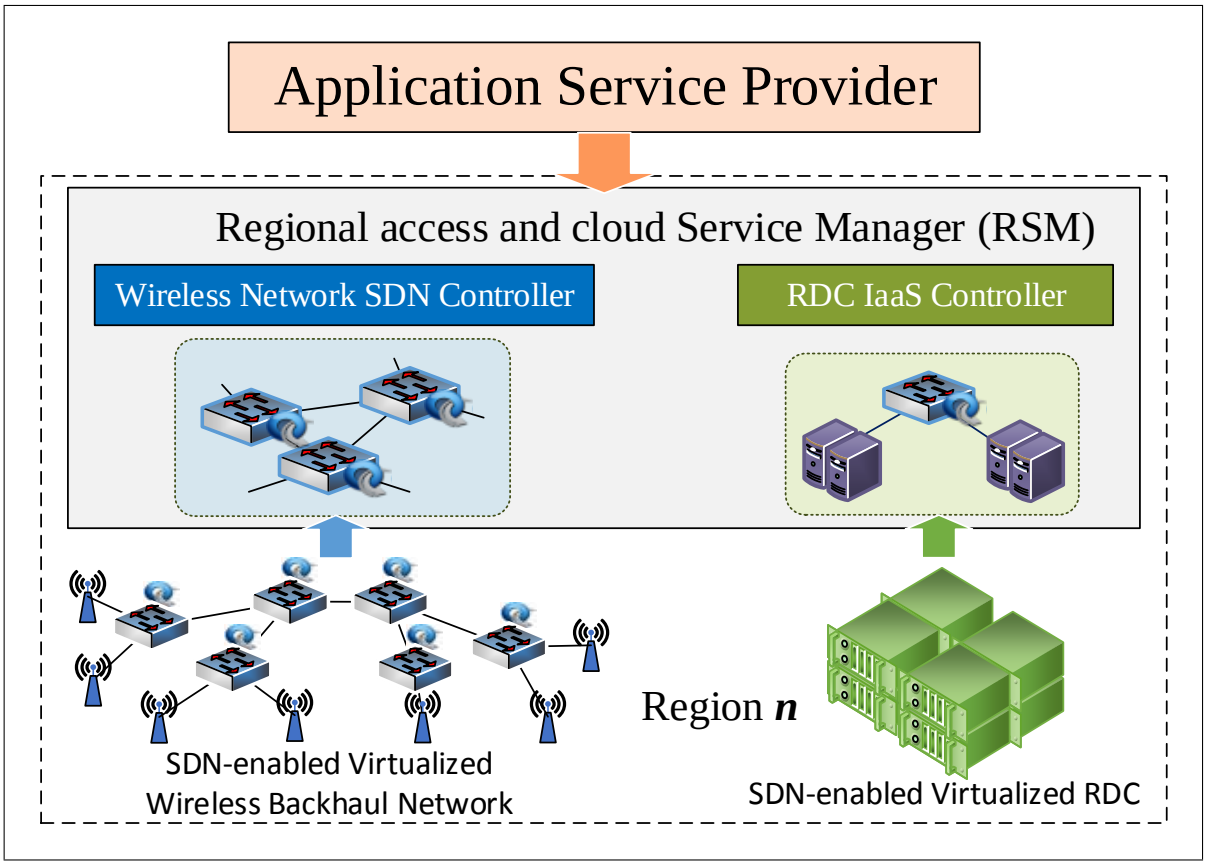


Figure 4.3: System architecture of the regional datacenter-based mobile cloud

RIC processes user application requirements, and generates an IaaS request to SDN-based virtualized RDC. Furthermore, it manages IaaS and communicates with the global control module mobile cloud service controller (MCSC) as shown in Figure 4.4. MCSC maintains an information database for each regional datacenter, acquired infrastructures and users. It also includes user identity, infrastructures identity, and network identities. RIC is composed of a local virtual resource controller that controls network and compute resources through an SDN based network resource controller and IT resource controller respectively. As mentioned before, this control module is obtained from the virtualized RDC designed according to architecture proposed in chapter 3.

4.4.2 Resource Allocation in Regional Clouds

During local RDC access, wireless network SDN controller resolves the domain name of the cloud and reply with the nearest RDC IP address. Then mobile device makes a cloud infrastructure request or accesses existing cloud services through cloud service access interface as shown in step (1) of Figure 4.4. In step (2), local RDC forwards such new resource requests to the MCSC. The cloud resource manager in turn calls the resource optimizer module to select a datacenter based on the user location. The selected datacenter guarantees user SLA, and also provides a minimal service provisioning cost for the cloud service provider.

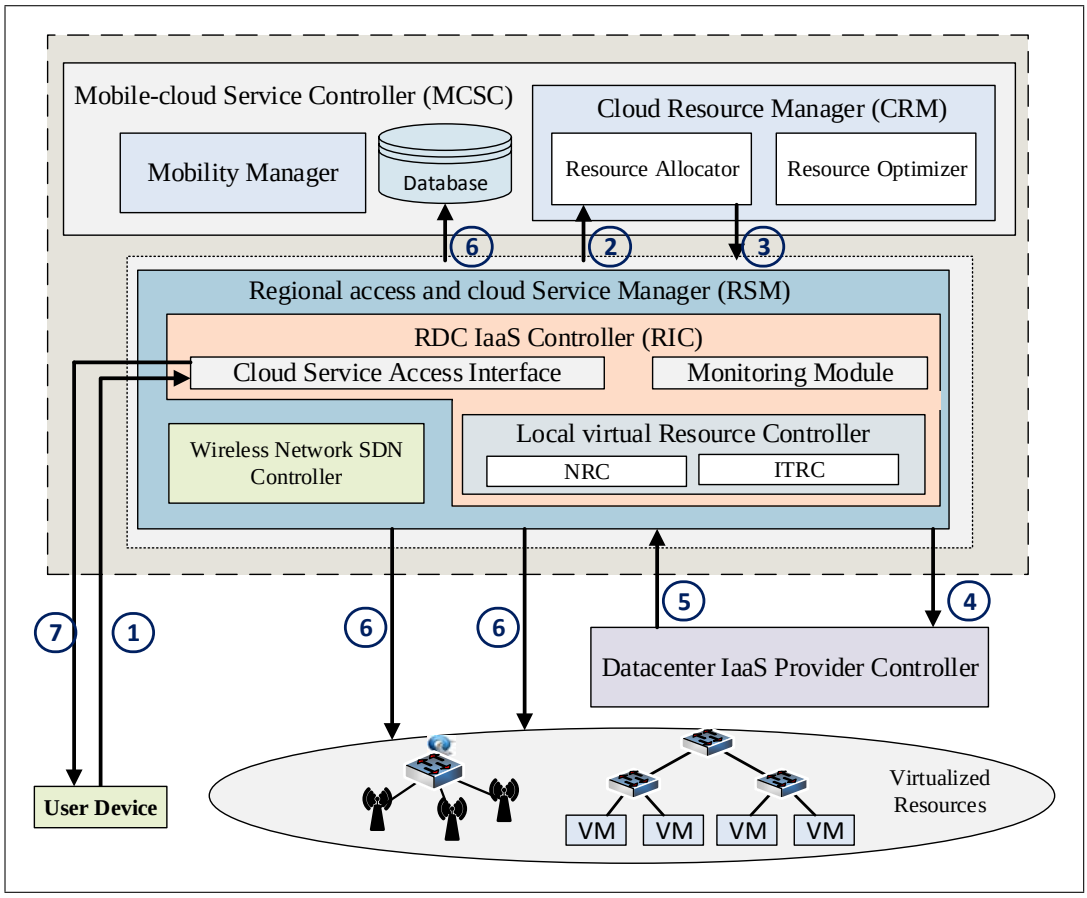


Figure 4.4: Resource allocation in regional infrastructure-as-a-service cloud

Once a datacenter is selected, cloud resource manager will notify the corresponding datacenter to create infrastructure in step (3). Subsequently, in step (4), the RIC will

make an IaaS request to the local Datacenter IaaS Provider Controller which replies back with dedicated IaaS as shown in step (5). For further details on resource allocation within a datacenter, reference is made to previous works in [90, 101]. When resources are created, in step (6), network flow paths are built on SDN-based networks. Subsequently, MCSC is updated to the IaaS acquisition, and then resource access information is passed on to the mobile user device.

4.4.3 User Mobility Management with Software Defined Networking

When a user moves to a new region, user is served by the new regional RDC. In the case of accessing an existing cloud resource at a remote datacenter, RSM requests mobile cloud service controller to locate user dedicated resource. The mobility manager of MCSC processes the request and replies the information available in database. It helps RSM to establish an end-to-end path to the remote RDC in SDN network. Once the traffic path in between RDCs is established, user can connect to the cloud resource remotely as shown in Figure 4.5(a).

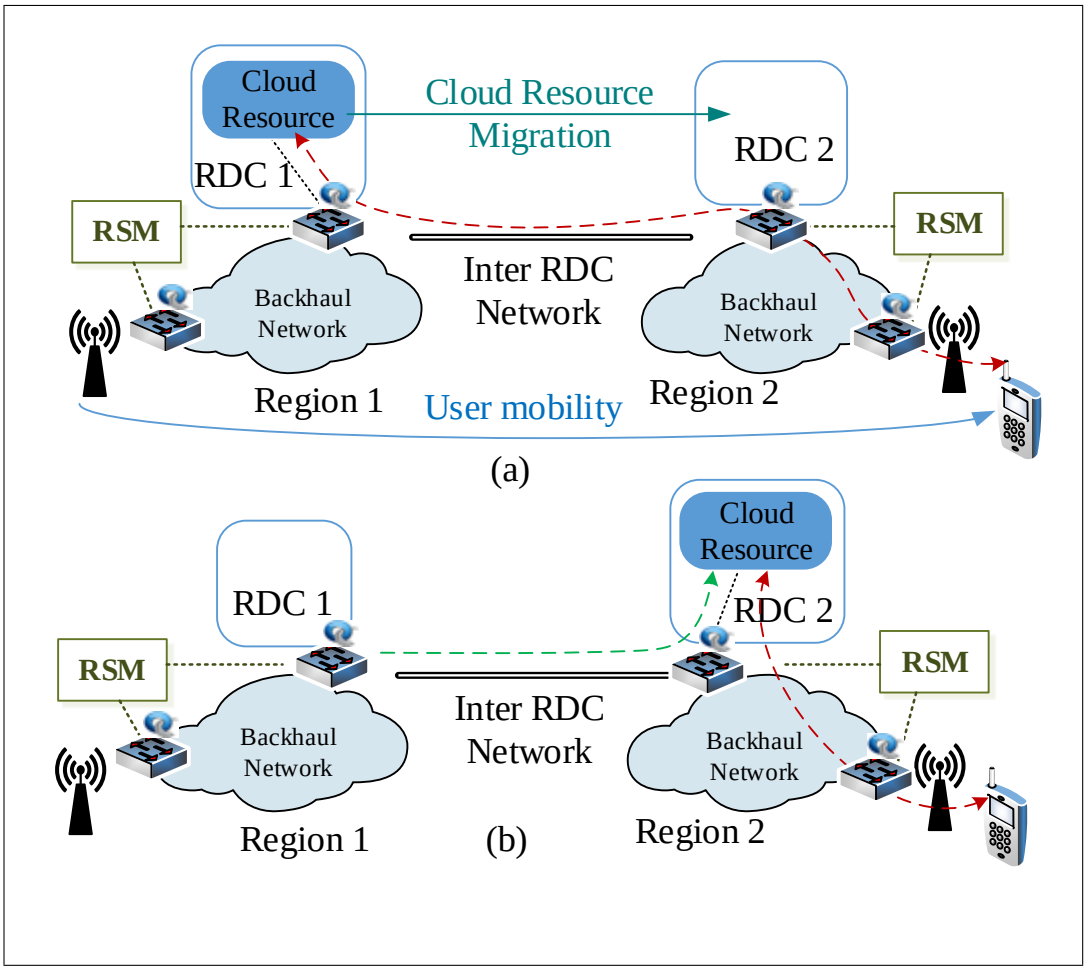


Figure 4.5: OpenFlow switches at network edges configure flow paths to/from cloud Resource and mobile device

However, during remote cloud access, the mobility manager of MCSC collects delay information from monitoring modules of RSMs for SLA monitoring. Then if necessary, it initiates a cloud service relocation process via CRM. When the cloud resource migration decision is taken, the resources are seamlessly migrated to the selected datacenter while user is accessing the cloud as shown in Figure 4.5(b).

In flow management, SDN plays a major role enabling seamless cloud service connectivity. When cloud resources are migrated across datacenters and their network boundaries, a number of additional techniques such as VPNs [128] are required to resolve ARP/RARP, and also to maintain ongoing communication. Otherwise, the cloud resource should renew its network configuration affecting TCP/IP connections. To avoid

such interruptions, SDN-based networks can be used to support cross subnet seamless cloud resource migration.

This is addressed by applying Identifier IP/Locator IP separation principle as mentioned in [124] and [125]. Generating a pseudo locator IP for the migrated cloud resource is done by the network controller. The pseudo IP is used in intermediate networks to route traffic to the mobile end nodes based on their location.

After selecting the suitable datacenter, cloud controller proceeds through following steps to perform seamless migration;

1. *Generating PsuedoIPs*: Generate pseudo IP considering parameters such as destination network, resource MAC

2. *VM Migration*: Live VM migration involves identifying memory threshold, Memory and operating state transfer. These steps are further elaborated in sub-section 4.4.4.

3. *Handling RARP*: Reverse Address Resolution Protocol (RARP) messages are intercepted by the controller and reply is generated using original IP and Destination network gateway.

4. *Handling ARP*: Controller need to intercept and reply for two types of Address Resolution Protocol (ARP) packets. First, ARP replies for the VM on behalf of original network gateway is generated using VM MAC, original IP and VM connected port. Second, ARP replies for destination RDC gateway from the device with pseudo IP are generated using VM MAC, pseudo IP and gateway connected port.

5. *Wireless Network Switch flow processing*:

   (a) Packets from mobile device to cloud $\rightarrow$ delete matching flows, update destination IP to pseudoIP

   (b) Packets from wireless gateway to the device with psuedoIP $\rightarrow$ Update source IP to original resource IP

6. *Source RDC Switch flow processing*:

   (a) Packets from mobile device to cloud → delete existing matching flows, forward with updating destination IP to pseudoIP

7. *Destination RDC Switch flow processing*:

   (a) Packets from cloud to mobile devices → Update Source IP to pseudoIP

   (b) Packets from RDC gateway to the device with psuedoIP → Update destination IP to original resource IP

After migration, mapping the pseudo IP-pseudo MAC to the original IP-original MAC and vice versa will maintain transparency between end points. In steps 8, 9 and 10, flow rules are generated such that OpenFlow switches located at network edges can forward traffic as shown in Figure 4.5(b). In step 5, SDN controllers are used to address RARP requests soon after live migration. Furthermore, an ARP handler was run in steps 6 and 7. ARP replies are created for the resource that is requesting the MAC addresses of the gateway. Similarly, replies are generated for the gateway requesting MAC of the pseudo IP bearing host.

4.4.4 Service Mobility through Live Virtual Machine Migration

There are four main factors that determine the success of the VM live migration process, namely VM memory size, memory page dirtying rate, network link bandwidth and latency. Page-dirtying rate is the rate at which the applications running in source VM updates memory pages (i.e. fixed-length contiguous segment of RAM). At the beginning of the live migration process, the source VM memory is copied to the destination VM iteratively. This is known as pre-copy stage and is illustrated in Figure 4.6 [136].

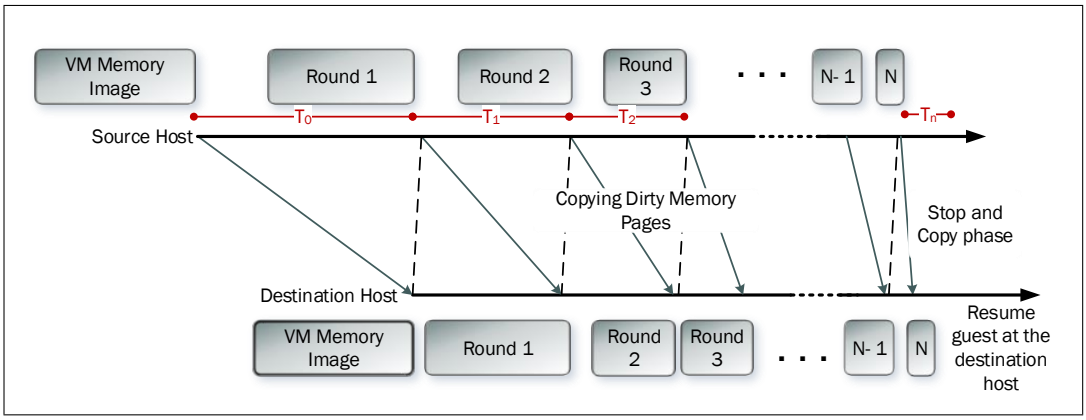


Figure 4.6: Pre-copy iterations in virtual machine live migration

At each pre-copy iteration, a set of memory pages from the source VM is copied over the network to the destination VM host, keeping track of pagers modified at the source VM. To maintain consistency, modified (dirtied) memory pages are copied again along with not-copied memory pages. With favorable bandwidth and link latency values, the number of memory pages need to be transferred is decreased with each iteration. However, after several iterations, page dirtying rate at the source VM applications becomes equivalent to the page transfer rate over the network. At this point, the pre-copy process is terminated, and stop-and-copy process is initiated. At stop-and-copy phase, the VM at the source is suspended and the final set of memory pages are transmitted instantly to the destination before resuming the destination VM back to running state. The following termination conditions determine when to end the pre-copy state and initiate stop-and-copy state:

- Source page dirtying rate exceeds page transmission rate
- Pre-defined remaining memory threshold has been reached
- Pre-defined number of iterations has been reached
- Total amount of data transmitted exceeds total VM memory (or reached pre-defined ratio)

Above exit conditions are defined in the pre-copy iterations as illustrated in pseudo code shown in algorithm 4.1.

Algorithm 4.1: Pre-copy termination conditions used in service migration

```
initiatePrecopy;
while MemoryThreshold > RemainingMemory do
    if Remaining Memory < MemoryThreashold then
        terminatePrecopy;
    else if NumIterations > IterationLimit then
        terminatePrecopy;
    else if TotalDataSent > VM_RAM then
        terminatePrecopy;
    else
        continuePrecopy;
    end
    RemainingMemory ← total_mem(res) - get_avail_mem(res);
end
initiateStopCopy;
```

Migrating VMs over LAN is a higly efficient and less complicated process. However live migration over WAN poses several challenges;

- Limited bandwidth and high latency from longer inter-DC links
- Lack of fast access to a shared storage from both source and destination hosts
- Need for L2/L3 address management to maintain consistency while satisfying traffic management policies in source and destination LANs

In this work, to further reduce VM offline time, the pre-copy process was constantly monitored and once the remaining memory at source reached pre-determined threshold, the cloud controller proactively starts flow rule initialization. This threshold was obtained based on the memory copying rate and remaining memory amount.

In the proposed framework, if a given user mobility scenario triggers service migration, it initiates live VM migration process that transfers VM to a data center in a different region over WAN. To minimize the application disruptions and maintain the seamless nature during the process, an SDN based approach inspired by CrossRoads [129] has

Table 4.1: Number of flow rules required to perform seamless service migration triggered by qualified user movement between regions

Main Task	Sub-tasks	Num rules	Installed in
Maintaining connectivity with the user (that moved to new region) while the cloud service is migrated to an RDC selected by the service migration algorithm.	Maps psudo-IPs of the packets destined to VM back to its original IP	1	User Destination region datacenter
	Maps original IPs of the packets destined to mobile device to its psudo-IP	1	User Destination region datacenter
	Maps psudo-IPs of the packets destined to mobile device back to its original IP	1	User Destination wireless network
	Maps original IP of the packets destined to VM device to its pseudo-IP	1	User Destination wireless network
	Reroute packets in transit (going to source datacenter) towards destination datacenter by adding VM pseudo IP (temporary triangulation)	1	Source datacenter
After transit packets are fully delivered, user device traffic is directly forwarded to the VM in destination RDC.	Pseudo IP and original IP mapping at ingress SDN switch adjacent to VM	2	VM Destination datacenter
	Pseudo IP and original IP mapping at ingress SDN switch adjacent to user device	2	User Destination wireless network
Handle ARP and RARP requests coming from gateways during the operation.	Maintain L2 transparency of the network by ingress SDN switch adjacent to VM	2	VM Destination datacenter
	Maintain L2 transparency of the network by ingress SDN switch adjacent to user device	2	User Destination wireless network

been employed in this work. CrossRoads presents an SDN based approach for seamless VM migration between sub-nets.

When performing service migration while maintaining seamless connectivity to a mobile user in an SDN platform, several flow rules are required to install for the purpose of performing different tasks. Table 4.1 summarizes the number of flow rules installed to achieve each task during service migration. It is worth noting that the user destination region RDC and VM destination RDC can be different since the VM placement algorithm may select a different RDC to migrate the VM, as described in section4.5.

4.4.5 Multiple Distributed Controllers for Cloud Service Migrations

In the proposed cloud service migration framework, the centralized SDN controller can easily become a performance bottleneck. Particularly in a scenario where multiple mobile user movements between regions trigger a large number of cloud service migrations, single centralized SDN controller can be overloaded. This issue can be addressed by adopting the distributed SDN control plane solution proposed in [130]. In [130], authors propose a role separation strategy based distributed controller approach where home, foreign and correspondent controllers employ locator and identifier IPs to perform seamless service migrations across regions.

Further analysis of distributed controller-based solutions revealed that maintaining consistency between distributed controllers during cloud service migration is a challenging task. Moreover, inter-controller communication delays can result in additional packet losses during cloud service migrations. Therefore, in this work, the performance of the proposed cloud service relocation algorithm was evaluated using a single centralized controller. However, to reduce the workload on the centralized SDN controller, two techniques were employed. First, most of the flow rules were installed proactively with large timeouts to avoid sending frequent requests to controller from forwarding switches. Second, in the proposed RDC selection algorithm, VM migration was considered only

when user movement violates application QoS.

4.5 User Mobility Induced Regional Datacenter Selection

In order to support QoS during user mobility, the monitoring module in each RSM keeps parameters for other datacenters including inter-RDC latency and bandwidth usage. Then the resource optimizer selects a set of RDCs based on latency requirement. Consideration is given to IaaS are being provided as dedicated VM, and hence movement of one VM does not affect the other.

The process of RDC selection after user mobility can be analyzed as follows. When a given mobile user u, move from a regional wireless network a_i to a destination wireless network a_j , define d_i as a datacenter in region i where the VM is originally located, and d_j as the local RDC to the user's current region j . Wireless network a_j will initially forward user request to d_j. Let $D^i_{j,u} = \{ d_1, d_2, d_3, \ldots, d_n \}$ $(n \in \mathbb{Z}^+)$ the set of RDCs that meets SLA requirements of the user application based on current location. $|D^i_{j,u}| \neq 0$, since we assume the RDC $d_j \in D^i_{j,u}$ with the minimum access delay. Also, $D^i_{j,u}$ may contain d_i if SLA is already met at the current VM location. Let d_k be a datacenter such that $d_k \in D^i_{j,u}$; $\delta^l_{u,j}$ and $\delta^l_{k,j}$ denotes individual link latencies from device u to d_j and datacenter d_k to d_j respectively.

If α is the maximum tolerable latency that guarantees SLA;

$$\delta^l_{k,j} \leq \alpha - \delta^l_{u,j} \tag{4.1}$$

$\delta^l_{k,j} = 0$ when $k = j$

The algorithm considers three resource placement options as shown in Figure 4.7. The objective is to find the most suitable RDC d_k. In [131], always moving the cloud resource to the nearest host d_j is preferred. However, in this work approach, moving

cloud resources to the nearest RDC may not always be cost-efficient due to capacity limits and resource and energy cost variations.

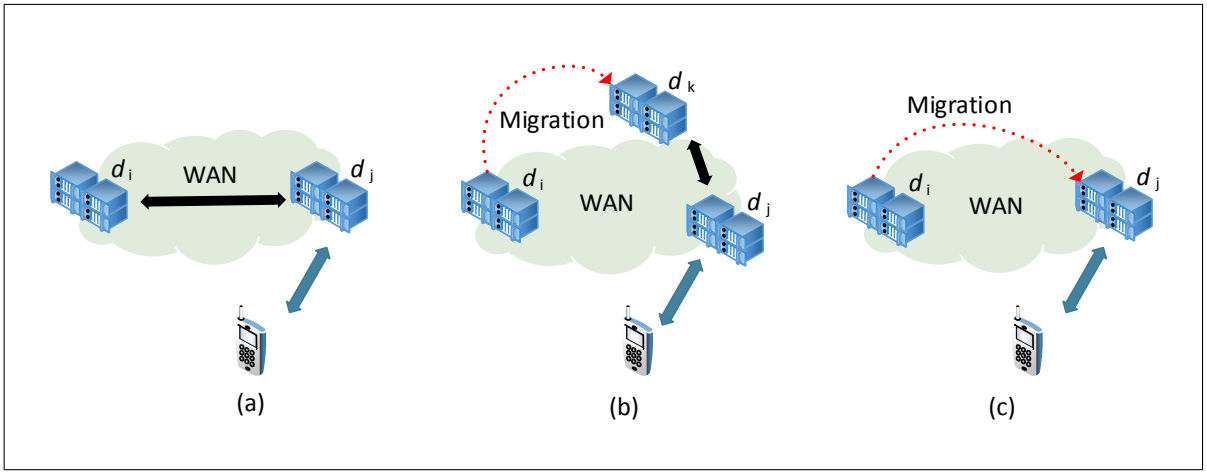


Figure 4.7: (a) Cloud resource is located at d_i forwards traffic through wide area network, (b) Cloud resource is migrated to d_k and connected to the user via d_j, (c) Cloud resource is migrated to dj which in turn locally serves the mobile user.

To find the suitable VM placement, consider the case of a single user accessing a VM located at d_i, and then moving to region a_j. Once the relocation procedure is triggered, it evaluates operational costs to decide whether to keep the VM in d_i or to move it to a different datacenter such as d_k. Before the migration process, the user accesses the VM located in d_i through WAN links $l_{i,j}$ and $l_{j,u}$ which are between d_i and u. Let $b^l_{i,u}$ be the sum of link bandwidth costs on links $l_{i,j}$ and $l_{j,u}$, when the user moved to a_j and received cloud services from d_i. If the computing cost per unit time for the normal operation of VM at d_i is represented by c^{d_i}, the cost $N_i^{u_j}$ can be calculated at time t when running user VM at d_i by;

$$N_i^{u_j}(t) = (b^l_{i,u} + c^{d_i})t \tag{4.2}$$

The parameter c^{d_i} contains the other costs of running the VM, including CPU and memory use. Bandwidth cost $b^l_{i,u}$ is the cost involved with WAN usage when the user is accessing the VM. Since the access network always forwards application traffic to the nearest datacenter d_j, the user also accesses the VM through d_j. Thus, link bandwidth cost $b_M^{i,k}$ is the sum of link costs per unit time between d_i and d_j during migration. Let

$c_M^{i,k}$ be the sum of additional computing costs imposed on both source and destination datacenters in a unit time. If the total VM migration time is $T_{i,k}^u$, the total cost at time instance t for both VM migration and operation at datacenter d_i becomes;

$$C_{i,k}^u(t) = \begin{cases} (b_M^{i,k} + c_M^{i,k})t + \int_0^t N_i^{u_j}(t)dt; & 0 < t < T_{i,k}^u \\ (b_M^{i,k} + c_M^{i,k})T_{i,k}^u + \int_0^{T_{i,k}^u} N_i^{u_j}(t)dt + \int_{T_{i,k}^u}^t N_i^{u_j}(t - T_{i,k}^u)dt; & t \geq T_{i,k}^u \end{cases} \tag{4.3}$$

The migration time $T_{i,k}^u$ depends on several factors including VM's total memory, migration throughput and, the memory dirtying rate [132]. The memory page dirtying rate is determined by the workload carried out during migration. For a given VM executing a specific application, a constant page dirtying rate during migration can be assumed. Then T_u can be predicted from historical user mobility patterns, through social networks or by direct application-level interaction [133].

4.6 OpenFlow-based Implementation of Regional Mobile Cloud Framework

The test environment for the RDC is composed of a physical and a Mininet-based test environment. In this work the distance was simulated using latency values. In Mininet, for the purpose of achieve isolation among virtual hosts, VMs were used on KVM hypervisor [65], and connected to Mininet network. These VMs that represent both RDCs and wireless networks, support nested virtualization providing the ability to create VMs within VMs. Therefore, a mobile device VM can migrate from one host VM to another VM which are acting as wireless networks emulating inter-region user movement and cloud access for evaluation.

In the physical testbed implementation, two datacenters were constructed, each containing compute nodes with Intel Xeon and i7 processors and a range of memory and

storage resources. Ubuntu server version 14.04 LTS was used in RDCs. In addition, Ubuntu 14.04 workstations were used to create Wi-Fi networks with physical adapters and Hostapd [134]. Android mobile devices were used as clients.

Using Open Virtual-Switches (OVS) [106], OpenFlow controllers were able to manage data flows of the VMs in datacenter networks. In both test-beds, a centralized cloud controller was created to evaluate the regional cloud framework. In Mininet implementation, four different regions were created. In physical implementation, we created two regions in order to evaluate resource creation and mobility performance. The MCSC was built as a Java application. It accepts IaaS requests from the RDC cloud controllers. Then local RDC controllers use its reply with to create VMs. Furthermore, Python based POX [48] controller was used to build SDN controller for the framework. Small-scale virtual machines with 128MB of RAM running Ubuntu server edition were created per application request.

Initially, a low latency network to distribute control and migration traffic. Secondly, the higher latency data network was employed in-band for the same purpose. In Mininet, ICMP ping were used to measure the connectivity during user mobility which reflects the seamlessness of the service. In the physical testbed, two user applications were considered in addition to the ICMP ping results, a video streaming service with MediaTomb [135] server and an FTP file transfer.

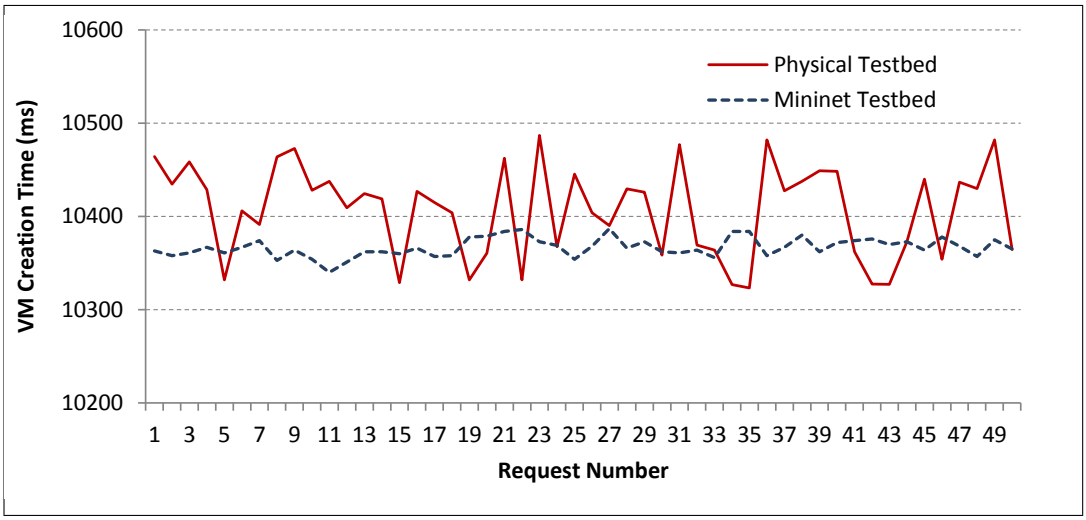


Figure 4.8: Time to allocate a virtual machine per application request

Since resource allocation time is a major factor determining the performance of the mobile cloud, initially, VM creation time with the centralized controller algorithm was measured. The VM creation time includes defining domain XML, reading and updating the database, disc creation and flow path update of the SDN network. Figure 4.8 shows the time of VM allocation per request. In the Mininet testbed, VM creation time appeared to be constant while the physical testbed shows a slight fluctuation. This fluctuation was mainly due to heterogeneity of the physical hosts used. Once resources were allocated, the system behavior was measured with user mobility.

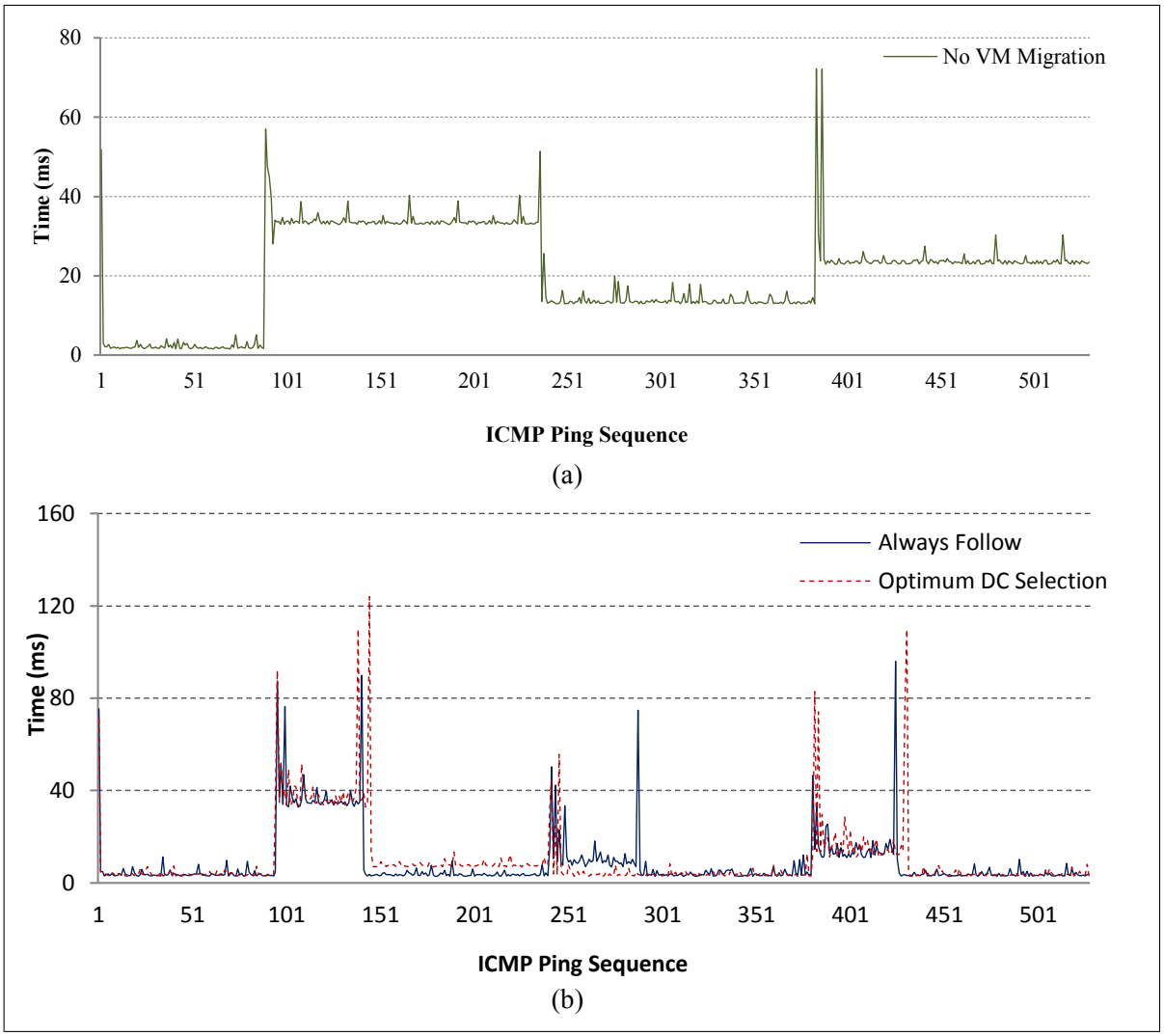


Figure 4.9: (a) Ping latency variation experienced by the mobile device virtual machine when not migrated, (b) When the virtual machine is always migrated to the nearest datacenter (blue) vs the virtual machine is migrated to the most suitable datacenter with cloud service relocation algorithm (red).

Figure 4.9 shows the network connectivity in the Mininet simulation when pre-defined cost values are used in decision making. The user is randomly moved among regions and observed the seamless end-to-end connectivity. Since VMs used as mobile devices were live migrated, at the IP networks, the SDN controller can consider it as a device handover. Transient peaks correspond to the time required for generating flow rules.

In Figure 4.9(a), the VM is kept at the source datacenter with no migration. It gives a higher variation of latency during mobility. In Figure 4.9(b), the relocation algorithm

places the cloud resources based on the QoS parameters. Here, the relocation algorithm minimizes unnecessary VM migration considering QoS and cost. This process shows the comparable benefit over the always-follow scenario for mobile clouds.

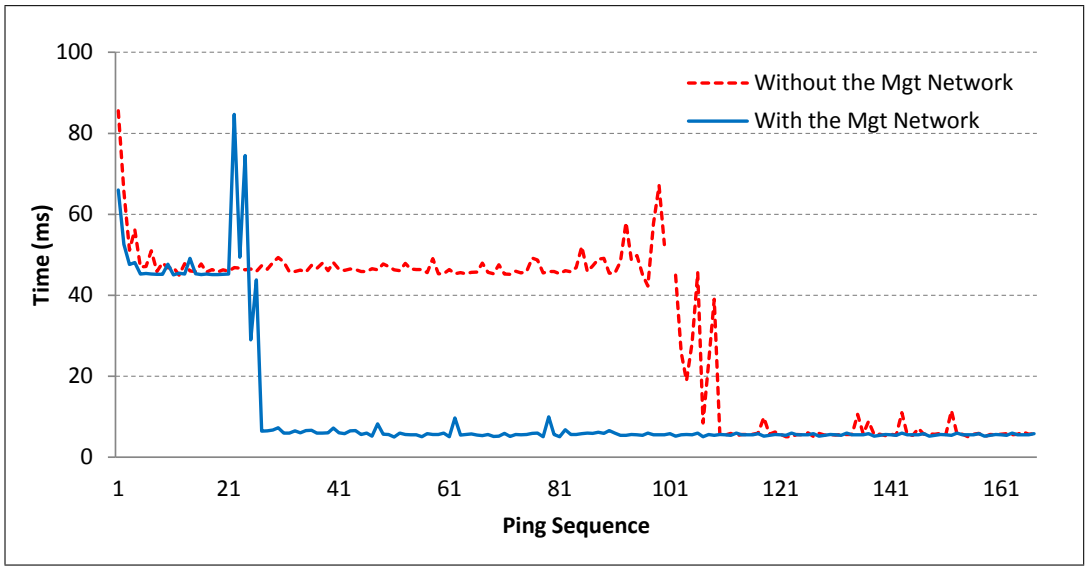


Figure 4.10: Ping latency variation in Mininet emulation environment

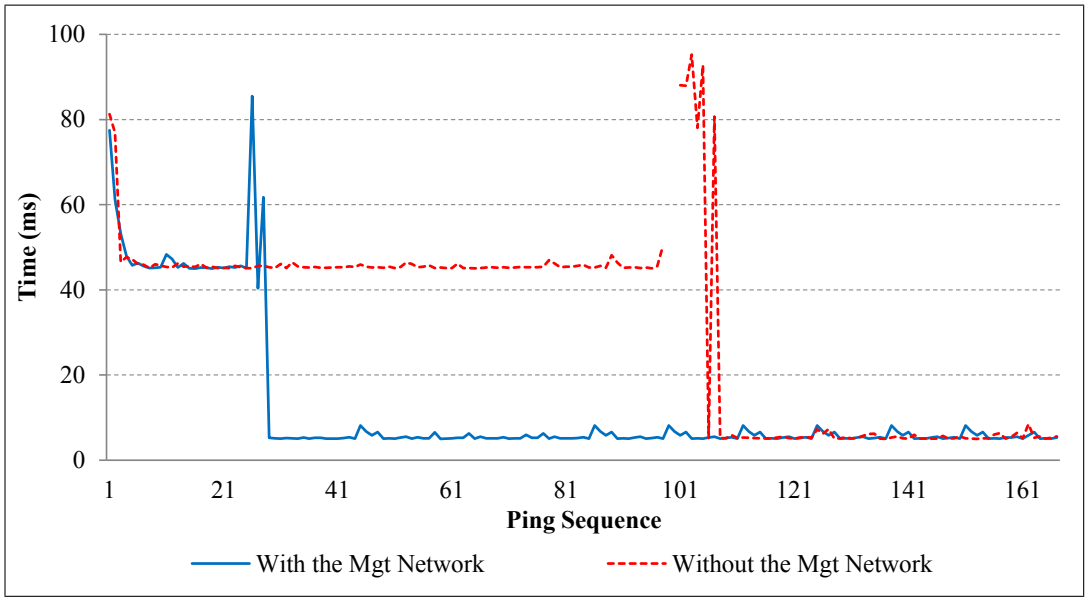


Figure 4.11: Ping latency variation in physical test environment

Figure 4.10 shows the access delay variation during migration in Mininet environment with and without an out-band management network. Out-band network is a completely

Table 4.2: Virtual machine migration completion time in physical test-bed

Test Case for VM live migration.	Average Migration Completion Time (seconds)	
	With management network	Without management network
ICMP Ping	10.7	36.7
Video Streaming	11.5	41.2
File Download	23.0	74.2

separate network with low latencies containing only control traffic. Figure 4.11 shows delay variation in the physical testbed. The observation results show a loss of ping packets following VM migration between ping sequences 99-101.

Regardless of the SDN-based techniques, packet-loss is noticeable when sending bursts of traffic. Due to the centralized nature of the controllers, VM migration monitoring and network state update is challenging when the high latency network is used to distribute the control traffic. Within this time interval, existing flow rules forward packets to the source RDC. To avoid packet loss, a precise time of migration completion is required. Hence, the migration monitoring threshold value becomes a variable.

The VM migration time mainly depends on its workload during migration. Hence, the VM migration completion times were analyzed as shown in Table 4.2 with an adaptive threshold. In this work, the WAN cost is identified as one of the compelling parameters in finding a regional datacenter that minimizes the service disruption and migration cost. Also, it can be concluded that the distribution of functions of the centralized controller further will minimize the bottlenecks in network state update.

4.7 Chapter Summary

In this work, a regional cloud framework with SDN was proposed, to provide QoS-aware IaaS for mobile cloud applications. With the proposed distributed RDC architecture, it is possible to allocate virtual computing resources for mobile applications. It was observed that during user mobility scenarios, the framework minimizes the service cost

and maintains QoS by seamlessly relocating cloud resources to a suitable RDC. By finding a set of RDCs based on user location and evaluating their eligibility, proposed algorithm avoids unnecessary migrations when QoS requirements are satisfied. Furthermore, the framework with SDN minimizes the network interruptions when end points move across subnets while maintaining end-to-end transparency. As future research related this thesis, its worth to investigate techniques to improve the decision-making algorithm and mobility predictions.

5 Cloud Network Resiliency With Software Defined Networking-based Dynamic Restoration

5.1 Introduction

Virtualization technologies play a key role in cloud resource management frameworks. They provide necessary abstractions and isolation required to create multiple logical instances on physical infrastructure, enabling dynamic resource management. Although virtualization improves resource utilization, it also increases the risk of interrupting operation of multiple cloud tenants with the failure of the hosted hardware. Compared to individual server failures, network failures affect more cloud users as they can cause service disruptions in multiple VNs as well as cutting access to one or more servers. Similarly, in comparison to network link failures within a datacenter, inter-datacenter link failures are highly disastrous to cloud operation due to the fact that such failure can completely disrupt a large number of client applications. Also, costs of inter-datacenter link failures can be significantly increased with high recovery times common in wide area networks.

One of the main challenges associated with network link failures is the lack of means to detect failures quickly for the purpose of applying fault tolerance measures. This is because failure of a network element usually breaks the communication path between the sender and and the receiver, reducing their ability to identify and localize failures. In

addition to lack of unified view over the network, traditional distributed routing networks can react to network events such as failures by converging into undesired states. These undesired states can adversely affect network service providers profits by causing poor resource utilization and QoS violations.

SDN and its leading deployment OpenFlow [14] have been widely regarded as a promising technology capable of addressing these challenges. Although, SDN effectively solved the majority of challenges faced by IaaS providers, fault tolerance in SDN-based network virtualization is one of the key areas that demands more investigation. Recent research in the domain of fault tolerant network virtualization can be classified into two groups: (a) path protection schemes [137–139] that proactively reserve resources for backup paths before detecting failures, and (b) path restoration schemes that reactively detect failures and install new paths during VN operation [140].

In this chapter, an IaaS management framework is introduced that: (i) allocates infrastructure resources satisfying application QoS requirements while increasing cloud service provider revenue through maximizing request acceptance and resource utilization, and (ii) assess the ability of SDN traffic engineering (TE) to handle unexpected multiple network link failures which minimises adverse effects on application QoS and substrate network utilization. The main contributions of this work include; (i) formulation of the VN resource allocation problem as a periodical integer linear programming (ILP) problem and optimization of the allocation of compute and network resources on geo-distributed cloud infrastructure, (ii) design and implementation of two OpenFlow-based QoS-aware reactive network failure restoration algorithms, which dynamically reroute traffic upon identifying link failures and, (iii) implementation of the framework and evaluation of the performance of proposed algorithms to identify their characteristics during multiple substrate link failures.

The rest of this chapter is organized as follows. First, existing IaaS management frameworks and their ability to protect and restore network failures has been examined. Second, the system architecture of the proposed framework and resource allocation prob-

lem formulation are described. Third, proposed SDN-TE path restoration algorithms are illustrated. Fourth, OpenFlow-based implementation of the proposed framework and its performance are evaluated in section V. Finally, the chapter summary and future research directions are presented.

5.2 Inter-Datacenter network failures

The proposed reliability approaches are designed for SDN based Networked Cloud Infrastructure (NCI) where a set of geographically distributed datacenters are interconnected by a wide area inter-DC network that is controlled by a logically centralized SDN control plane as illustrated in Fig. 5.1. In this work, it is assumed that the inter-DC network has a survivable topology where it remains as a connected graph after any single link failure. Even though distributed IaaS cloud environments employee network protection and restoration approaches to minimize service disruptions, their applications in the IaaS life-cycle differ due to limitations and benefits of each technique.

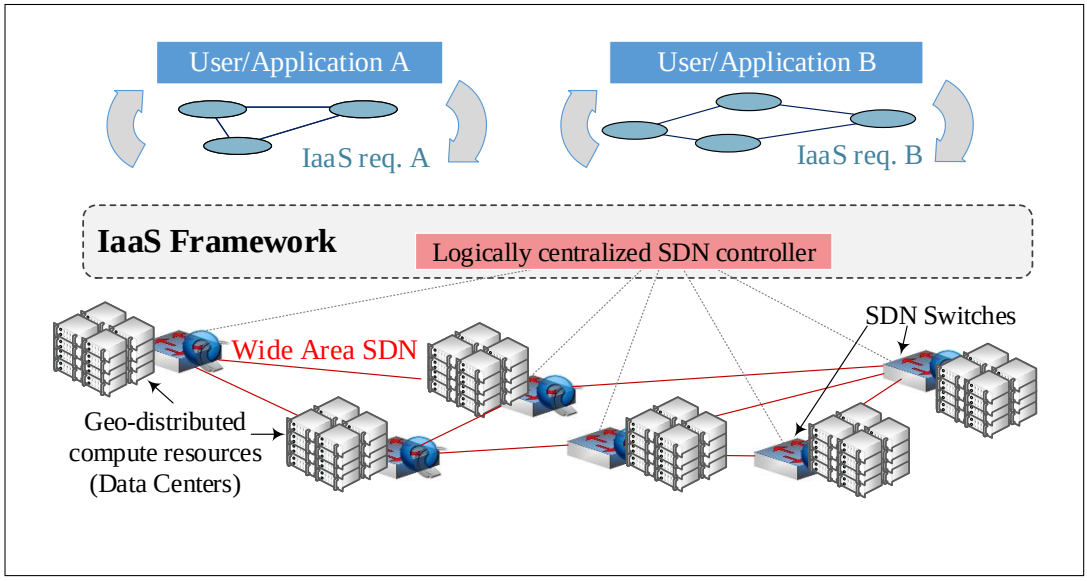


Figure 5.1: Geo-distributed software defined networking-based networked cloud infrastructure

Nearly one third of service disruptions in distributed DC cloud environments resulted

from connectivity losses in inter-DC network links [141]. Particularly in a scenario where cloud services are provisioned with VNs for each tenant, a failure in a substrate network link affects multiple tenant networks. Conventional local link backups are not very effective for such a scenario since each virtual link that uses the failed link may end up having inefficient routes causing additional delays. Fig. 5.2(a) shows an example scenario where two virtual links VL1 and VL2, allocated to supposedly two IaaS requests were affected from the failure of the substrate link l_1. In the topology shown in Fig. 5.2(a) and (b), failure in the physical link l_1 between datacenter A and B is recovered by reserving backup bandwidth in links l_2, l_3 and l_4. Since A is aware of the failure, it can directly select a backup path for VL1 through l_2 and l_4. However, if A attempts to forward VL2 to B using backup links l_2, l_3 and l_4, it creates a routing loop at link l_2 as shown in Fig. 5.2(b).

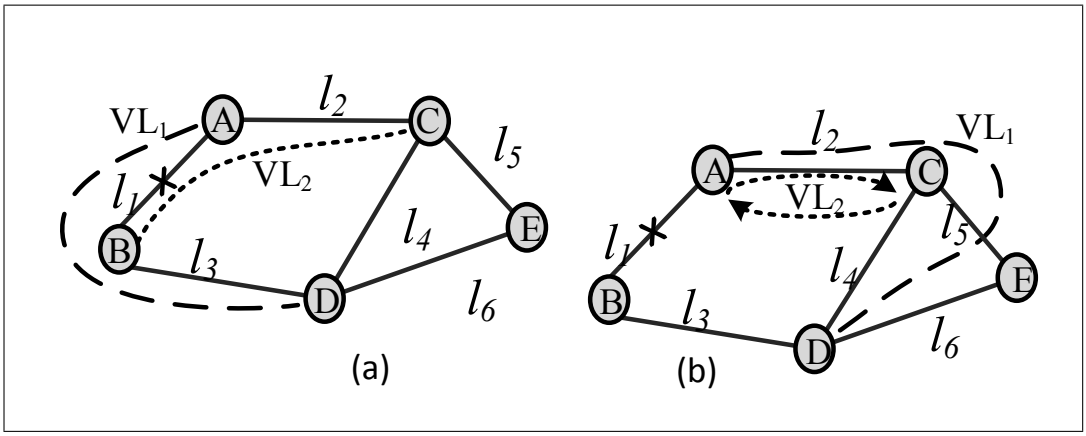


Figure 5.2: Link failures in inter-datacenter networks

In distributed routing protocols, this is effectively avoided by converging to a new network state. However, convergence times can extend up to tens of seconds resulting in prolonged service disruptions and often result in sub-optimal allocations. On the other hand, centralized network management and traffic engineering technologies such as MPLS, SDN are capable of recovering from link failures by leveraging an end-to-end view of the network to achieve faster recovery and better substrate network resource utilization.

5.2.1 Path Protection vs Restoration

Path protection approaches are designed to protect end-to-end network paths. They rely on fallback mechanisms that usually require advance reservation of additional substrate network resources (backup nodes, links or node/link capacities) along the alternative path. Path restoration implies taking necessary steps to recover from failures during run-time by failure detection, route calculation and path configuration.

Disjoint backup is one of the most common path protection approaches employed in WANs. In disjoint backup. path selection usually involves identifying the links selected for primary paths, either by increasing their cost or pruning (removing) them from the network topology graphs, used by path computation algorithms. Since the same amount of bandwidth in primary path must be reserved in the backup path, disjoint backup approaches result in significantly lower levels of average network resource utilization.

In contrast, restoration approaches rely on dynamic fault tolerance mechanisms. They do not require advance reservation of backup links or bandwidth. Instead, restoration techniques focus on detecting link failures during run-time and installing alternative paths to reroute traffic. Restoration can be performed by installing pre-calculated paths (proactive) as well as reactively calculating paths following the detection of failures. Although pre-calculation can effectively reduce restoration times, it lacks the ability to consider QoS requirements of the affected virtual links during restoration. Network control features of SDN such as centralized view, real-time network state detection and dynamic configuration have been employed to perform both proactive and reactive restoration.

5.3 Related Work

Failure restoration approaches are distinguished from protection approaches in that they do not need to reserve network resources in advance of failure. These approaches are also referred to as IP fast rerouting (IPFRR) [142, 143]. In [142] the authors point out that failure of advertising and routing table recalculation in commonly used protocols (OSPF,

IS-IS etc.) can result in long service disruptions, perhaps up to several seconds. These authors propose an ILP technique to calculate backup paths in advance. This approach can guarantee fast recovery from link failures with better cost efficiency than dedicated protection. A multi-link failure restoration IPFRR technique that pre-computes rooted, arc-disjoint, spanning trees is proposed in [143]. One of the main limitations of these IPFRR approaches is the inability to guarantee bandwidth and other QoS requirements during restoration. Also, high packet header overheads and increasing routing table entries can be expected, resulting in scalability issues and processing delays. In this work, consideration was given to the notion of IPFRR in an SDN environment and proactive path pre-computation as a benchmark was regarded as a means to evaluate the proposed SDN restoration technique.

In [144], the authors proposed a coordinated game theory optimization technique to re-embed virtual routers and VLs reactively after detecting failures. The authors argued that re-embedding is more cost effective and realistic compared to migrating virtual router images (and routing table states) to activate backup paths. However, this is not exactly true for an SDN environment since, in the event of failure, the centralized controller can both efficiently install new flows reactively and proactively configure forwarding elements to automatically switch to backup flows.

Comparatively few works have evaluated the potential of SDN technologies to restore link failures considering the QoS requirements of the affected IaaS requests and the service provider's objectives. In [145], the authors compare several path restoration options available in OpenFlow networks and propose a new, fast restoration algorithm to recover from link failures. This approach does not consider bandwidth, link latency or any other QoS parameters; therefore it cannot be directly adapted to the inter-DC network virtualization environment. Liu, et al. [146] designed a comprehensive VN management framework that introduces a hybrid scheme of protection and restoration, with SDN-based VN request embedding. A VN manager allocates VN requests with redundant nodes and applies a dynamic optimum restoration scheme to select suitable nodes after

failures are detected. Authors argue that even though node-related failures attribute to about 17% of unplanned network failures, failure of a node in a network virtualization environment has a bigger impact than in legacy network.

While this provides a valid motivation to proposed work, the reservation of standby routers for each VN can be highly expensive and can lead to lower levels of overall resource utilization. In addition, the proposed system will require selecting standby routers in the event of more frequent link failures, resulting in heavy restoration costs. On the contrary, by abstracting all routing and switching elements, as well as physical links in the core network as end-to-end paths that are allocated to virtual links, this work provides broader dynamic virtual link restoration solutions that can effectively select alternative routes to avoid both node and link failures. Also in this work, an SDN-based comprehensive VN management framework was introduced where a centralized resource manager performs VN embedding through optimum resource allocation and guarantees fault-tolerant operation. In addition an ILP optimization problem was formulated to provide dynamic restoration after network failures were detected. However, in this chapter, more attention was given to VL restoration since a node failure can also be regarded as multiple link failures.

5.4 Framework Design and Virtual Network Embedding

In this Section, the architecture of the framework is illustrated and the ILP problem used for VN embedding is formulated.

5.4.1 Periodical Infrastructure-as-a-Service Requests

Application service providers serve end-users connected from different geographical locations by deploying their applications on IaaS obtained from the framework. Each IaaS request specifies the estimated compute resources requirements in terms of virtual

machine specifications (e.g. lower bounds of the number of CPU cores RAM and storage) and connectivity QoS specifications (e.g. minimum bandwidth, maximum delay, and maximum hop-count) for the required links. Each compute node may also have a preferred set of DC sites characterized as embedding location requirements.

In a typical cloud environment, the IaaS provider cannot assume that the requests from application service providers arrive sequentially and at regular intervals [147]. Realistic IaaS provisioning therefore involves a periodic approach, where requests are queued and processed in small batches in order to optimize the service providers' profits over time. Accordingly, the embedding time was divided into a sequence of consecutive periods (windows). Between two consecutive periods, it is assumed that a significant proportion of IaaS requests remain the same as explained in [148].

The service demand change is measured with the turnover rate where 40% of incoming (new or add) and 30% of leaving (ending or drop) requests. More precisely, if the set of embedding planning periods is denoted by P and the initial set of IaaS requests denoted by $R(0)$, the set of IaaS requests $R(p)$ indexed by $p \geq 1$ is defined as:

$R(p) = R(p-1) + R_{NEW}(p) - R_{DROP}(p)$

where $R(p-1)$ is the set of accepted IaaS requests at the end of period $p-1$. $R_{NEW}(p)$ is the set of new incoming and $R_{DROP}(p)$ is the set of ending IaaS requests at the start of period p, where new and drop are randomly selected between 10% and 40%. This gives us a range of cases of IaaS demand, from slow (10%) to quick (40%).

IaaS requests arriving in each period are embedded with coordinated node and link embedding [147]. This approach was followed to highlight and evaluate the performance of proposed shared backup path allocation. Thus in the node embedding phase, CPU, memory and storage resources are allocated to satisfy compute requirements of IaaS requests.

5.4.2 Framework Architecture

The architecture of the proposed IaaS management framework is illustrated in 5.3. It has three main components:

i IaaS Resource Allocator: runs virtual node and link embedding algorithms;

ii Compute Resource Manager (CRM): allocates compute resources to IaaS requests by creating VMs in DCs;

iii Network Resource Manager (NRM): allocates virtual links and provide failure restoration.

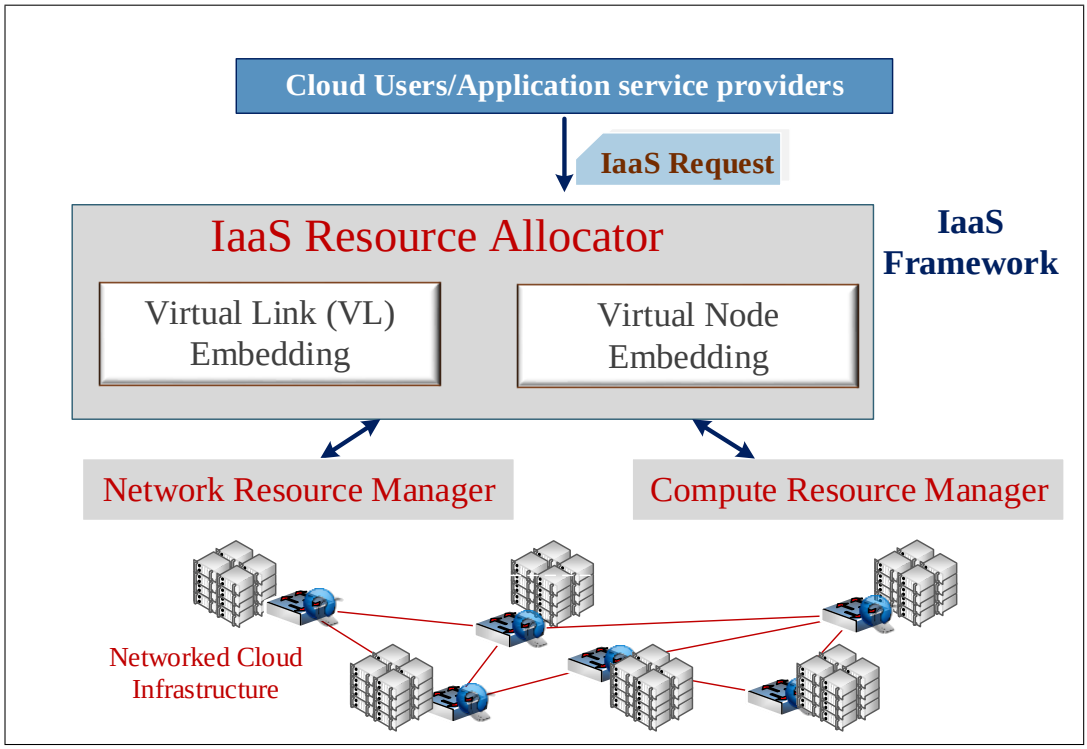


Figure 5.3: System architecture of the infrastructure-as-a-service management framework

The IaaS resource allocator collects all requests received during each period and forwards compute and connectivity requirements to virtual node and virtual link embedders. This is done in two phases. First, from a preferred set of DCs, the Virtual node embedder selects the optimum DCs that satisfy minimum resource requirements with the objective

of maximizing IaaS provider profits. After virtual nodes are embedded, selected results are taken as inputs to the VL embedder along with the VN link requirements and the substrate topology state. The Virtual link (VL) embedder hosts shared backup based fault tolerant link embedding algorithm. It (further illustrated later in this section) solves an ILP model to find optimum paths to interconnect virtual nodes, with the intention of minimizing the cost of the allocated resources and maximizing resource utilization. A periodical VN embedding based on a pricing/profit scheme is introduced to allow these two intentions to be merged into a single objective function.

The VN embedding results are used as input for the next embedding iteration, along with substrate resource details. These results are also forwarded to the network resource manager (NRM) and the compute resource manager (CRM) to create VMs and flows respectively. The CRM directly interacts with resource allocation platforms [90], in each DC, through application program interfaces (APIs) and produces virtual machines with the requested compute specifications. Substrate network paths allocated for VLs are reserved through the NRM, using SDN flow configuration. In addition to installing VLs during embedding, the NRM hosts an SDN-TE module, which provides dynamic path restoration. Section 5 contains a detailed discussion of the SDN-TE dynamic path restoration.

5.4.3 Periodical Virtual Network Embedding

IaaS requests which arrive in each period will be embedded in two phases: (i) node embedding and, (ii) link embedding, illustrated in this section.

Phase I: Node Embedding

The inter-DC network is represented by an undirected graph $G^s = (W^s, L^s)$, where W^s denotes the set of substrate nodes (DCs) and L^s the set of bidirectional inter-DC links. In a given time period t, each substrate link $l \in L^s$ has a residual bandwidth capacity $B^l(t)$ and each substrate node $w \in W^s$ has residual CPU capacity $P^w(t)$, memory capacity

$M^w(t)$, and storage capacity $S^w(t)$. Bandwidth unit cost of l is denoted by c_b^l and CPU, memory and storage unit costs of w are represented by c_p^w, c_m^w and c_s^w respectively. The set of VN requests arrived at t is given by $V^n(t)$,$n \in N = 1, 2, .., |N|$ such that each request $v^n(t) \in V^n$ is represented by a directed graph $G^n(A^n, E^n)$, where A^n denotes the set of virtual nodes and E^n the set of directed links. The QoS requirements of each VL $e \in E^n$ belonging to class $i \in Q^E = 1, 2, .., |Q^E|$ are defined by couple (b^i, h^i), where b^i is the required bandwidth and h^i is the maximum number of hops. Likewise, QoS requirements of each virtual node $a \in A^n$ belonging to class $j \in Q^A = 1, 2, .., |Q^A|$ are defined by the vector (p_j, m_j, s_j, r_j), where p_j is the required CPU, m_j is the required memory, s_j is the required storage and r_j is the set of substrate embedding locations. The QoS class of VL $e \in E^n$ (resp. virtual node $a \in A^n$) is denoted by $c(e)$ (resp. $c(a)$). Let $\pi_(u, v)^e$ denote a network path between DC nodes (u, v) assigned to e and $L(\pi_(u, v)^e)$ gives the number of substrate links in $\pi_(u, v)^e$. s and d are the source and destination nodes of e.

In the work of this thesis, the binary variable z^n is defined such that $z^n = 1$ if v^n is accepted to use node resources and $z^n = 0$ otherwise. Similarly, x_a^u is defined such that, $x_a^u = 1$ if virtual node a is assigned to substrate node $u \in W^s$ and $x_a^u = 0$ otherwise. For each time interval t , our objective is to maximize the profit by allocating DC node resources to requests. If P^e is the price offered by v^n for $e = (s, d)$, the objective function can be written as:

$$max \sum_{n \in N} \left[\sum_{e=(s,d) \in E^n} P^e z^n - \sum_{(u,v) \in W^s \times W^s} \left\{ \begin{array}{c} (c_p^u p_s + c_m^u m_s + c_s^u s_s) + \\ (c_p^v p_d + c_m^v m_d + c_s^v s_d) \\ + L(\pi_{u,v}^e) \times c_b^l \times b^{c(e)} \end{array} \right\} \times x_s^u x_d^v \right] \quad (5.1)$$

Subjecting to the following constraints;

i Embedding is done for all nodes of an accepted V_n:

$$z^n \leq \sum_{(u,v)\in W_s \times W_s} x_s^u x_d^v; \qquad (s,d) = e \in E^n, n \in N \tag{5.2}$$

ii Node a can be assigned to only one substrate node u:

$$\sum_{u \in r_{c(a)}} x_a^u \leq z^n; \qquad a \in A^n \tag{5.3}$$

iii Node a can be assigned only to a subset of substrate node regions $r_{c(a)}$:

$$x_a^u = 0; \qquad \forall u \in W^s, u \notin r_{c(a)}, \forall a \in A^n, n \in N \tag{5.4}$$

iv Residual node CPU capacity:

$$\sum_{n\in N} \sum_{a \in A^n} x_a^u p_{c(a)} \leq P^u(t) - P^u(t-1); \qquad c(a) \in Q^A, u \in W^s \tag{5.5}$$

v Residual node memory capacity:

$$\sum_{n\in N} \sum_{a \in A^n} x_a^u m_{c(a)} \leq M^u(t) - M^u(t-1); \qquad c(a) \in Q^A, u \in W^s \tag{5.6}$$

vi Residual node storage capacity:

$$\sum_{n\in N} \sum_{a \in A^n} x_a^u s_{c(a)} \leq S^u(t) - S^u(t-1); \qquad c(a) \in Q^A, u \in W^s \tag{5.7}$$

Phase II: Link Embedding

Let (u^*, v^*) be the embedding nodes assigned to source s and destination d of $e \in E^n$ in node embedding phase. To determine the embedding of e, the binary variable $y^{\pi}_{u^*v^*}$ is defined such that $y^{\pi}_{u^*v^*} = 1$ if the path $\pi^e_{u^*v^*}$ between substrate nodes u^* and v^*,

is assigned to VL $e = (s, d)$, and $y^{\pi}_{u^*v^*} = 0$ otherwise. $\alpha_l(\pi^e_{u^*v^*})$ is used as the link embedding parameter to inform if substrate link $l \in L^s$ is used by the shortest path $\pi^e_{u^*v^*}$. Hence, $\alpha_l(\pi^e_{u^*v^*}) = 1$ if l is used in $\pi^e_{u^*v^*}$, and $\alpha_l(\pi^e_{u^*v^*}) = 0$ otherwise. For simplicity of representation C^u and C^v are defined such that $C^u = (c^u_p + c^u_m + c^u_s)$ and $C^v = (c^v_p + c^v_m + c^v_s)$. The objective of the link embedding phase is to select substrate paths that maximize the profits by selecting least cost paths that match QoS requirements. Thus the objective funciton can be written as:

$$\sum_{n \in N} \sum_{e \in E^n} \left[P^e z^n - y^{\pi}_{u^*v^*} \left(C^u + C^v + \sum_{l \in \pi} c^l_b b^{c(e)} \right) \right] \tag{5.8}$$

Subjecting to the following constraints;

i For each link e in accepted v^n, an embedding path $\pi^e_{u^*v^*}$ must be calculated:

$$z^n \leq y^{\pi}_{u^*v^*}; \qquad e \in E^n, n \in N \tag{5.9}$$

ii For each link e, embedding path $\pi^e_{u^*v^*}$ is selected based on maximum number of hops defined in its QoS class:

$$L(\pi^e_{u^*v^*}) \leq h^{c(e)}; \qquad e \in E^n, n \in N \tag{5.10}$$

iii For each link e, embedding path $\pi^e_{u^*v^*}$ is selected based on available residual bandwidth defined in its QoS class:

$$\sum_{n \in N} \sum_{e \in E^n} b^{c(e)} \alpha_l(\pi^e_{u^*v^*}) y^{\pi}_{u^*v^*} \leq B^l(t) - B^l(t-1); \quad l \in L^s \tag{5.11}$$

As mentioned before, after solving node and link embedding problems for all IaaS requests arrived during the time interval t , VMs for accepted IaaS requests will be created by CRM and substrate network paths will be created by NRM.

5.5 Link Failure Restoration with Software Defined Networking

In this section, the operation of NRM is elaborated and description of proposed restoration algorithms which improve fault tolerance in VN management framework is presented.

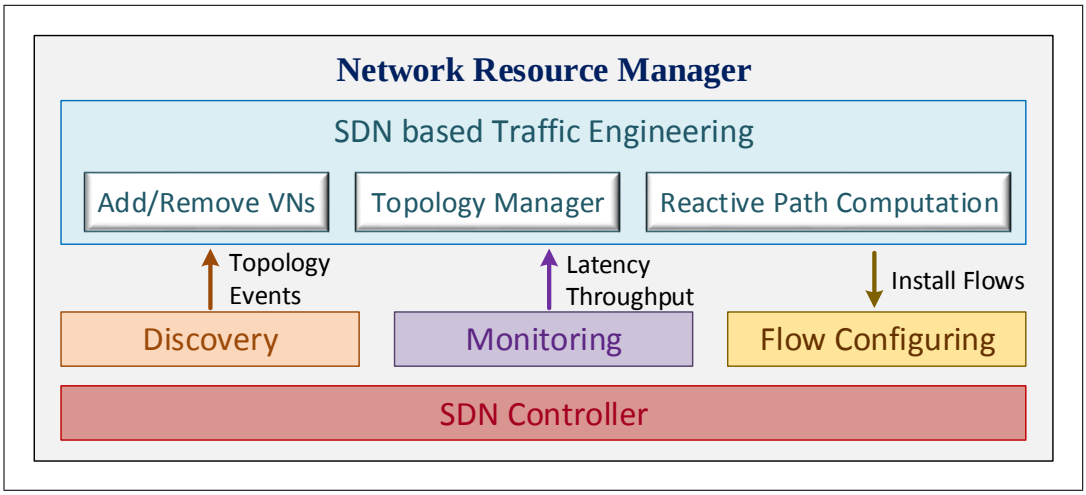


Figure 5.4: Software defined networking-based network resource manager

Main components of the NRM are shown in Fig. 5.4. The NRM contains an SDN-based TE module, SDN controller, and three controller application modules: discovery, monitoring and flow configuration. SDN-based TE module is composed of: add/remove VNs, topology manager and reactive path restoration functions. Discovery is responsible for identifying network nodes/links and provides their connectivity details to topology manager. Topology manager also uses throughput and latency measurements collected from the monitoring module to build and manage a detailed logical topology graph with individual link delays and bandwidth availability. This topology graph is maintained in the controller, and the average values are used periodically by IaaS allocator for VN embedding. The monitoring module was designed to measure per-flow throughput and individual link latency values, based on techniques proposed in [149]. The monitoring technique adopted to measure the per-flow throughput and the latency works as follows:

Monitoring Throughput

Obtaining per-flow throughput values is relatively straightforward in an SDN environment. For this, egress switch of each flow path is polled to retrieve per-flow counters adaptively in varying intervals. When new flows are frequently installed or when flow statistics are frequently changing, sampling rates are dynamically increased for higher resolution. Since these polling messages are short packets that read control-plane per-flow statistics, they do not add significant data-plane overhead.

Monitoring Latency

In IP networks, latency values are commonly measured by employing active and passive measuring techniques. Active latency measurements rely on injecting specialized monitoring while passive monitoring relies on none-intrusive traffic statistic measurements collected from network nodes. Although passive monitoring adds less monitoring overhead, real-time latency measurements obtained through passive monitoring can be relatively less accurate. Specifically, in SDN environments, passive latency measurements can be affected by lack of time synchronization between switches. Hence, in this thesis, controller assisted active monitoring technique proposed by [149] is employed. The adopted technique injects monitoring packets periodically from ingress switch of each flow path (with controller time as the payload) and retrieve from the egress node. From this, controller to ingress and egress node delays are deducted to calculate flow-path latency.

After embedding each set of VNs received from the IaaS resource allocator, the monitoring module is triggered to update the topology with link latency values and bandwidth utilization. Once NRM received a set of VLs from IaaS allocator with embedded substrate paths, add/remove VNs function will generate flow rules for all switches (in each embedded path) and will forward to flow configuring module of the SDN controller for installation. Following installation, discovery module will continuously verify node and link availability while monitoring module measure end to end latency and bandwidth

usage of installed VLs. Although the proposed framework is capable of responding to bandwidth violations, link congestions, and latency variations, this thesis considers them as subjects for future work. During a link failure, discovery module will fire a link failed event, which will be caught by the reactive path computation module. It will instruct topology manager to update the topology and at the same time, execute path restoration mechanisms. In this work, two path restoration algorithms have been introduced, namely: (a) QoS-aware reactive path computation and, (b) shared risk link group (SRLG) aware proactive path computation, designed to be individually executed in reactive path computation module.

5.5.1 Quality of Service-aware Reactive Path Computation

The proposed QoS-aware reactive path computation algorithm is shown in Algorithm 1.

Algorithm 1 Reactive Path Computation and Installation

Input: disconnected link l^d ; max compute paths c_π^{max} ; minimum under-allocation bandwidth b^{min} ; topology graph $G(W^s, L^s)$ where $l^d \notin L^s$;

Output: allocated flow set F^{aloc}; unallocated flow set F^{naloc}; opposite paths dictionary $P_{f,\pi}^{opo} : F \rightarrow \Pi$ where flows $f \in F$ as keys and corresponding unidirectional paths $\pi \in \Pi$ as values (calculated in previous iterations of Algorithm 1);

1. F^R = ***delBrknFlows(*** l^d ***)***;
2. **for** each flow f in F^R
3. **if** f *not in* $P_{f,\pi}^{opo}$ **then**
4. **while** calculated paths $< c_\pi^{max}$ **do**
5. get next *Eppstein* path π_{ep} from source to destination of f on $G(W^s, L^s)$;
6. **if** path bandwidth $b(\pi_{ep}) < b^{min}$ **then**
7. **continue;**
8. **else if** requested bandwidth by flow $b(f) \leq b(\pi_{ep})$
9. selected path $\pi_s \leftarrow \pi_{ep}$;
10. **break**;
11. **else**
12. low bandwidth path set $Q^\pi \leftarrow Q^\pi \cup \{\pi_{ep}\}$
13. **end if**;
14. **end while**;
15. **if** $|Q^\pi| \neq 0$ *AND* requested bandwidth not available **then**
16. $\pi_s \leftarrow$ *Highest available path bandwidth*(Q^π);
17. **end if**;
18. **else**
19. $\pi_s \leftarrow P_{f,\pi}^{opo}.get(f)$;
20. $P_{f,\pi}^{opo}.delete(f)$;
21. **end if**;
22. **if** $\pi_s \neq \emptyset$ **then**
23. $P_{f,\pi}^{opo} \leftarrow P_{f,\pi}^{opo} \cup \{(f, \pi_s)\}$;
24. $F^{aloc} \leftarrow F^{aloc} \cup \{f\}$;
25. **for** each node in π_s
26. allocate flow f on node;
27. **end for**;
28. **else**
29. $F^{naloc} \leftarrow F^{naloc} \cup \{f\}$;
30. **end if**;
31. **end for**;
32. **return** F^{aloc} and F^{naloc};

The algorithm takes the disconnected link l^d, upper bound of path computations c_π^{max}, lower bound of bandwidth under allocation b^{min} and pruned topology graph $G(W^s, L^s)$ as input and return allocated flow-set F^{aloc}, un-allocated flow-set F^{naloc} as output. It initially executes delBrknFlows sub-routine, to release resources allocated to flows affected by link failure. This was achieved by iterating over flows that used failed link and removing corresponding flow table entries from switches in their flow-paths. Affected

flows are restored immediately by finding multiple shortest paths between end-points, and evaluating their QoS parameters (bandwidth and latency). To find multiple paths between source and destination, we have implemented k-shortest path algorithm proposed by Eppstein D. [150], considering its improved complexity $O(m + nlogn + K)$. Thus the proposed algorithm iterates over affected flows (line 2) and execute k-shortest path algorithm in line 5.

The discovery module fires two unidirectional link events during each physical link failure for forward and backward paths. To maintain the symmetry (i.e. bidirectional co-routed property) in restored links and to minimize the number of shortest path calculations, an opposite path dictionary $P_(f, \pi)^{opo}$ is maintained. $P_(f, \pi)^{opo}$ stores calculated unidirectional links during each link failure. Thus line 3 guarantees that shortest paths are calculated only if pre-calculated path is not available in $P_(f, \pi)^{opo}$.

5.5.2 Shared Risk Link Group-aware Proactive Path Computation

Finding backup paths with a minimum number of shared links greatly reduces the risk of breaking down both primary and backup from any single link failure. Thus computing backups with shared risk link group (SLRG) awareness can potentially improve their effectiveness during failures. Algorithm 2 illustrates the proposed SRLG-aware proactive path computation approach which pre-calculates backup paths during installation of accepted VNs.

Algorithm 2 SRLG Aware Proactive Path Computation

Input: flow path to calculate backup π_f; topology graph $G(V, E)$; max compute paths c_π^{max} ;

Output: selected backup path π_b;

1. **while** calculated paths $< c_\pi^{max}$ AND $\delta^{\pi_{ep}} \neq 0$ **do**
2. get next *Eppstein* path π_{ep} from source to destination of π_f on $G(V, E)$;
3. Calculate SRLG score $\delta^{\pi_{ep}}$ for path π_{ep}
4. **end while**;
5. **return** π_b $\leftarrow$ path with minimum $\delta^{\pi_{ep}}$;

This algorithm takes primary flow path, topology graph and upper bound of a number of shortest paths as inputs and returns an alternative path with a minimum number of common links with the primary path. For each allocated flow path π_f, maximum number of distinct shortest paths c_π^{max} is calculated by Eppstein algorithm employing monitored link latency values as edge weights. If the set of links in each shortest path π_{ep} is given by $S(\pi_{ep})$, and the set of links in the primary path is given by $S(\pi_f)$, then the SRLG score $\delta^{(\pi_{ep})}$ is calculated in line 3 by;

$$\delta^{(\pi_{ep})} = \frac{|S(\pi_f) \cap S(\pi_{ep})|}{|S(\pi_f)|} \tag{5.12}$$

In addition to computing backup paths after periodic VN embedding, recalculation is necessary after detecting a link failure since both primary and backup paths on that link need alternative paths. However, this does not affect the critical path of operation since it can be performed passively in both cases. In the next Section, the overhead of backup path calculations after link failure is analyzed, along with other performance measurements.

5.6 Maintaining Data-Plane Consistency During Flow Configuration

Both QoS-aware reactive and SRLG-aware proactive path computation algorithms rely on dynamic flow configuration features offered by SDN to delete interrupted paths and install alternative routes after detecting a failure. Dynamic flow configuration allows network routes to be computed at the central controller applications, and derive and send flow rules to corresponding forwarding elements in the new path. Similarly, the controller must send flow delete/modify messages to forwarding elements in the older path to guarantee new packets do not keep following the same broken route. If all forwarding elements in a path receive flow install, delete or modify messages with in few milliseconds, network state will be predictable and will behave as intended. However if flow rules arrive at forwarding elements with varying intervals due to congestion, distance (speed-of-the-light delay), buffering and other delays, the network might go through several transitional states which can lead to unpredictable outcomes such as routing loops, controller channel overloading and loss of management plane access to devices. In addition, severe packet losses can be expected and also security issues can be caused by forwarding packets to unintended destinations

In datacenter networks, these issues are highly unlikely to occur due to two main reasons. First, geographical distances between controller and forwarding elements are very low and do not vary significantly for different switches. Second, typically in datacenters, a dedicated management network is maintained to access control interfaces of servers and switches, which can be used to send flow rules to forwarding elements. Hence in datacenters, latency between controller and forwarding elements do not vary significantly. However the same is not true for wide area networks such as inter-datacenter networks where the controller messages can reach forwarding elements with highly varying delays. On the other hand, SDN technologies such as OpenFlow do not explicitly specify how to guarantee data-plane consistency while changing substrate paths corresponding to exist-

ing VLs. As a result, some forwarding elements may forward packets based on old rules and some may use new rules, causing aforementioned network state issues.

To avoid such undesirable network states resulting from dynamic path configuration, in this thesis it is proposed to employ flow versioning based per-packet consistency approach presented in [169]. With per packet flow versioning, a header field in each packet is reserved to carry a flow version number which is used to identify flow rules that must be used to match and forward the packet in each forwarding element. Thus the per packet consistency guarantees that every packet entering the network either follows old or new forwarding rules, but not some arbitrary combination of both. Although flow versioning do not avoid packet losses, it provides a simple and effective solution to guarantee data-plane consistency during transitional states caused by SDN-TE flow install/delete/modify operations. As a result. flow versioning makes the changes more predictable.

According to the authors of [169], version numbers can be carried in any packet header field that can be accessed through OpenFlow match operations. Thus they suggest VLAN header fields as suitable option to include version numbers. However in the framework implementation presented in this chapter, VLAN header fields are employed to distinguish VNs corresponding to different IaaS requests. Hence any other non-conflicting header field such as IP DSCP (6 bits in ToS field), differentiated service bits or IP ECN (2 bits in ToS field) can be used to include version numbers. In such cases, due to the limited number of bits in such fields, it is necessary to reuse previous version numbers upon reaching the maximum capacity of the header field.

Algorithm 3 Reactive Path Computation and Installation with per-packet consistency

Input: l^d , F^R, c_π^M , b^m , $G(W^s, L^s)$ where $l^d \notin L^s$;
Output: F^a, F^u,

1. $C^f \leftarrow C^f + 1$
2. **if** $C^f > C^{f\_MAX}$:
3. $C^f \leftarrow 0$
4. **for** each flow f in F^R
5. delete installed rules and release allocated bandwidth
6. **for** each flow f in F^R
7. **if** f *not in* $P_{f,\pi}^o$ **then**
8. **while** calculated paths $< c_\pi^M$ **do**
9. get next *Eppstein* path π_{ep} from source to destination of f on $G(W^s, L^s)$;
10. **if** path bandwidth $b(\pi_{ep}) < b^m$ **continue;**
11. **else if** requested bandwidth by flow $b(f) \leq b(\pi_{ep})$
12. selected path $\pi_s \leftarrow \pi_{ep}$;
13. **break**;
14. **else**
15. add π_{ep} to low bandwidth path set Q^π;
16. **end if**;
17. **end while**;
18. **if** $|Q^\pi| \neq 0$ *AND* requested bandwidth not available **then**
19. $\pi_s \leftarrow$ *Highest available path bandwidth*(Q^π);
20. **end if**;
21. **else**
22. $\pi_s \leftarrow$get and remove path of $f$from $P_{f,\pi}^o$;
23. **end if**;
24. **if** $\pi_s \neq \emptyset$ **then**
25. add (f, π_s) to $P_{f,\pi}^o$ and f to F^a;
26. **for** each node $w \in \pi_s$:
27. create a flow rule f^* for f with C^f in header
28. install flow f^* on w;
29. **else**
30. add f to F^a;
31. **end if**;
32. **end for**;
33. **return** F^a and F^u;

Let current version number installed in flow rules in the network corresponds to previous failure is given by C^f and the maximum version number that can be carried in the header field is denoted by C^{f_MAX}. Per-packet consistency can be integrated to the proposed QoS aware reactive path computation algorithm as shown in Algorithm 3.

5.7 OpenFlow-based Implementation and Performance Evaluation

In this thesis, to evaluate the performance of the proposed framework during link failures, a prototype was implemented. Networked cloud infrastructures were created on Mininet, emulating the topology of 2007-2008 North American AT&T network connecting 20 major cities [151]. NRM was developed in python as POX [48] controller application and IaaS resource allocator and its internal virtual node and link embedding algorithms were developed in C++.

IaaS requests were generated using open source, GT-ITM topology generator [152]. IaaS request topologies were created using Waxman [153] random graph model with $\alpha = 0.5$ and $\beta = 0.2$ and 10% of the requests have star connected topologies while remaining 90% of the requests and all the substrates have general random topologies. Each generated IaaS request has 2 to 4 virtual nodes and 1 to 3 VLs, with each randomly requesting 5% - 25% of substrate node and link QoS resources. The ILP problems for both virtual node and link embedding were programmatically formulated and solved in CPLEX [154]. A JSON [155] based API was also implemented over TCP sockets for faster, light-weight interaction between NRM and IaaS resource allocator. Developed API isolated optimization solver and SDN controller allowing simultaneous execution in separate machines with minimum performance impact on each other.

5.7.1 Performance Metrics and Benchmarks

In this research an analysis was conducted on the performance of the following proposed failure restoration approaches:

- QoSaware reactive path computation with bandwidth under allocation (React_under),
- SRLG aware proactive shortest path computation with monitored linklatency values as edge weights (Proact_min-lat_SRLG).

The above approaches were evaluated with respect to follow-ing as benchmarks:

- QoSaware reactive path computation with exact bandwidth allocation (React_exact),
- Proactive minimum hop-based backup path computation (Proact_minhop) and,
- Without any restoration (No_reroute).

The Mininet CLI was modified and a function was introduced to randomly select and remove 10 substrate links, by sequentially calling link-down functions. Each substrate link failure generated two unidirectional link failure events, resulting in 20 measurement points in each experiment. Average values were obtained by running experiments for 10 iterations for the following performance metrics;

1. Bandwidth utilization: percentage of total bandwidth used over total bandwidth available in the network.
2. VL blocking ratio (inverse of acceptance ratio): number of VLs not rerouted over the total number of VLs initially installed (before introducing link failures).
3. Number of flows: variation of the total number of flow rules installed in all switches in the network with after failures.
4. TCAM utilization: VL restoration percentage over normalized flow rule allocation.
5. Path re-allocation time: time to install new flow-rules after detecting a link failure.
6. Path re-computation time: total time to calculate and install flow-rules after detecting the failure.

5.7.2 Analysis of Experimental Results

The bandwidth was allocated to VLs by configuring QoS queues in switches and openflow.ofp_action_enqueue function was used to dynamically assign flows to output queues

during flow installation. Bandwidth utilization was calculated as a measurement of network resource utilization efficiency. The results for all five cases are graphed in Fig. 5.5.

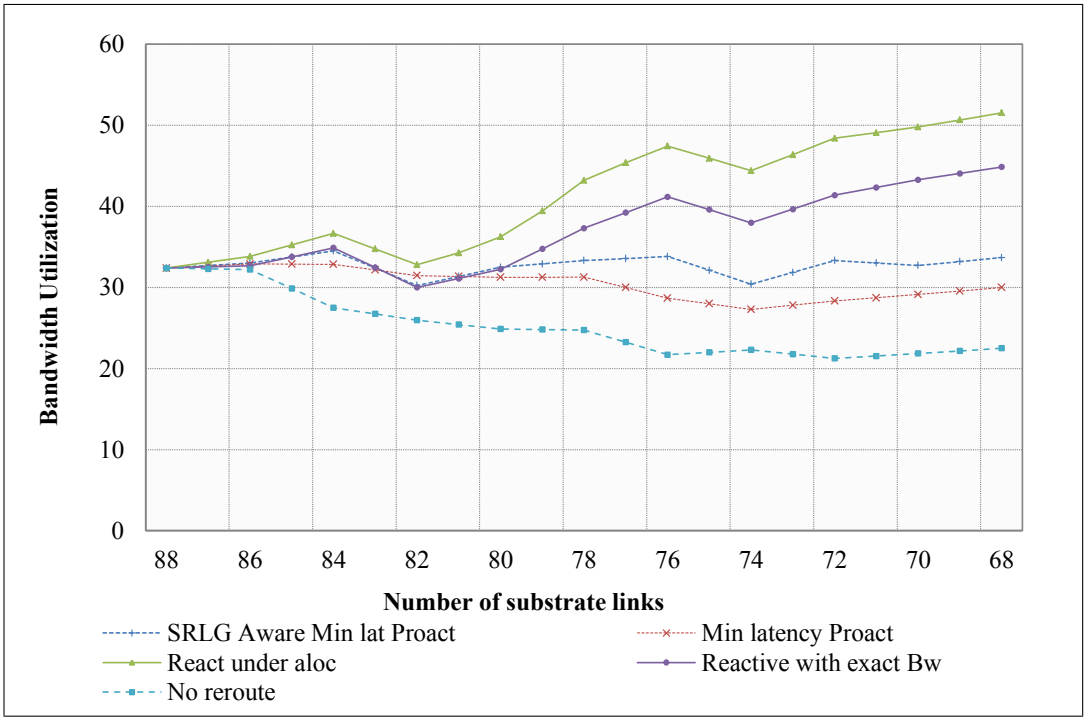


Figure 5.5: Variation of Bandwidth utilization with respect to number of uni-directional links available in the substrate network

According to the graph, it can be seen that React_under allocation gives the highest network utilization, followed by React_exact, Proact_minhop_SRLG and Proact_minhop respectively, with least utilization in No_reroute. This observation is justified by the bandwidth relaxation adopted in React_under. When flexibility is given to reserve lesser than requested bandwidth to VLs affected by failures, more VLs can be accommodated in the available resources. For certain application scenarios, such relaxed resource allocation is beneficial instead of completely rejecting them when the exact amount of resources are not available.

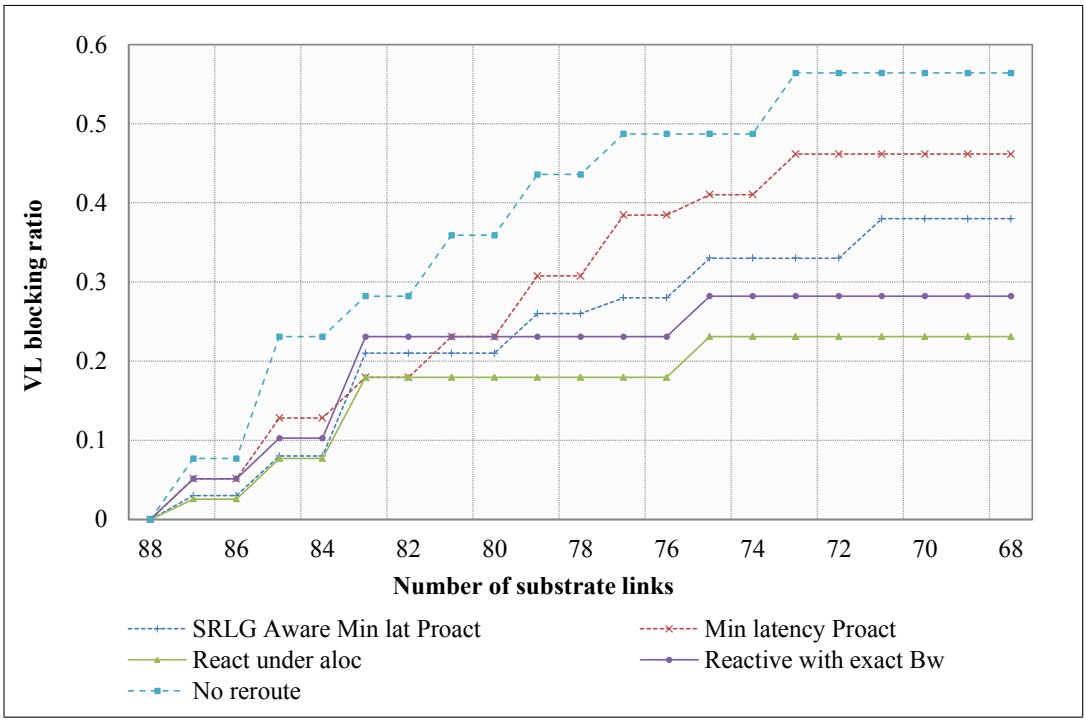


Figure 5.6: Virtual link blocking ratio with respect to number of uni-directional links available in the substrate network

Further, this observation is evident in Fig. 5.6, where React_under demonstrate lowest blocking ratio (i.e. highest acceptance ratio). In the figure, both proactive approaches show better utilization and acceptance over No_reroute, but relatively lower network utilization and acceptance ratio with respect to reactive approaches. To provide better clarification, the VL restoration ratio was calculated for different link failure percentages. For a given number of substrate link failures, the VL restoration ratio was calculated by dividing the total number of VLs restored from the total VLs broken. The results are tabulated in Table 5.1.

Although in this research we have integrated bandwidth relaxation has been integrated into proactive approaches, they still yield lower acceptance and utilization due to their inability to account for changes in bandwidth availability during path computation. For example, after removing 20% of the substrate links, Proact_min-lat_SLRG and Proact_min-hop successfully restored 32.6% and 18.2% of failed VLs com-pared to 59.1% and 49.9% restorations in reactive approaches. However, relative to the Proact_min-hop

Table 5.1: Percentage of virtual links restored

% of links failed	Proact_min-lat_SRLG	Proact_min-hop	React_under	React_exact
5%	65.3	44.4	66.7	55.5
10%	41.5	35.7	50.0	35.7
15%	42.5	21.1	63.2	52.6
20%	32.6	18.2	59.1	49.9

approach, the Proact_min-lat_SRLG approach demonstrate improved performance as the latter minimizes the probability of single link failure affecting both primary and backup paths.

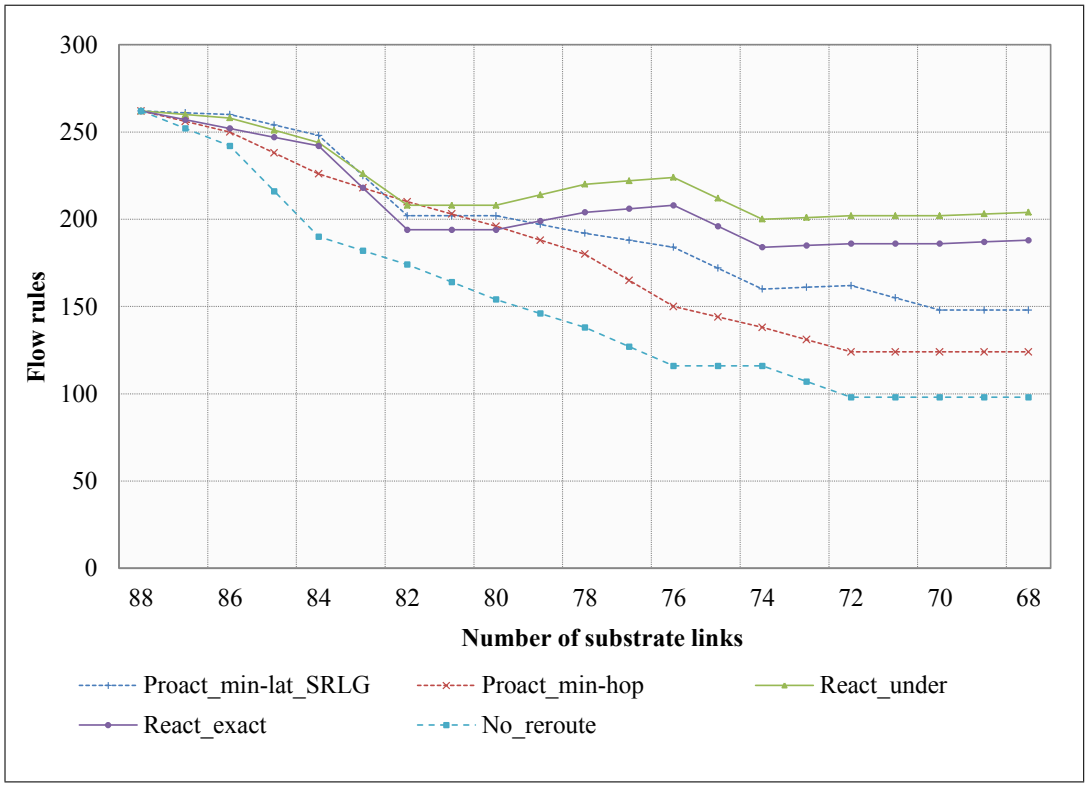


Figure 5.7: Number of flow-rules installed with respect to number of uni-directional links available in the substrate network

Compared to legacy networks, SDN introduces an additional constraint due to limited availability of ternary content-addressable memory (TCAM) in switches for flow rules. The total number of flow rules installed in the network after each link failure was

calculated and plotted in Fig. 5.7. According to the graph, React_under installed the most number of flow rules and Proact_min-hop installed the least number, out of all four restoration approaches. Since these raw measurements are not sufficient to reach a meaningful conclusion, in this research TCAM utilization efficiency has been defined by dividing VL restoration ratio from a normalized number of flow rules. The number of flow rules installed is normalized by the total number of flow rules installed before introducing failures (after previous VN embedding iteration). The resulting percentage values are plotted in Fig. 5.8.

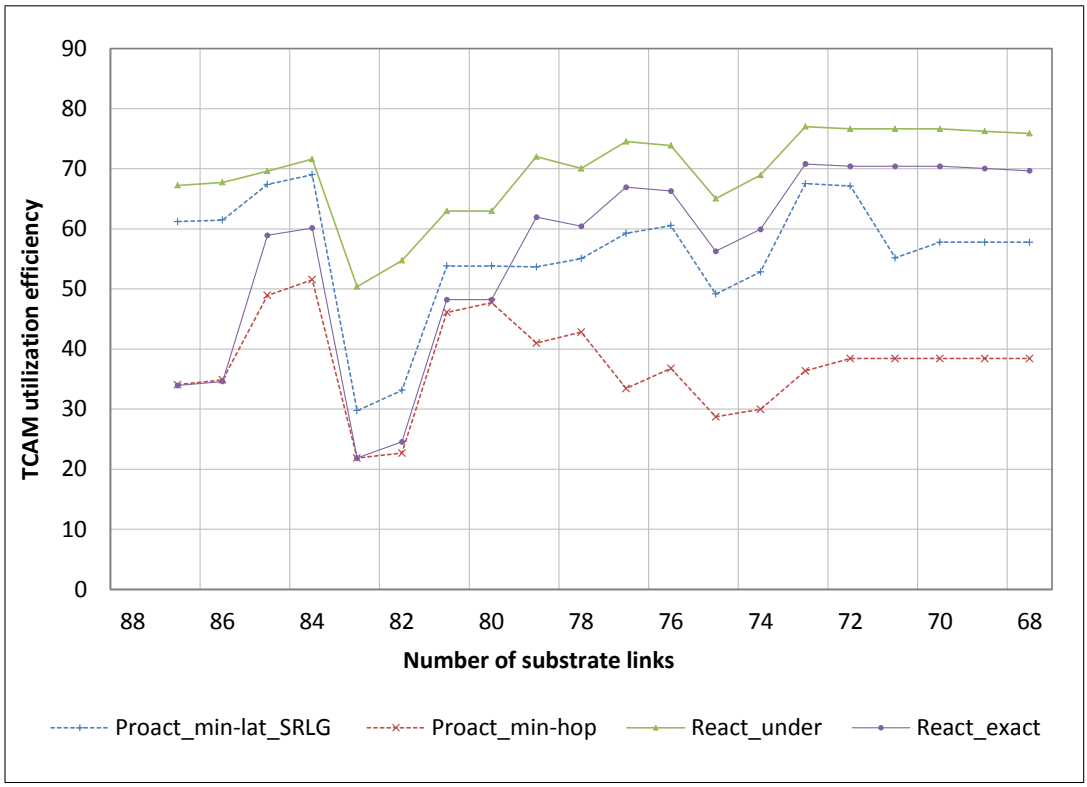


Figure 5.8: Ternary content-addressable memory utilization efficiency with respect to number of uni-directional links available in the substrate network

From this graph, it can be clearly seen that, despite the fact that React_under installed more flow-rules, it has used switch flow tables most efficiently compared to all the other restoration approaches. On the other hand, Proact_min-hop, which showed least flow-rule installations, has resulted in significantly lower TCAM utilization efficiency.

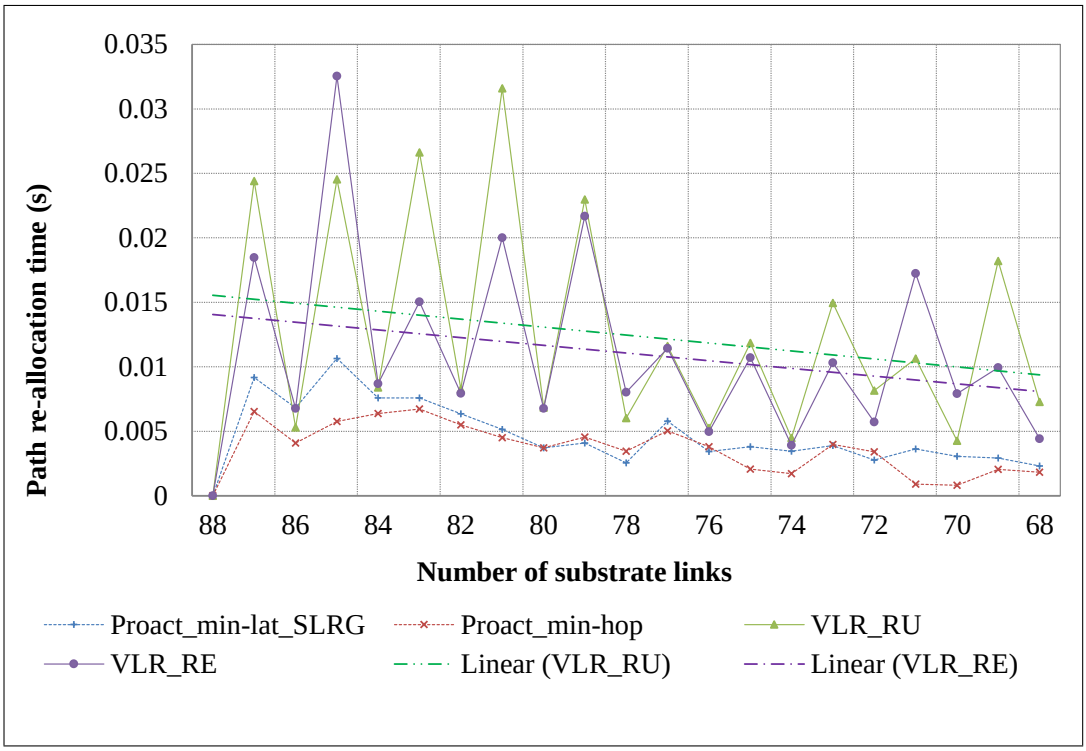


Figure 5.9: Path re-allocation time with respect to number of uni-directional links available in the substrate network

The graph in Fig. 5.9 illustrates the time taken by NRM to send flow rules to switches after detecting failures in each restoration approach. As expected, both proactive backup-path computation approaches generated flow rules faster than reactive approaches, since path calculations done offline (before and after restoration). High variation in reactive path computation times (repetitive saw-tooth pattern) has been observed due to the one-time calculation of alternative routes, for both unidirectional link failure events, raised from bidirectional link failure. In Fig. 5.10, total time to calculate and install paths was measured by including offline path computation times of proactive approaches.

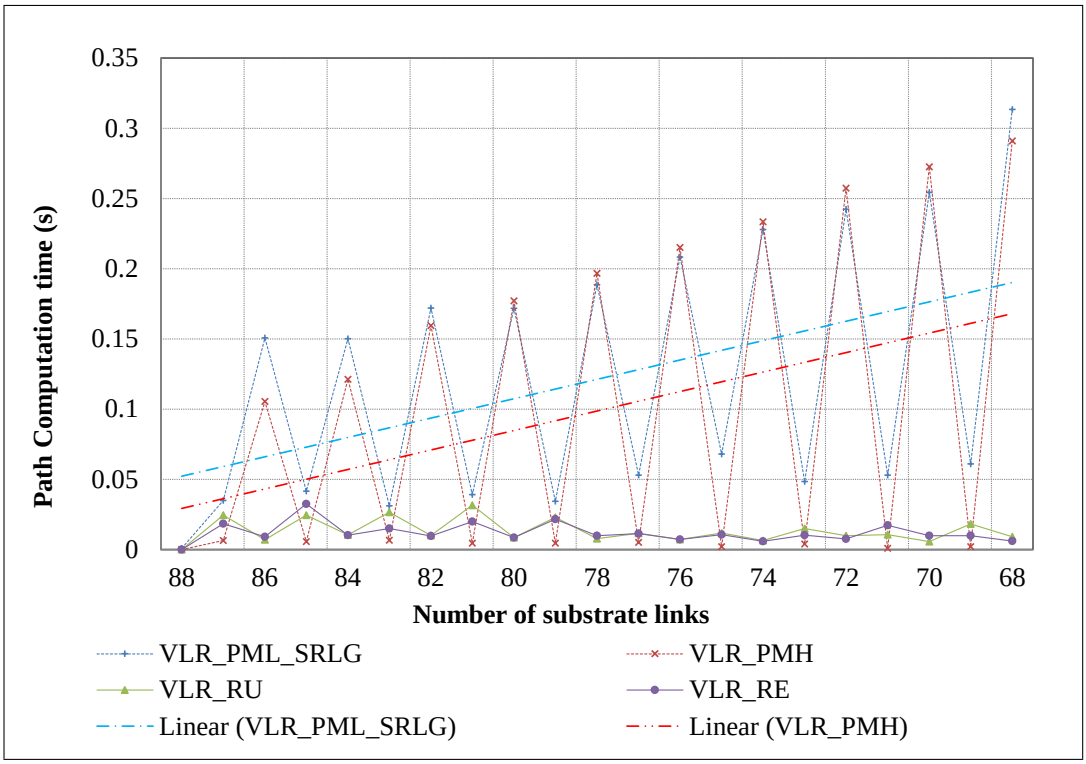


Figure 5.10: Path re-computation time with respect to number of uni-directional links available in the substrate network

With regard to reactive approaches, both proactive approaches displayed significantly higher path computation times during each first unidirectional link event. This is due to higher path computation overheads in proactive approaches incurred by recalculations of backup paths that were on failed link. Slightly higher delays were noticeable in the Proact_min-lat_SLRG approach over Proact_min-hop in second unidirectional link event. This can be attributed to higher success rates in the former approach since it required deriving more flow rules to install accepted backup paths.

Also a decrementing trend was noticeable in proactive path computation and an incrementing trend in proactive path computation times as highlighted by trend lines in Fig. 5.9 and Fig. 5.10. For reactive approaches, this is due to the reduction in path computation complexity with a lower number of links in the topology. On the other hand, for proactive approaches, when the number of substrate links decreases, the number of backup paths per link increases, escalating backup path calculation overhead. Thus in a

scenario where multiple substrate link failures occurred as a result of node failures or from natural disasters, proposed SDN-based proactive and reactive approaches demonstrate unique failure restoration characteristics, that can be applicable based on service provider requirements.

5.8 Chapter Summary

In this Chapter, the design and implementation of an IaaS management framework have been detailed that is capable of performing optimum IaaS resource allocation and fault tolerant operation in an SDN-based networked cloud environment. The proposed framework contains an ILP-based virtual node and link allocation approach that minimizes the cost of allocation and improves the IaaS provider resource utilization. Two SDN-TE based path restoration schemes were integrated into the framework, to perform QoS-aware failure restoration, while maintaining high network resource utilization. The proposed framework was implemented and several experiments were carried out to evaluate the performance of SDN-based proactive and reactive path computation strategies for failure restoration. Experimental results demonstrate that reactive path computation can successfully restore nearly 60% of failed links with better switch flow table utilization compared to 32.6% restoration in proactive path computation.

6 Cloud Network Resiliency With Shared Backup Protection

6.1 Introduction

With the popularity of cloud service paradigm, adopting fault-tolerant mechanisms that deliver high availability and low mean time to recovery (MTTR) has become a key requirement for infrastructure service providers. Particularly for wide area networks such as inter-datacenter networks, such guarantees are essential to maintain a high quality of service. Restoration techniques are capable of providing means to recover from failures while maintaining higher levels of network resource utilization. However, they lack the ability to guarantee QoS parameters such as latency and bandwidth following failures. Furthermore, delays associated with each and every step in the dynamic fault restoration process such as failure detection, localization, path re-computation and installation are typically hard to avoid especially in wide area networks as they are often tied-up with higher round-trip times to remote nodes. Hence, it is essential to study fault protection mechanisms that are capable of delivering guaranteed QoS and faster recovery from failures, while providing increased levels of substrate resource utilization.

In this work, VN protection schemes employed at the resource-allocation level were evaluated and compared to restoration schemes employed at the resource-operation. An IaaS resource management framework was designed that provides substrate link failure protection and restoration for VNs embedded in SDN-based NCI. Substrate resources were allocated with the objective of satisfying application QoS requirements while in-

creasing the IaaS providers' revenue by maximizing request acceptance and resource utilization. The IaaS resource allocation problem was formulated as a periodical Integer Linear Programming (ILP) problem that optimizes the allocation of compute and network resources on a geo-distributed NCI. A shared backup protection scheme was integrated into the virtual link embedding operation, which guaranteed fast and resource efficient recovery from any single substrate link failure. The proposed link-embedding technique increases the overall substrate resource utilization by maximizing shared backup bandwidth. The application of the proposed technique was demonstrated through the IaaS resource allocation framework presented in the previous chapter. The performance of the proposed backup bandwidth sharing technique was evaluated during single-random substrate link failures.

The remainder of this chapter is organized as follows. Section II, examines existing protection schemes, their features, benefits, and limitations in network virtualization scenarios. Section IV describes the changes to the IaaS resource allocation framework as described in Chapter 5 and the resource allocation problem formulation. As well, section V evaluates the performance of the proposed backup sharing techniques in an SDN/OpenFlow environment. Finally the chapter summery is presented in Section V.

6.2 Related Work

Although the cloud service model is relatively new, key aspects of fault-tolerance of its enabler technologies (compute, storage and network virtualization) have been fairly well studied. However, a number of recent proposals [158–160] highlight the importance of further investigating fault tolerant network virtualization because of the wide adoption of SDN and related technologies. Comprehensive classifications of existing research in VN resiliency can be found in [156, 157, 159]. In this section, we present on prominent fault tolerant schemes available in the literature which focus on link protection in physical and virtual domains.

Existing protection schemes can be further classified according to their focus on providing protection for the substrate links, the substrate paths, and at the VN level. VN level protection schemes [161,162] mainly propose enhancing resiliency by adding virtual nodes and/or links to the original VN topology (VN augmentation). Substrate link protection [137,163] relies on pre-configuring the physical network by reserving links (all or part of the bandwidth) to back up other links, prior to accepting IaaS requests. Substrate path protection [159] has been extensively studied as a cross-layer protection strategy that covers the full length of network paths reserved for virtual links, by reserving additional capacity during the VN embedding phase. Its popularity can be partly attributed to the wide adoption of flexible path configuration technologies such as MPLS-Traffic Engineering (MPLS-TE) and SDN. Research in substrate path protection extends in several directions, including dedicated path protection [164,165], multi-path protection [138] and shared path protection [140, 166, 167].

Dedicated path protection is usually employed to satisfy service level agreements (SLAs) with stringent QoS and fast recovery requirements. It has been widely studied with respect to optical transport networks [165]. Among works that are non-specific to optical, a dedicated back-up scheme is proposed in [164], guaranteeing fast recovery from any single node or link failure. The paper proposes an ILP-based solution and a much faster heuristic solution to provide 1+1 protection for the whole VN. However, the authors recognize that this guarantee of fast QoS recovery through dedicated protection comes at a high cost and may result in the substrate network being underutilized. Moreover, such extreme SLA requirements are only applicable to limited application scenarios.

A multi-path link embedding strategy for VN survivability is presented in [138]. It requires only a fraction of the virtual link (VL) bandwidth for backups. Though this approach can potentially improve fault tolerance and bandwidth use, the authors acknowledge that the path splitting may introduce additional overhead in terms of the number of routing entries. Some cloud applications (such as live streaming and video conferencing) may also inherently not accommodate path splitting due to varying trans-

mission delays. Furthermore, in an SDN environment, path splitting can significantly impair VN request acceptance, as pointed out in [139].

One of the main advantages of path protection is its ability to strategically improve substrate resource utilization by sharing backup bandwidth among multiple VNs. In [166], a VN-embedding approach is presented with shared backup embedding for survivability along with ILP formulations. The model assumes that VNs arrive sequentially and are embedded one at a time. Although this sequential embedding simplifies the formulation, it may result in sub-optimal network resource use in contrast to batch processing. Furthermore, the authors do not illustrate how they guarantee disjoint-ness in backup paths during the embedding phase.

An OpenFlow path protection mechanism that maintains backup paths for fast link failure recovery is introduced in [140]. In this approach, flow rules need to be installed in corresponding switches for primary and backup paths. In [167], SDN-based path protection is enhanced with VLAN-based flow aggregation to minimize the flow entries required for primary and backup paths. Although this approach efficiently uses ternary content-addressable memory (TCAM) space in OpenFlow switches, flow aggregation is not effective in network virtualization environments where traffic and resource isolation are considered as critical requirements.

6.3 Fault-Tolerant Infrastructure-as-a-Service Embedding

Several modifications has been made to the resource allocation framework presented in Chapter 5 to allocate network and compute resources to IaaS requests in a networked cloud environment. These include adding backup path management functions to network resource manager (NRM) as well as enhancing VL embedder with backup path embedding functionalities as shown in Fig. 6.1.

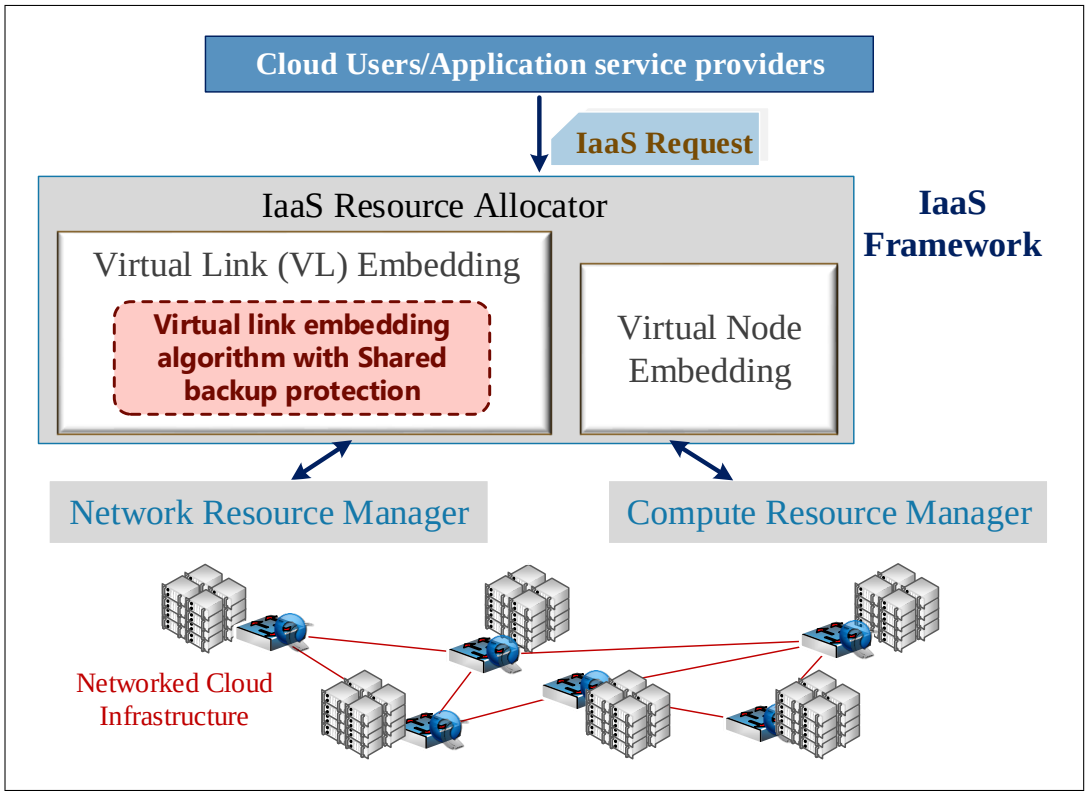


Figure 6.1: Integration of shared backup virtual link embedding algorithm in the cloud resource allocation framework

In order to include maximization of backup bandwidth sharing along with link embedding objectives, the VL embedding solution presented in chapter 5 has been completely re-formulated. Remainder of this section provides a detailed description of the proposed ILP solution used for IaaS embedding, with example illustrations.

6.3.1 Infrastructure-as-a-Service Embedding Problem

In a typical ILP-based IaaS allocation framework, requests that are processed through periodical VN embedding can be categorized into one-shot and 2-phase embedding [148]. In one-shot, virtual node and link embedding is formulated as a single ILP problem and solved with the intention of achieving a globally optimized solution. In contrast, 2-phase embedding formulates node and link embedding as two ILP problems where the node embedding result is used as an input for the link embedding. Typically one-shot embedding yields better solutions at the expense of higher computational complexity.

In addition, the complexity can be escalated if backup-path embedding is also included into one-shot embedding. Hence, in this work, a coordinated node and link embedding technique was employed which was initially proposed in [147]. The 2-phase coordinated node and link embedding has proved to be an effective optimization strategy that yield better results with reduced embedding complexity. However, the link embedding formulation presented in this thesis is significantly different from the link embedding approach followed in [147]. This is mainly because, in this work, link embedding formulation extended with backup path embedding. Thus, in particular for inter-datacenter IaaS resource allocation scenario, we perform node embedding as follows:

The inter-DC network is represented by an undirected graph $G^s(W^s, L^s)$, where W^s denotes the set of substrate nodes (DCs) and L^s the set of bidirectional inter-DC links. The set of IaaS requests arriving at time period t is given by $V^n(t), n \in N = 1, 2, .., |N|$ such that each request $v^n \in V^n(t)$ is represented by a directed graph $G^n(A^n, E^n)$, where A^n denotes the set of virtual nodes and E^n the set of directed links. Summarized definitions of the parameters used to model substrate network, IaaS requests and active/backup embedding operation are shown in Table 1.

The QoS requirements of each VL $e \in E^n$ belonging to class $i \in Q^E = 1, 2, .., |Q^E|$ are defined by couple (b^i, h^i), where b^i is the required bandwidth and h^i is the maximum number of hops. Likewise, the QoS requirements of each virtual node $a \in A^n$ belonging to class $j \in Q^A = 1, 2, .., |Q^A|$ are defined by the vector (p_j, m_j, s_j, r_j), where p_j, m_j, s_j and r_j are required CPU, memory, storage and preferred set of substrate embedding locations respectively. Let $\pi_(u, v)^e$ denote a network path between DC nodes (u, v) assigned to e and $L(\pi_(u, v)^e)$ gives the number of substrate links in $\pi_(u, v)^e$. s and d are the source and destination nodes of virtual link e.

6.3.2 Virtual Node Embedding

To decide on the acceptance of a VN request v^n, in this work the binary variable z^n, such that $z^n = 1$ if v^n is accepted to use node resources and $z^n = 0$ otherwise. Similarly, x_a^u is

Table 6.1: Notations for infrastructure-as-a-service allocation problem

Parameters	Description
	Substrate Network $\boldsymbol{G^s(W^s, L^s)}$
$\boldsymbol{W^s}$	Set of substrate nodes (Datacenters)
$\boldsymbol{L^s}$	Set of substrate bidirectional links (inter-datacenter links)
$\boldsymbol{B^l(t)}$	Residual bandwidth capacity of link $l \in L^s$ at time period t
$\boldsymbol{c_b^l(t)}$	Bandwidth unit cost of $l \in L^s$ at t
$\boldsymbol{P^w(t), M^w(t), S^w(t)}$	Residual CPU, memory and storage capacity of node $w \in W^s$ at t
$\boldsymbol{c_p^w(t)}$, $\boldsymbol{c_m^w(t)}$, $\boldsymbol{c_s^w(t)}$	CPU, memory and storage unit costs of w at t
	IaaS Request $\boldsymbol{G^n(A^n, E^n)}$
$\boldsymbol{V^n(t)}$	Set of IaaS requests arrived during t
$\boldsymbol{A^n}$	Set of virtual nodes requested in the request $v^n \in V^n(t)$
$\boldsymbol{Q^A}$	Set of virtual node QoS classes
$\boldsymbol{p_j}$, $\boldsymbol{m_j}$, $\boldsymbol{s_j}$	Required CPU, memory and storage of node QoS class $j \in Q^A$
$\boldsymbol{r_j}$	Potential set of embedding locations for QoS class j
$\boldsymbol{c(a)}$	QoS class of virtual node $a \in A^n$
$\boldsymbol{E^n}$	Set of bidirectional virtual links requested in $v^n \in V^n(t)$
$\boldsymbol{Q^E}$	Set of virtual link QoS classes
$\boldsymbol{b^i}$	Required bandwidth for link QoS class $i \in Q^E$
$\boldsymbol{h^i}$	Maximum hop count for link QoS class i
$\boldsymbol{c(e)}$	QoS class of virtual link $e \in E^n$
	Embedding
$\boldsymbol{s}$, $\boldsymbol{d}$	Source and destination virtual nodes of virtual link $e \in E^n$
$\boldsymbol{u}$, $\boldsymbol{v}$	Source and destination datacenter nodes selected for s and d
$\boldsymbol{\Pi_{u,v}^e}$	Set of network paths between u, v calculated for e
$\boldsymbol{L(\pi_{u,v}^e)}$	Number of substrate links that form the path $\pi_{u,v}^e \in \Pi_{u,v}^e$
$\boldsymbol{P^e}$	Price offered by the IaaS requester for a given $e = (s, d)$ in v^n

defined such that, $x_a^u = 1$ if virtual node a is assigned to substrate node $u \in W^s$ and $x_a^u = 0$ otherwise. For each time interval t , the objective is to maximize the profit by allocating DC node resources to requests. To abstract source and destination substrate node CPU, memory and storage costs, C_s^u and C_d^v are defined such that $C_s^u = (c_p^u p_s + c_m^u m_s + c_s^u s_s)$ and $C_d^v = (c_p^v p_d + c_m^v m_d + c_s^v s_d)$. If P^e is the price offered by v^n for $e = (s, d)$, the objective function can be written as:

$$max \sum_{n \in N} \left[\sum_{e \in E^n} P^e z^n - \sum_{(u,v) \in W^s \times W^s} \left[C_s^u + C_s^v + c_b^l b^(c(e)) L(\pi_{u,v}^e) \right] x_s^u x_d^v \right] (1) \qquad (6.1)$$

Subject to the following constraints:

i Embedding is done for all nodes of an accepted V_n:

$$z^n \leq \sum_{(u,v) \in W_s \times W_s} x_s^u x_d^v; \qquad (s, d) = e \in E^n, n \in N \qquad (6.2)$$

ii Node a can be assigned to only one substrate node u:

$$\sum_{u \in r_{c(a)}} x_a^u \leq z^n; \qquad a \in A^n \qquad (6.3)$$

iii Node a can be assigned only to a subset of substrate node regions $r_{c(a)}$:

$$x_a^u = 0; \qquad \forall u \in W^s, u \notin r_{c(a)}, \forall a \in A^n, n \in N \qquad (6.4)$$

iv Residual node CPU capacity:

$$\sum_{n \in N} \sum_{a \in A^n} x_a^u p_{c(a)} \leq P^u(t) - P^u(t-1); \qquad c(a) \in Q^A, u \in W^s \qquad (6.5)$$

v Residual node memory capacity:

$$\sum_{n \in N} \sum_{a \in A^n} x_a^u m_{c(a)} \leq M^u(t) - M^u(t-1); \qquad c(a) \in Q^A, u \in W^s \qquad (6.6)$$

vi Residual node storage capacity:

$$\sum_{n \in N} \sum_{a \in A^n} x_a^u s_{c(a)} \leq S^u(t) - S^u(t-1); \qquad c(a) \in Q^A, u \in W^s \tag{6.7}$$

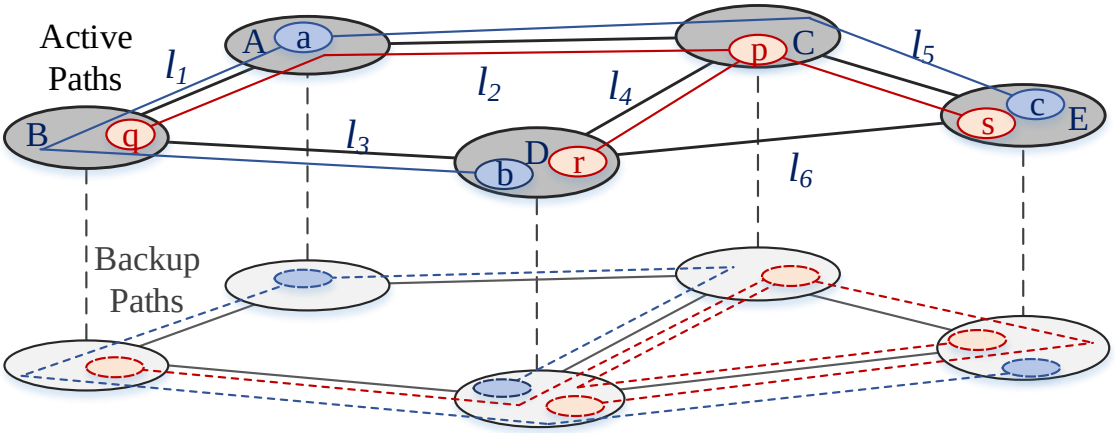
In the node embedding formulation, virtual nodes are embedded with a fair consideration to link resource availability between chosen end nodes. This coordination between node and link embedding guarantees improved performance in the VN embedding process.

6.3.3 Fault Tolerant Virtual Link Embedding

The shared backup embedding approach involves obtaining substrate links allocated to existing (retained) IaaS requests and identifying which portion of the backup bandwidth in each link can be reused for new backups. The following steps are used to create sets of active-backup pairs for each virtual link in a given IaaS request. To better explain the steps, a sample scenario is provided in Figs. 6.2 - 6.4.

i Obtain existing VL allocations: If $N(t-1)$ is the set of IaaS requests allocated and still alive at the end of previous time period $(t-1)$, determine the set of VLs given by $E^n(t-1)$ for each $n(t-1) \in N(t-1)$.

ii For each VL $e^n(t-1) \in E^n(t-1)$, calculate the set of substrate links used for active paths Π^e and backup paths Φ^e. For example, assume VLs corresponding to IaaS 1 and 2 are already embedded and exist by the end of period $t-1$ as shown in Fig. 6.2.



	Virtual nodes	Embedded nodes	Active links	Backup links	BW req.
IaaS 1	a, b	A, D	1, 3	2, 4	3
	a, c	A, E	2, 5	1, 3, 6	2
IaaS 2	p, q	C, B	1, 2	3, 4	1
	p, r	C, D	3, 6	1, 2, 5	3
	p, s	C, E	5, 6	2, 1, 3	2

Figure 6.2: Identifying existing virtual link and their requested bandwidth

iii Calculate δ matrix: For a given substrate link $l^* \in L^s$, $\delta_l^{l^*}$ is the total backup bandwidth allocated to existing VLs, in which the active path uses the link $l \in L^s$. The binary variable $\Phi^e_{l^*,l}$ is defined such that $\Phi^e_{l^*,l} = 1$ if VL e use l^* in the backup path and l in the active path and $\Phi^e_{l^*,l} = 0$ otherwise. If the bandwidth demanded by each VL $e(t-1)$ is denoted by $b^c(e)$, $\delta_l^{l^*}$ can be written as:

$$\delta_l^{l^*} = \sum_{n \in N(t-1)} \sum_{e \in E^n} b^{c(e)} \Phi^e_{l^*,l}; \forall l^*, l \in L^s \times L^s, l^* \neq l \tag{6.8}$$

Fig. 6.3(a) shows the δ matrix calculated for the sample embedding scenario illustrated in Fig. 6.2.

iv Determine backup bandwidth needed for existing VLs: To recover from any single substrate link failure, the total backup bandwidth needed in each substrate link l^*

is equivalent to the largest $\delta_l^{l^*}$ value given by

$$B_{l^*}^b(t-1) = \max_{l \in L^s, l^* \neq l} \delta_l^{l^*} \tag{6.9}$$

This can also be obtained as the maximum column vector from δ matrix as shown in Fig. 6.3(a).

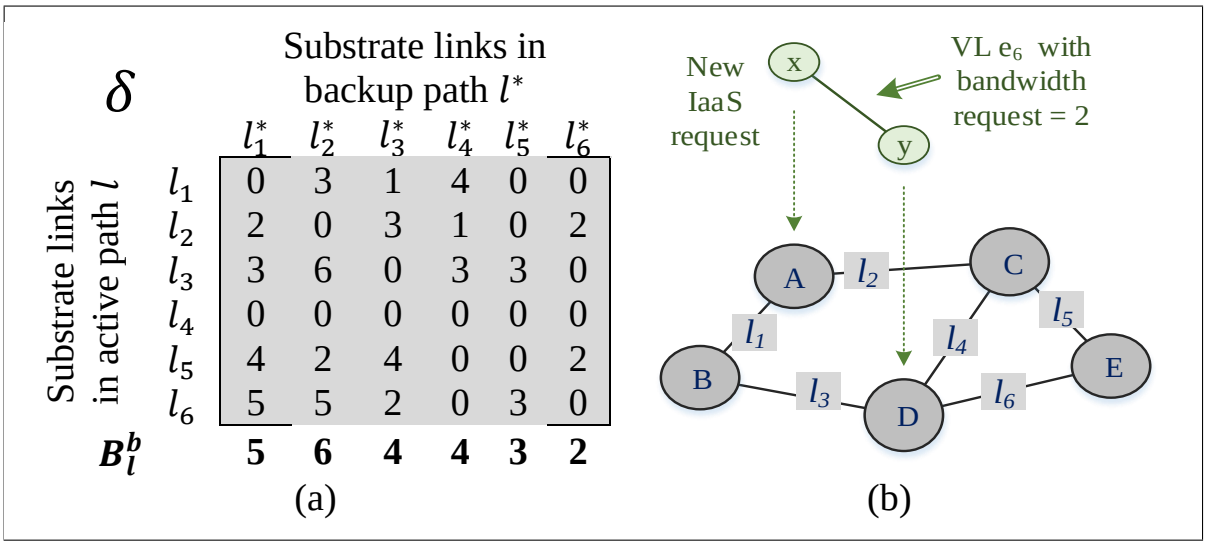


Figure 6.3: (a) δ matrix and max column vector B_l^b calculated for existing virtual link at the end of $t-1$ (b) New infrastructure-as-a-service request arrived at t

v Find active paths for the new requests: If a set of $N(t)$ IaaS requests arrives in the current time period t, such that given request $n \in N(t)$ contain $E^n(t)$ set of VLs, we employ a k-shortest path algorithm with equal unit link weights to find set of $\Pi^e(t)$ active paths for each VL $e \in E^n(t)$.

vi Compute θ matrix: If the active bandwidth demanded by e is $b^c(e)(t)$, the θ matrix for each active path $\pi \in \Pi^e(t)$ is calculated such that, for each $l^* \in L^s, \theta_l^{l^*}$ are the additional backup bandwidth units required on l^* to backup $l \in \pi(t)$. In this work, $\theta_l^{l^*}$ values were calculated similar to the cost model proposed in [168]. If residual

bandwidth in l^* at period t is given by $B^r_{l*}(t)$, $\theta^{l^*}_l$ is computed as follows;

$$\theta^{l^*}_l = \begin{cases} 0 & \text{If } l^* \neq l \text{ and } \delta^{l^*}_l + b^{c(e)} \leq B^b_{l^*} \\ \text{delta}^{l^*}_l(t-1) + b^{c(e)}(t) & \text{If } l^* \neq l \text{ and } \delta^{l^*}_l(t-1) + b^{c(e)}(t) > B^b_{l^*}(t-1) \text{ and} \\ \text{B}^b_{l^*}(t) & \text{B}^r_{l^*}(t) \geq \delta^{l^*}_l(t-1) + b^{c(e)}(t) - B^b_{l^*}(t-1) \\ \infty & \text{Otherwise} \end{cases} \tag{6.10}$$

In the given example, assume a new IaaS request arrives at the link embedding phase with its two virtual nodes x and y already embedded on substrate nodes A and B respectively. If the VL e_6 between x and y requests 2 units of bandwidth, as shown in Fig. 6.3(b), the θ matrix computed for e_6 is given in Fig. 6.4(a).

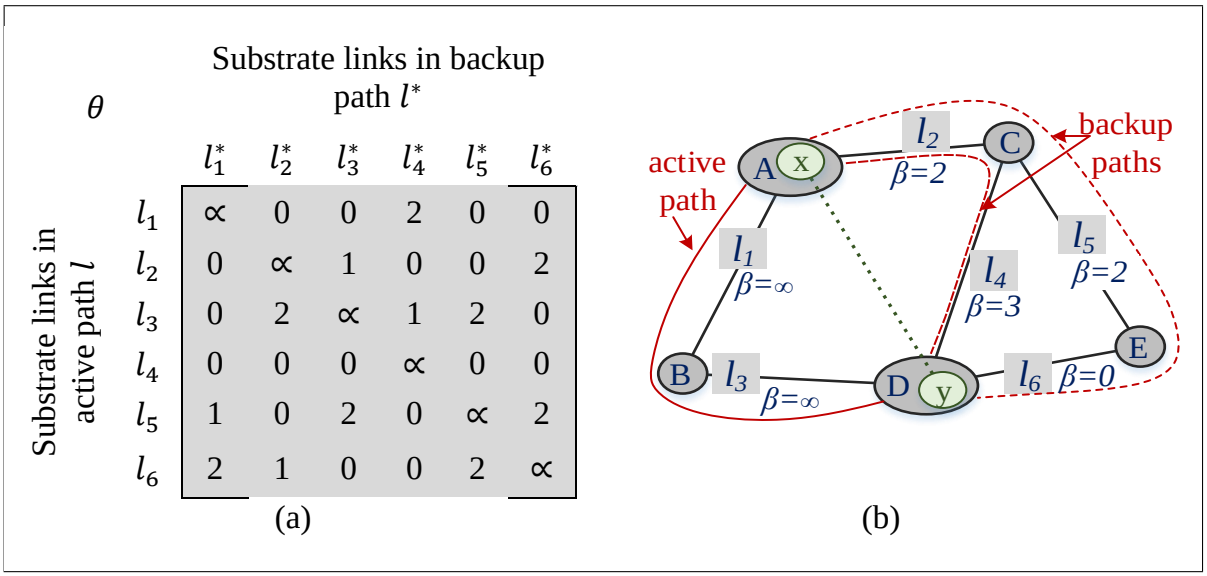


Figure 6.4: (a) θ matrix calculated for new virtual link e_6, (b) An active path and corresponding backup paths calculated for e_6

vii Find $\beta^{l^*}_\pi$ values for each substrate link: For the active path π, total additional bandwidth units required on l^* to successfully back up the failure of any $l \in \pi$ is given by $\beta^{l^*}_\pi = \max_{\forall l \in \pi} \theta^{l^*}_l$ As shown in Fig. 6.4(b), for a given active path between x and y that spans over links l_1 and l_3, $\beta^{l^*}_\pi$ values can be calculated for all substrate

links. Resulting $\beta_{\pi}^{l^*}$ values are also depicted in Fig. 6.4(b)

viii Finding backup paths: For each π, after finding $\beta_{\pi}^{l^*}$ values for every link in the substrate network topology, k-shortest path algorithm [150] is used again to calculate a set of backup paths considering products of each link's bandwidth unit cost and $\beta_{\pi}^{l^*}$ values (i.e. $c_b^l \beta_{\pi}^{l^*}$) as link weights. Finally, active-backup pairs are created by pairing each active path with corresponding backup paths.

Let a backup path calculated for the active path π be denoted by ρ^{π}. If the set of all active-backup pairs computed for VL e is denoted by Γ^e, it can be written as:

$$\Gamma^e = \Big\{(\pi_1, \rho_1^{\pi_1}), (\pi_1, \rho_2^{\pi_1}), ...(\pi_2, \rho_1^{\pi_2}), ...\Big\} \tag{6.11}$$

The set Γ^e is considered as input in the proposed ILP formulation in order to embed e in the substrate network.

ILP Formulation for link embedding with shared backup

In this section, the proposed ILP formulation developed to perform VL embedding is illustrated by allocating substrate link resources to active and backup paths simultaneously. If an active-backup pair calculated for e is given by $\tau \in \Gamma^e$, the embedding binary variable y_{u*v*}^{τ} can be defined such that $y_{u*v*}^{\tau} = 1$ if active-backup pair τ between substrate nodes u^* and v^* is assigned to VL $e = (s, d)$, and $y_{u*v*}^{\tau} = 0$ otherwise. $\alpha_l(\pi_e)$ and $\alpha_l(\rho^{\pi})$ are introduced as link embedding parameters to indicate whether substrate link $l \in L^s$ is used by the active path $\pi_{u^*v^*}^e$ and the backup path $\rho_{u^*v^*}^{\pi}$ respectively. Hence $\alpha_l(\pi_e) = 1$ if l is used in $\pi_{u^*v^*}^e$, and $\alpha_l(\pi_e) = 0$ otherwise (resp. for $\alpha_l(\rho^{\pi})$) Thus, the objective can be written as:

$$\max \sum_{n \in N} \sum_{e \in E^n} \left[P^e z^n - \sum_{\tau \in \Gamma^e} y_{u*v*}^{\tau} \left(\sum_{l \in \pi} c_b^l b^{c(e)} + \sum_{l^* \in \rho^{\pi}} c_b^l \beta_{\pi}^{l^*} \right) \right] \tag{6.12}$$

subjected to the following constraints:

i For each accepted e, one active-backup pair is selected:

$$\sum_{\tau \in \Gamma^e} y^{\tau}_{u*v*} = z^n; \quad \forall e \in E^n, n \in N, \pi \neq \rho \tag{6.13}$$

ii For each link e, active and backup paths must not exceed the maximum number of hops defined in its QoS class:

$$L(\pi) \leq h^{c(e)}; \qquad \pi \in \Pi^e, e \in E^n, n \in N$$
$$L(\rho^{\pi}) \leq h^{c(e)}; \quad \rho^{\pi} \in P^{\pi}, e \in E^n, n \in N$$

iii For each link e, active-backup pair τ^e is selected based on residual bandwidth defined in its QoS class:

$$\sum_{n \in N} \sum_{e \in E^n} \sum_{\tau \in \Gamma^e} y^{\tau}_{u*v*} \left(\alpha(\pi^e) b^{c(e)} + \alpha(\rho^e) \beta^{l^*}_{\pi} \right) \leq B^r_l(t); \quad \forall l \in L^s \tag{6.14}$$

The solution to the above problem reveals the set of IaaS requests that are most profitable to embed and their best active and backup paths. Link compositions of these active and backup paths are forwarded to the NRM (see the architecture) for deployment by creating flows and reserving bandwidth.

6.4 Performance Evaluation

To evaluate the performance of the protection techniques through experiments, the proposed resource allocation framework was developed as a C++ project which uses Mininet [105] network emulation environment. A network topology was constructed similar to 2007-2008 North American AT&T network connecting 20 major cities [151].Each substrate node has 20 CPU, memory and storage resource units and each substrate link has 20 bandwidth units. As previously mentioned, periodical node and link embedding was followed and the sequence of program modules executed in each period, together

with their inputs and outputs (read/write files), are shown in Fig. 6.5. All three link-embedding algorithms are executed in parallel at each time period for the same set of inputs in order to compare performance of each approach.

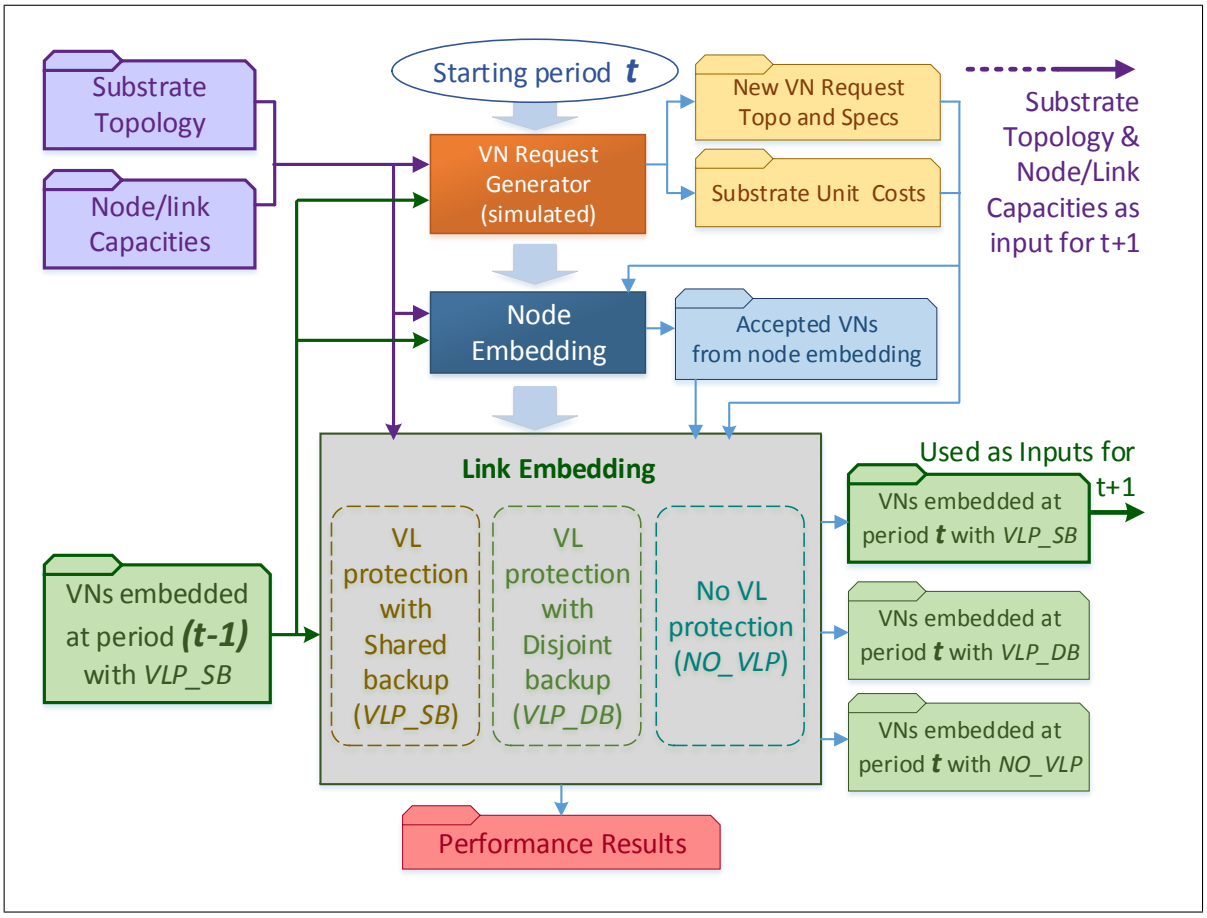


Figure 6.5: Execution flow of the node and link embedding iteration at time period t , with input/output files

To simulate user requests, a traffic generator module inspired from the GT-ITM [170] tool was also developed in C++. IaaS requests were created using a Waxman graph model with 2 to 4 virtual nodes and 1 to 4 VLs. Each randomly requests 5% - 25% of the substrate node and link resources. During normal operating conditions, WAN links are typically provisioned to 30-40% utilization [24]. However, to better evaluate the algorithms, the experiments were conducted in saturated operating conditions. The IaaS request generator was initially triggered to make a high number of requests (about 20 compared to $2-7$ in subsequent periods), bringing the utilization levels of the substrate

network to around 60%. Subsequently, the experiment was executed for 20 consecutive time periods, recording embedding results and substrate resource utilization. This was repeated 20 times and for the analysis, the average values were used.

Components in the IaaS resource allocator that performed virtual node and VL embedding (with and without backups) were developed modules in C++. Thus, node embedding and link embedding functions were performed by separate modules. The network resource manager (NRM), which deploys IaaS embedding on the substrate, was developed in Python as a POX [48] controller application as detailed in chapter 5. ILP problems for both virtual node and VL embedding were formulated and solved using IBM CPLEX Studio 12.3 [154] integrated into GCC C++, in the Ubuntu Linux server edition. A JSON [155] application programming interface (API) over TCP sockets was been implemented between NRM and the IaaS resource allocator to facilitate isolated and independent execution in separate machines. Specifically, the framework modules and components were executed in three machines. The Mininet topology and KVM [65] virtual machines that emulate networked cloud infrastructure used an Intel Xeon dual CPU server with 16 cores and 196 GB RAM. Two desktops with Intel i7 CPUs and 8 GB RAM were used for the IaaS allocator and the NRM.

6.4.1 Evaluation Metrics

The performance of the proposed *VL protection with shared backup (**VLP_SB**)* approach was measured with respect to *VL protection with disjoint backup (**VLP_DB**)* and *VL embedding with no failure protection (**NO_VLP**)* approaches. The table 6.2 shows performance metrics considered for above experimental scenarios.

Table 6.2: Performance Evaluation Metrics

Experiment Scenario	Performance Metrics	Description
Embedding Performance	Acceptance ratio	Number of VL requests accepted over received at link embedding
	Link resource utilization	Total bandwidth allocated (to active and backup) over total bandwidth capacity of used substrate links
	Embedding cost per VL	Average bandwidth cost of active and backup paths per accepted VL
	Bandwidth Redundancy	Ratio of bandwidth used for backup to active paths of embedded VLs
	Normalized hop count	Average number of hops normalized to NO_VLP hops
Embedding efficiency vs Path-computation depth	Acceptance ratio and link resource utilization	Variation of acceptance ratio and bandwidth utilization due to number of active and backup paths calculated from shortest path algorithm
Embedding performance w.r.t Network density	Acceptance ratio and link resource utilization	Performance characteristics of embedding algorithms in terms of acceptance ratio and substrate link resource utilization, due to variations in network densities

6.4.2 Analysis of Experimental Results

Embedding Performance

To evaluate the performance of the proposed techniques, initially the acceptance ratio and the substrate link resource utilization were calculated. All three link embedding strategies were executed in parallel for 20 time periods for the same set of IaaS requests. Also, the substrate state that resulted from VLP_SB embedding in the previous time period was used for the current period as illustrated in the execution flow diagram shown in Fig. 6.5. In most results, the initial time period (t_0) is not shown as it was used to bring the substrate network to a saturated operating state.

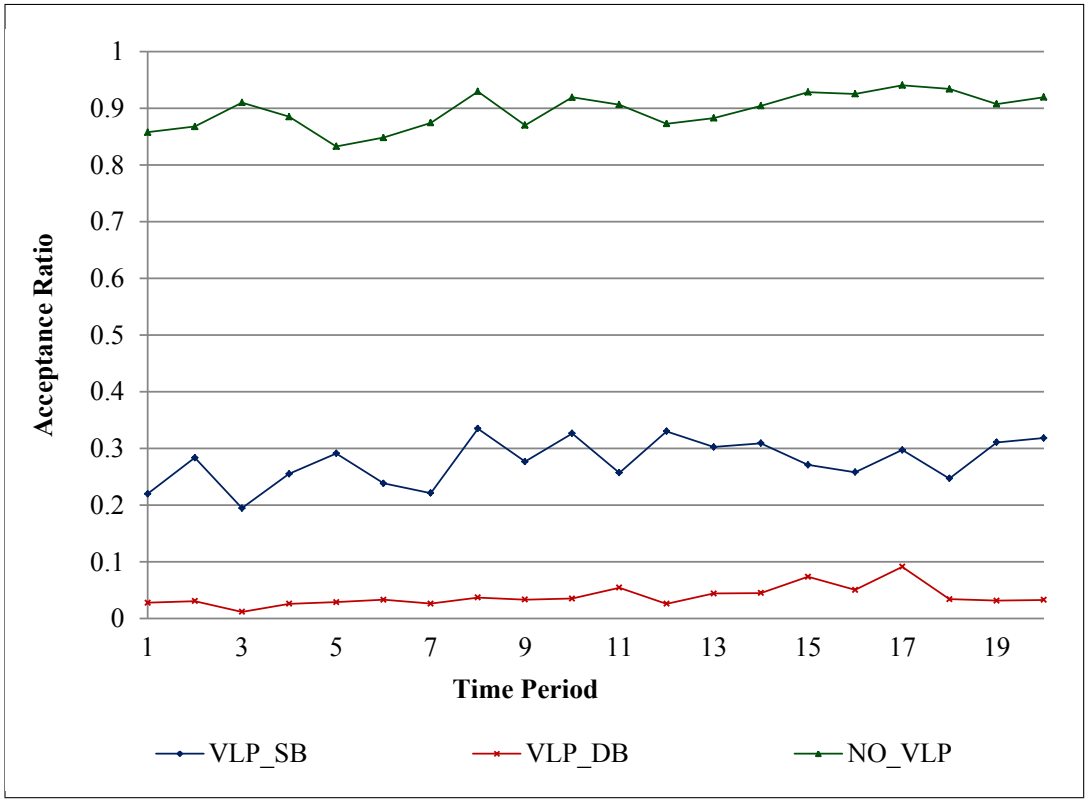


Figure 6.6: Virtual link acceptance ratio for each embedding period when results from shared embedding is used as previous state

Acceptance ratios are shown in Fig. 6.6. They were measured as the number of VL requests accepted over the VL requests received in the link embedding phase (for which

virtual nodes are accepted). It is evident that NO_VLP link embedding gives the highest acceptance (nearly 90%) of the three approaches. These high acceptance values can be attributed to two factors: (i) NO_VLP does not reserve additional backup bandwidth during embedding, (ii) recalculation of allocated bandwidth in the substrate yields high available capacities as it disregards backup bandwidth allocation in the previous embedding state (recall that substrate state prior to link embedding is based on previous VLP_SB results). Out of the backup embedding approaches, VLP_DB gave the lowest average acceptance. Backup bandwidth sharing therefore improves the acceptance, as it consumes less bandwidth than disjoint backups.

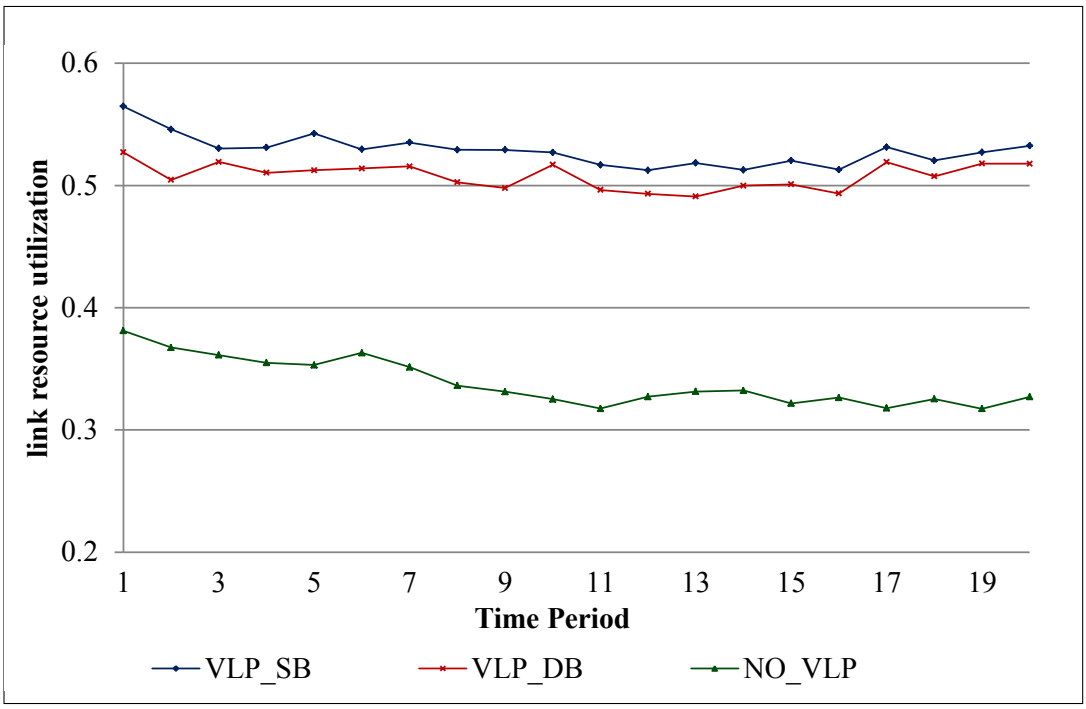


Figure 6.7: Substrate link resource utilization for each embedding period when results from shared embedding is used as previous state

The bandwidth utilization can be used to measure how efficiently the substrate resources are allocated to IaaS requests. For this, link resource utilization was measured as a percentage of bandwidth used for embedding in each substrate link used, as shown in Fig. 6.7. According to the figure, VLP_SB yields a higher level of utilization followed by VLP_DB and NO_VLP. As demonstrated in the acceptance results, lower bandwidth

utilization in NO_VLP can be due to the availability of link capacity at bandwidth recalculation with the previous embedding state.

Although these results show that the proposed approach has better acceptance and resource utilization, they do not provide sufficient information to reach substantial conclusions such as the percentage of link resources used for protection, the cost of embedding costs and the quality of shortest paths in terms of hop count. For that reason, in this work such measurements were taken, using the previous VLP_SB embedding state as a reference, for all three embedding strategies.

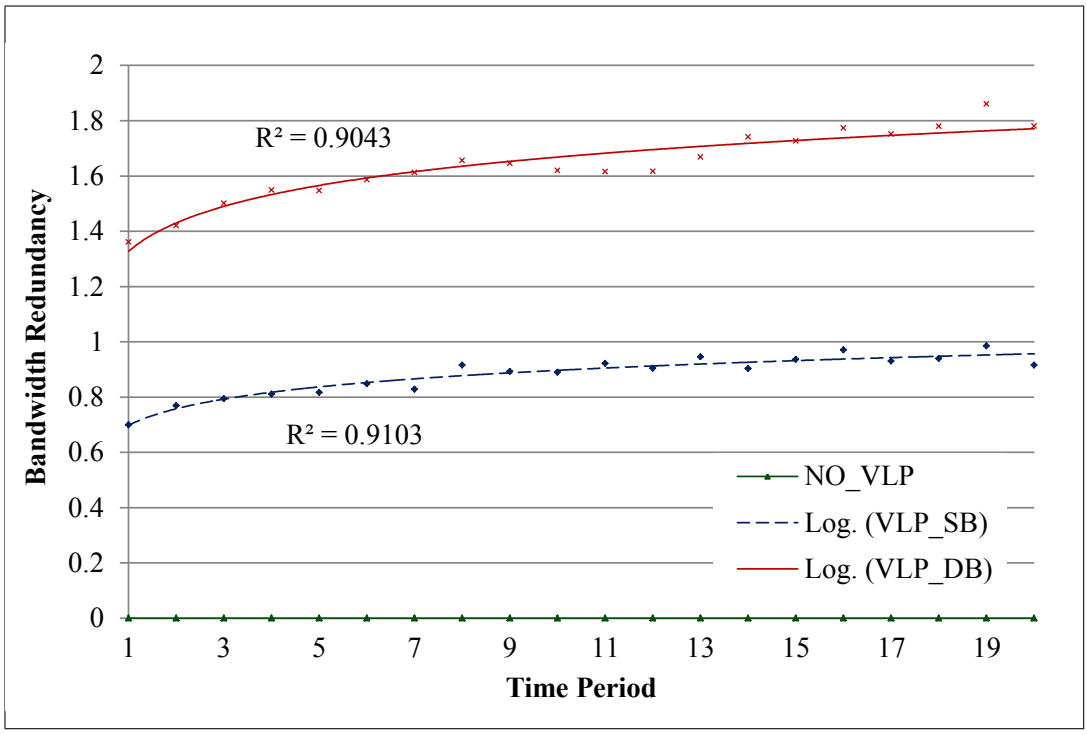


Figure 6.8: Bandwidth Redundancy for each embedding period when results from shared embedding is used as previous state

The ratio of link resources used for backup paths to those used for active paths, also known as bandwidth redundancy, is presented graphically in Fig. 6.8. Since NO_VLP link embedding does not allocate any backup bandwidth, it has zero bandwidth redundancy. For VLP_DB and VLP_SB data points, a logarithmic trend is observed in bandwidth redundancy variations over 20 time periods and trend lines are drawn accordingly. This figure also shows that VLP_DB has higher bandwidth redundancy (1.3 1.9)

than VLP_SB (0.75 1.0).

In VLP_DB, each substrate link in the backup path needs to reserve an exact amount of bandwidth required by the VL. Hence when the backup path is composed of multiple substrate links, higher aggregated bandwidth units are allocated for protection. However, for VLP_SB, the proposed optimization model attempts to reduce embedding cost by maximizing the shared portion of substrate link bandwidth. Thus, a significant reduction in bandwidth redundancy is evident in the VLP_SB approach. In this work the accuracy of the trend lines were verified by calculating the coefficient of determination values (0.9043 for VLP_DB and 0.9103 for VLP_SB). Since logarithmic trends tend to level out to a constant, it can be argued that bandwidth redundancy values may stay constant in subsequent periods. After finding the total number of VLs embedded after each period, it was determined that the logarithmic trend may be caused by the fact that the experiments started from near-saturated embedding states that eventually converged towards more stable/normal operating conditions. Relatively high bandwidth sharing can be anticipated at initial time periods in the shared backup approach, resulting in a higher rate of variation. Since VLP_DB embedding was also performed based on the previous VLP_SB embedding state, a similar variation can be expected.

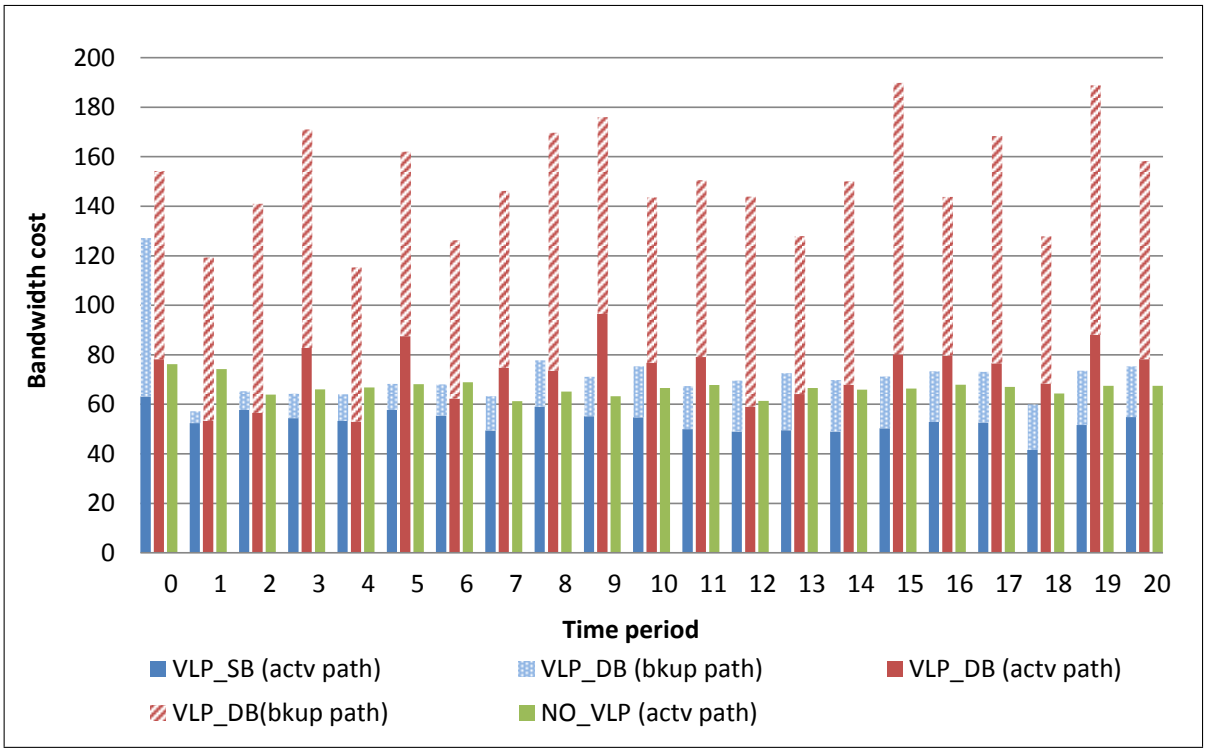


Figure 6.9: Average bandwidth used per virtual link for each embedding period when results from shared embedding is used as previous state

Average bandwidth cost per embedded VLs is presented in Fig. 6.9. To better comprehend the cost variation in the VLP_SB approach during saturated and normal operating conditions, time period t_0 was included in the histogram where the traffic generator made a high number of requests to saturate the infrastructure. From the histogram, it is evident that the cost of active paths for all three approaches is roughly equal. In Table 6.3, per VL backup bandwidth cost for several time periods is shown for further analysis. In the initial time period, no previously reserved backup bandwidth was available in the network. As a result, new backup paths allocated from VLP_SB can only share backup bandwidth within the same time period. Thus, relatively similar backup bandwidth costs (i.e. 63.99 and 75.96) are shown for VLP_SB and VLP_DB in t_0. However, during subsequent time periods, shared backup link embedding shows significantly less cost as it minimizes the bandwidth consumption by maximizing backup sharing with previous embeddings within the same period.

The significant drop in backup cost from $t_0(63.99)$ to $t_1(4.75)$ is mainly due to the

Table 6.3: Backup bandwidth cost per virtual link for different periods

Time Period	t_0	t_1	t_4	t_8	t_12	t_16	t_{20}
VLP_DB	75.96	66.0	62.29	89.5	84.76	64.06	80.1
VLP_SB	63.99	4.75	10.76	18.64	20.58	20.39	20.41

availability of high backup bandwidth suitable for sharing. After t_0, the backup path cost of VLP_SB embedding shows a gradual increase from t_1(4.75) to t_8(18.64) in both Fig. 6.9 and Table 3. However in subsequent time periods ($t_1$2 to $t_2$0), the backup cost largely stayed constant around 20. Bandwidth redundancy can be justified by the system gradually moving to stable/normal operation conditions from the initial saturation.

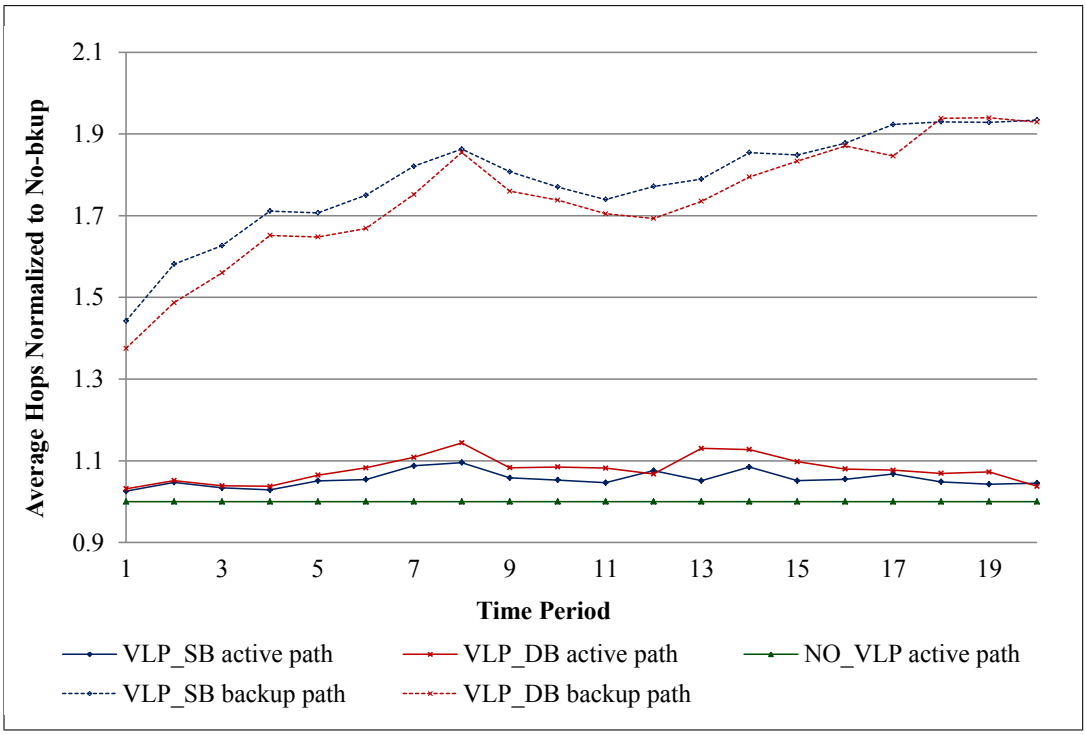


Figure 6.10: Average number of hops normalized to no backup in each embedding period when results from shared embedding is used as previous state

Fig. 6.10 shows the variation of average hop-count in allocated VLs for both active and backup paths. In this graph, it is evident that number of hops for active paths in all three approaches is equivalent. The number of hops in the backup paths for both VLP_SB and VLP_DB is relatively higher than in the active paths since the objective

of the selection algorithm is to reduce backup costs through maximum sharing. Rather than attempting to find the best QoS routes, as in active paths, backup paths attempt to use more shareable bandwidth between source and destination, while satisfying the maximum hop count constraints of each VL. From this graph, it can also be observed that the VLP_SB paths have a similar number of hops to the VLP_DB paths. As well, it can be concluded that VLP_SB gives higher acceptance with backup paths which has an equivalent number of hops.

6.4.3 Embedding Efficiency vs. Path-computation Depth

The optimization algorithm used for VL embedding with VLP_SB protection takes active-backup, shortest path combinations as inputs and solves the proposed ILP model to determine the most profitable substrate paths. This is done by selecting links with the least cost and the most resource sharing. Therefore, it is worthwhile to investigate the effects of the shortest path calculation process on VL acceptance and the efficiency of substrate resource use.

In this work, VL acceptance and bandwidth utilization variation were measured with regard to the number of active and backup paths. First, the number of active path calculations were maintained constant at 5 per VL and the number of backup paths were varied from 1 to 9 per VL. Then the numbers of backup path calculations were kept constant at 5 and varied the number of active path calculations from 1 to 9 per VL. The resulting values were plotted in dual vertical axis graphs, (Fig. 6.11 and Fig. 6.12). Bandwidth utilization and VL acceptance results were plotted in columns and lines respectively.

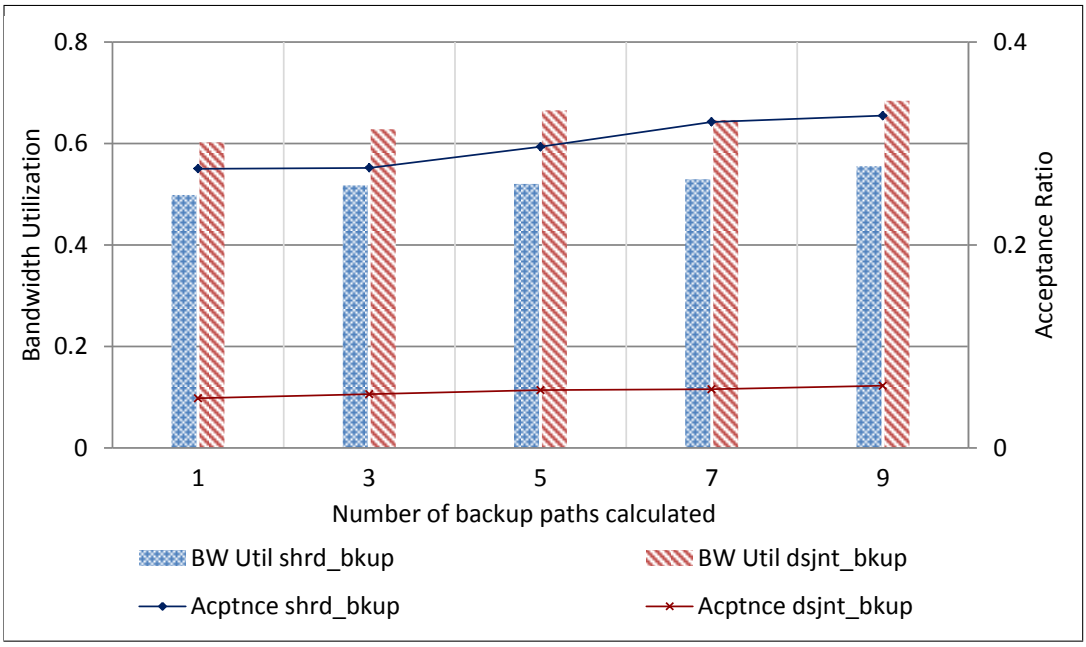


Figure 6.11: Bandwidth utilization and virtual link acceptance variation with respect to number of active path calculations

Fig. 6.11 shows that the increased number of backup path calculations in VLP_SB VL embedding marginally increased both acceptance ratio and bandwidth utilization. Specifically, each additional path calculation increases acceptance by 2.2% and link utilization by 1.4%. A similar variation can be seen for VLP_DB, where each additional path calculation increases acceptance by 2.4% and bandwidth utilization by 1.4%. However, in Fig. 6.12 an increase in the number of active path calculations shows a noticeable increase in acceptance in VLP_SB link embedding. Higher acceptance with lower increment in bandwidth use implies more efficient resource allocation during the embedding phase. It can therefore be concluded that active path calculation has a higher impact on IaaS providers' profits due to the efficiency of substrate resource use.

Table 6.4: Generated substrate network parameters

Edge Probability	Nodes	Links	Density	Total BW	Per link BW
0.15	20	25	0.1316	875	35
0.25	20	44	0.2316	880	20
0.35	20	68	0.3579	884	13
0.45	20	88	0.4632	880	10
0.55	20	110	0.5789	880	8

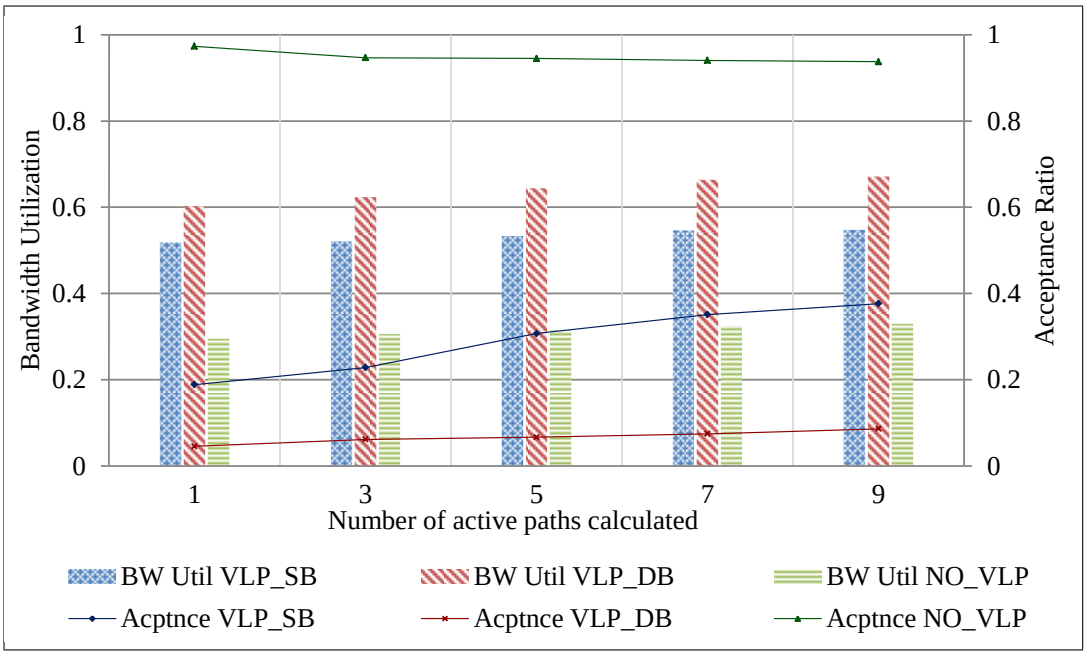


Figure 6.12: Bandwidth utilization and virtual link acceptance variation with respect to number of backup path calculations

6.4.4 Embedding Performance and Network Density

To evaluate the impact of network density on VLP_SB embedding performance, four random network topologies were created using the NetworkX toolkit [171]. Specifically, the "erdos_renyi_graph(n, p)" method was used, which employs Erdos and Renyi models [172] to create network topologies with a pre-specified number of nodes and edge probabilities, (Table6.4).

In all three approaches, total substrate bandwidth was maintained nearly constant around 880 units. This is achieved by reducing per link bandwidth capacity as the number

of substrate links increases. Node and link embedding was performed using each topology for 21 time periods, as previously, with VLP_SB, VLP_DB and NO_VLP algorithms. The results are plotted in Fig. 6.13. In each time period, for all three strategies, the network state of the previous time period is taken from VLP_SB embedding.

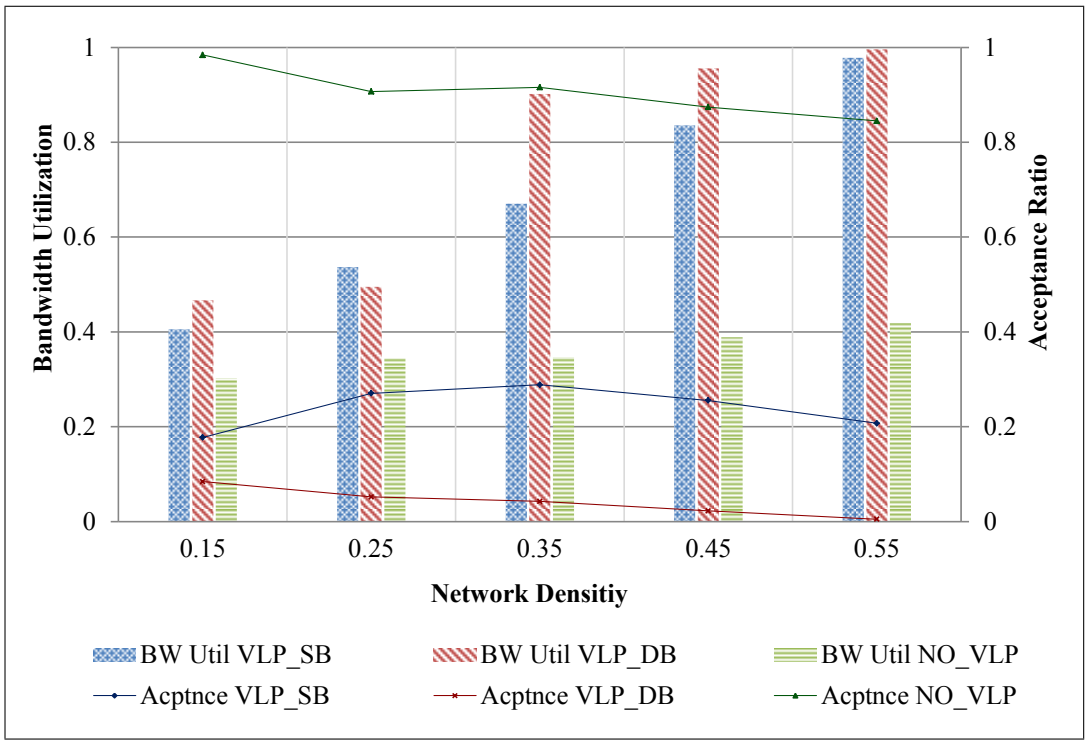


Figure 6.13: Variation of acceptance ratio and bandwidth utilization with respect to network densities

The Figure shows that bandwidth utilization values were increased with network densities in all three approaches. However, VLP_SB and VLP_DB show a significant increase in bandwidth utilization with network densities. Specifically, in high network densities, these two approaches allocated over 96% of substrate resources to VLs. However, the acceptance results show a considerably better performance with VLP_SB than with VLP_DB. Diminishing acceptance values during higher densities can be attributed to the saturation causing a lack of available link capacities.

6.5 Chapter Summary

In this chapter, a shared backup VL protection approach is presented for geo-distributed networked cloud infrastructure. The proposed protection approach is capable of recovering from any single link failures, while maintaining high network resource utilization. The application of the protection mechanism was demonstrated by integrating with cloud resource management. A number of experiments were conducted to evaluate the protection and restoration mechanisms. The results demonstrate that the proposed approach outperforms common protection mechanisms and can increase IaaS request acceptance and substrate resource utilization. Furthermore, the proposed reactive path restoration can successfully restore nearly 60% of failed links with better switch flow table use, compared to 32.6% restoration in the proactive path restoration.

7 Conclusion and Future Work

In this chapter, a conclusion to the dissertation was provided by summarizing conducted research work, contributions and outcomes. In addition, identified limitations and promising future research directions have been described.

7.1 Conducted Research Work

The objective of the conducted research was to identify the limitations in the area of cloud network resource management, and to propose innovative solutions by applying state-of-the-art tools and technologies. The initial step towards achieving this objective was conduct a literature study. Through the study, the following aspects were identified: (i) limitations in network resource management in centralized, mobile and geo-distributed data center clouds, (ii) constraints in traditional network control and virtualization technologies in addressing those limitations, and (iii) capabilities of virtual network embedding and control technologies such as SDN to overcome such limitations. Those limitations include, lack of virtualization support, poor resource utilization, limited traffic control features in client virtual networks, lack of seamless service migration capabilities and unavailability of efficient fault tolerant techniques. To address these limitations, three cloud network resource management frameworks were proposed which lead to the following contributions:

- *An SDN-based cloud datacenter resource management framework [90]:* A datacenter resource management architecture was designed to facilitate automated

and on-demand provisioning of compute, storage and network infrastructure. An Ontology-based resource composition approach that exploits semantic similarity and closeness centrality characteristics was adopted to perform VN embedding with less computational overhead and time. The framework was designed to provide IaaS with a high degree of control over network connectivity, and QoS to application service providers. This was archived by integrating control-plane virtualization functions of SDN into the cloud resource management framework. The proposed cloud resource management framework was implemented on a real-physical test-bed as well as on an emulated test-bed constructed using the Mininet toolkit.

- *QoS aware mobile cloud resource management framework [25]:* A mobile cloud resource management framework that allocates cloud resources from RDCs to mobile applications was presented. Using cloud services acquired from RDCs, the framework serves mobile end users by providing QoS and seamless service migration services. In the event of user mobility, the framework monitor QoS, and reactively relocates cloud resources to an optimally selected RDC while maintaining end-to-end transparency. The framework makes use of data-plane event handling and control plane programmability functionalities offered by SDN to detect user movements between regions and to dynamically configure switches at the edge of the wireless and RDC networks. In order to evaluate the performance of the framework with respect to existing scenarios, a prototype on the Mininet simulation platform and a physical test-bed was developed. Through performance evaluation, handling of cloud service migrations seamlessly to the user with minimal overhead was demonstrated.

- *QoS-aware fault tolerant cloud resource management framework with SDN-based dynamic restoration [176]:* A geo-distributed multi-datacenter cloud resource management framework that employs an SDN-based fault tolerant scheme, to dynamically detect and restore failures in WAN links was introduced. The proposed

framework allocate infrastructure resources to satisfy application QoS requirements while maximizing request acceptance and resource utilization. Key contributions of this work include; (i) formulation of the virtual network resource allocation problem as a periodical ILP problem and optimization of the allocation of compute and network resources on geo-distributed cloud infrastructure, (ii) design and implementation of two OpenFlow-based QoS-aware reactive network failure restoration algorithms that dynamically reroute traffic upon identifying link failures and, (iii) implementation of the framework and evaluation of the performance of proposed algorithms to identify their characteristics during multiple substrate link failures.

- *QoS guaranteed protection from any single link failures through shared backup embedding [177]:* First, VN protection techniques employed at the resource-allocation level was studied. Then the ability of shared backup protection to improve substrate resource utilization and IaaS acceptance was evaluated by introducing a shared backup link embedding technique. In the proposed technique, substrate resources were allocated with the objective of satisfying application QoS requirements while increasing the IaaS providers' revenue by maximizing request acceptance and resource utilization. The IaaS resource allocation problem was formulated as a periodical ILP problem that optimizes the allocation of compute and network resources on a geo-distributed NCI. The proposed technique was integrated into the IaaS allocation framework presented in Chapter 5 which guaranteeing fast and resource efficient recovery from any single substrate link failure. Through performance evaluation, it was shown that the introduced shared backup link embedding technique can increase the overall substrate resource utilization by minimizing overall bandwidth reservation as backup bandwidth from the substrate network.

7.2 Limitations and Future Research Work

In this thesis, extensive effort has been made with research ideas and a myriad of implementation methodologies have been adopted to verify and evaluate these ideas. Despite such efforts, few limitations have been identified. In this section, such limitations in each area of research are summarized along with potential solutions and improvements that can be conducted in future research studies.

7.2.1 Limitations

An SDN-based cloud datacenter resource management framework

The cloud datacenter resource management framework that employs SDN based virtualization to deliver a higher degree of control over virtual networks has the ability to improve IaaS by providing more customized virtual networks. Such improvements benefit from enhanced resource description and modeling as well as powerful virtualization capabilities offered by SDN. However, several aspects that limits the proposed framework's ability to achieve its design goals have been recognized.

In this work, a resource composition approach was employed to construct VNs and serve cloud user IaaS requests. Several limitations were identified in the proposed framework. First, it is understood that the VN composition and resource allocation algorithm may lack the ability to perform optimum allocations that maximize resource utilization and service provider profits. Although the composition has the potential to surpass resource allocation techniques that employ integer linear programming and other optimization solvers, in terms of speed and computing overhead, it cannot guarantee profit maximization. Hence in subsequent work of this dissertation, optimization techniques have been adopted to achieve better resource allocations in the expense of more computing overhead and delay.

Second, the use of INDL extended from OWL ontology introduces several limitations

to the proposed framework. In [112,113], the authors highlight a number of known limitations in ontology such as: very little support for modelling temporal aspects, inadequate operators to define complex context descriptions, lack of expressive class constructors and lack of scalability. Additionally, ontologies are renown for serious performance issues in reasoning. Hence, in this work, ontologies were used for data description and modeling, without relying on reasoning capabilities to compose virtual networks. However, these limitations calls for more scalable and cost efficient tools to construct PR and VR pools in future studies.

Finally, SDN tools used for virtualization and flow control bring several limitations to the proposed framework. FlowVisor has been used to achieve virtualization properties such as isolation and resource abstraction to construct VNs for each tenant. As mentioned in [114,115], several disadvantages in FlowVisor based network virtualization include the following:

- Only supports OpenFlow specification version 1.0 with limited functionalities (v1.6 is available at the time of this writing)
- Topology bounded virtualization does not allow the creation of any arbitrary virtual topology. Only a subset of the physical topology can be constructed as a virtual topology.
- Use of packet header space to separate slices reduce accessible header fields. This further affects nested virtulization with multiple flowvisors.
- The centralized nature of the FlowVisor significantly limits the scalability.

Because the operation of the FlowVisor is central to the proposed framework in creating VNs, it is also limited by the aforementioned issues. However, due to the modular architecture of the framework, SDN based network hypervisors [114] designed to overcome one or more of these limitations can be integrated.

QoS aware mobile cloud resource management framework

Seamlessly migrating cloud services between datacenters (while maintaining strict QoS requirements), is the key design objective of the mobile cloud framework. Although the proposed QoS aware mobile cloud service management framework successfully demonstrated the ability to address this requirement, several limitations have been recognized.

One of the main limitations is in optimality and efficiency of the datacenter selection. In the given solution, if a QoS violation is identified, the proposed approach attempts to find a destination datacenter to migrate the cloud service by calculating migration cost and datacenter resource costs. Since this is performed individually, per user movement basis, two disadvantages are noted. First, considerable overhead was added to the centralized SDN controller to calculate and configure paths dynamically, maintain IP to pseudo IP mapping and address ARP/RARP messaging to maintain seamelessness. Second, current datacenter selection mechanism provides a greedy solution by minimizing cost of individual service relocation, rather than minimizing overall cost. Hence, its useful to formulate an ILP problem by considering service migration requests in batches, collected for a pre-determined interval.

The proposed solution also suffers from data-plane consistency related issues. When flow rules are configured dynamically to perform service migration tasks, they need to be installed in all forwarding elements in each path at the same time. If delays occur when installing rules and different OpenFlow switches in the path take different times, it can result in multiple transitional states causing unpredictable behaviour in the network, such as routing loops and buffer overloading. This is particularly critical as it can cause severe packet loss, congestion and delays. However, in this work a simple and effective solution to this issue is adopted in chapter 5, through flow versioning. Flow versioning guarantees per-packet consistency during flow install, delete and modify operations. The per-packet versioning solution can only assure that each packet will follow either old or new rule, not a random combination of both. Although this solution can guarantee data-plane consistency, it will make maintaining seamless nature during service migration

complex and challenging.

Fault tolerant cloud resource management framework

Providing fault tolerance through dynamic restoration and shared backup protection allows cloud service providers to improve profits by increasing substrate resource utilization and request acceptance. Several limitations in proposed fault tolerance schemes were identified that limit their applicability into specific IaaS request and traffic types. First, the proposed SDN-TE based dynamic restoration sufferd from noticeable MTTR. This happened because whenever a failure occurred in a substrate network link, the periodical link-state up messages needed to detect the failure and inform the centralized controller. At this time the controller may start calculating new routes in the case of reactive restoration and may use the pre-calculated paths in the case of proactive restoration. Afterwords, new flow rules will be derived and sent to each forwarding element in the new path. This process adds considerable number of delays to the critical paths and in some cases may take tens of seconds. In addition if link failure causes congestion or adds additional distances for signaling packets to reach the switch, more delays can be expected. Therefore, dynamic restoration based fault tolerance is usually suitable for low priority best-effort traffic such as database sync, nightly backup and VM migration. If the majority of the traffic in the inter-datacenter network falls into these categories, service provider can be benefited highly by using SDN-TE based dynamic restoration schemes.

7.2.2 Future Work

In addition to overcoming the above limitations, this work leads to a number of interesting future research directions. For the datacenter resource management framework, it would be useful to investigate different resource allocation techniques. Furthermore, recent findings in data representation and storage technologies (such as NoSQL databases) could be a potential solution to overcoming scalability limitations of Ontology. For seam-

less mobile cloud service migration framework, it is important to conduct experiments with multiple wireless node movement between regions to evaluate the scalability of the proposed solution. This will also provide an opportunity to identify better scenarios for batch processing to serve migration requests. In addition, it is interesting to study the construction of a test-bed with wireless equipment to experiment under more realistic scenarios. For the proposed SDN-TE based dynamic path restoration approach, it is worth conducting an investigation to identify causes for fault restoration delays and ways to eliminate/compensate them. Also effectiveness of restoration schemes with different cloud application types (e.g. delay sensitive vs throughput sensitive) need to be studied. For the path protection-based fault tolerance approach, it is interesting to compare the performance with other more prominent shared backup protection approaches available in the literature. In addition, further investigation of protection and restoration approaches using class-based traffic characterization are important aspects to be explored.

7.3 Concluding Remarks

Resource management in cloud computing is a complex task involving many challenges. In this dissertation we have focused on three different areas of cloud network resource management, namely: resource management within datacenters, cloud service management with multiple datacenters, and providing fault tolerant cloud services with geo-distributed datacenter clouds. Limitations of existing research works have been identified along with and challenges associated with each of these areas. With that knowledge, three cloud resource management frameworks have been proposed that present novel solutions to number of issues. Proposed solutions have been verified through simulations, test-bed implementations and performance evaluations. Finally, several limitations in proposed cloud network resource management frameworks have been highlighted that could lead to several future research directions.

Bibliography

[1] P. Burris, "Wikibon report preview: How big can Amazon web services get", Available [Online] https://siliconangle.com/blog/2017/02/20/wikibon-report-preview-big-can-amazon-web-services-get/.

[2] S. Nag, F. Ng, H. H. Swinehart, and F. Biscotti, "Gartner forecast analysis: Public cloud services, worldwide, 2Q17 Update" Sept. 2017.

[3] M. P. Papazoglou, and W. Heuvel, "Blueprinting the cloud," IEEE Internet Computing, vol. 15, no. 6, pp. 74-79, 2011.

[4] T. A. Genez, L. F. Bittencourt, and E. Madeira, "Workflow scheduling for SaaS/PaaS cloud providers considering two SLA levels," In Proc. Network Operations and Management Symposium (NOMS), 2012 IEEE, pp. 906-912. IEEE, 2012.

[5] M. Liaqat, V. Chang, A. Gani, S. H. Hamid, M. Toseef, U. Shoaib, and R. L. Ali, "Federated cloud resource management: Review and discussion," Journal of Network and Computer Applications, vol. 77, pp. 87-105, 2017.

[6] H. N. Van, F. D. Tran, and J. M. Menaud, "SLA-aware virtual resource management for cloud infrastructures," In Proc. Ninth IEEE International Conference on Computer and Information Technology (CIT'09), vol. 1, pp. 357-362, 2009.

[7] B. Jennings, and R. Stadler, "Resource management in clouds: Survey and research challenges," Journal of Network and Systems Management, vol. 23, no. 3, pp. 567-619, 2015.

[8] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: Challenges and opportunities for innovations," IEEE Communications Magazine, vol. 51, no. 7, pp. 54-62, 2013.

[9] S. Mustafa, B. Nazir, A. Hayat, and S. A. Madani, "Resource management in cloud computing: Taxonomy, prospects, and challenges," Computers & Electrical Engineering, vol. 47, pp. 186-203, 2015.

[10] A. Gulati, G. Shanmuganathan, A. M. Holler, and I. Ahmad, "Cloud scale resource management: Challenges and techniques," In Proc. HotCloud pp. 3-3, 2011.

[11] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," In Proc. ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50-55, 2008.

[12] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev et al, "Network virtualization in multi-tenant datacenters," In Proc. USENIX, NSDI, vol. 14, pp. 203-216, 2014.

[13] D. Kreutz, F. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015.

[14] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks," In Proc. SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, Mar. 2008.

[15] I. Houidi, W. Louati, D. Zeghlache, and S. Baucke, "Virtual resource description and clustering for virtual network discovery," In Proc. IEEE International Conference on Communications Workshops (ICC Workshops 2009), pp. 1-6. 2009.

[16] P. T. Endo, A. V. A. Palhares, N. N. Pereira, G. E. Goncalves, D. Sadok, J. Kelner, B. Melander, and J. Mangs, "Resource allocation for distributed cloud: concepts and research challenges," IEEE network, vol. 25, no. 4, 2011.

[17] R. Choubey, R. Dubey, and J. Bhattacharjee, "A survey on cloud computing security, challenges and threats," International Journal on Computer Science and Engineering (IJCSE), vol. 3, no. 3, pp. 1227-1231, 2011.

[18] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," In Proc. 10th IEEE International Conference on High Performance Computing and Communications (HPCC'08), pp. 5-13. 2008.

[19] H. Nuseibeh, "Adoption of Cloud Computing in Organizations," In Proc. Americas Conference on Information Systems (AMCIS 2011), vol. 372, 2011.

[20] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," IEEE Transactions on Network and Service Management, vol. 9, no. 4, pp. 373-392, 2012.

[21] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," ACM SIGCOMM Comput. Commun. Rev., vol. 39, pp. 68-73, Jan. 2009.

[22] A. Ahmed, and E. Ahmed, "A Survey on Mobile Edge Computing,", In Proc. 10th IEEE International Conference of Intelligent Systems and Control (ISCO 2016), 2016.

[23] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi, "Improving convergence speed and scalability in OSPF: A survey." IEEE Communications Surveys & Tutorials, vol. 14, no. 2, pp. 443-463, 2012.

[24] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata et al., "B4: Experience with a globally-deployed software defined WAN," In ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 3-14, 2013.

[25] W. Ekanayake, H. Amarasinghe, and A. Karmouch, "SDN-based IaaS for mobile computing," In Proc. 14th IEEE Annual Consumer Communications & Networking Conference (CCNC 2017), pp. 179-184, 2017.

[26] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," NIST Special Publication, vol. 800, no. 145, p. 7, 2011.

[27] W. Tsai, X. Sun, and J. Balasooriya, "Service-oriented cloud computing architecture," In Proc. Seventh International Conference on Information Technology: New Generations (ITNG 2010) , pp. 684-689, 2010.

[28] F. Liu et.al., "NIST Cloud computing reference architecture(draft)," NIST Special Publication, vol. 500, no. 292, p. 35, 2011.

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

[30] I. Drago, M. Mellia, M. M. Munafo, A. Sperotto, R. Sadre, and A. Pras, "Inside dropbox: understanding personal cloud storage services," In Proc. of the 2012 ACM conference on Internet measurement, pp. 481-494, 2012.

[31] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu, "Cloud storage as the infrastructure of cloud computing," In Proc. International Conference on Intelligent Computing and Cognitive Informatics (ICICCI 2010), pp. 380-383, 2010.

[32] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," ACM SIGOPS operating systems review, vol. 37, no. 5, pp. 164-177, 2003.

[33] B. Kocoloski, and J. Lange, "Better than native: using virtualization to improve compute node performance," In Proc. of the 2nd ACM International Workshop on Runtime and Operating Systems for Supercomputers, p. 8, 2012.

[34] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," ACM SIGCOMM Computer Communication Review, vol. 33, no. 1, pp. 59-64, 2003.

[35] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," Computer, vol. 38, no. 4, pp. 34-41, 2005.

[36] M. Yu, Y. Yi, J. Rexford, and M. Chiang, “Rethinking virtual network embedding: substrate support for path splitting and migration,” SIGCOMM Computer Commun. Rev., vol. 38, no. 2, pp. 17–29, 2008.

[37] T. Ghazar, and N. Samaan. "Pricing utility-based virtual networks," IEEE Transactions on Network and Service Management, vol. 10, no. 2, pp. 119-132, 2013.

[38] H. Amarasinghe, "Utilization of Dynamic Attributes in Resource Discovery for Network Virtualization," University of Ottawa, ON, Canada, 2012.

[39] H. Amarasinghe, A. Belbekkouche, and A. Karmouch, "Aggregation-based discovery for virtual network environments," In Proc. IEEE International Conference on Communications (ICC 2012), pp. 1276-1280, 2012.

[40] M. M. Hasan, H. Amarasinghe, and A. Karmouch, "Network virtualization: Dealing with multiple infrastructure providers," In Proc. IEEE International Conference on Communications (ICC 2012), pp. 5890-5895,2012.

[41] A. Belbekkouche, M. M. Hasan, and A. Karmouch, "Resource discovery and allocation in network virtualization," IEEE Communications Surveys & Tutorials, vol. 14, no. 4, pp. 1114-1128, 2012.

[42] H. Kim, and N. Feamster, "Improving network management with software defined networking," IEEE Communications Magazine, vol. 51, no. 2, pp. 114-119, 2013.

[43] S. Schenker, "The future of networking, the past of protocols," Oct. 2011. [Online]. Available: http://www.youtube.com/watch?v=YHeyuD89n1Y.

[44] R. Jain, and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," IEEE Communications Magazine, vol. 51, no. 11, pp. 24-31, 2013.

[45] Cisco One Platform Kit [Online], Available: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/bulletin-c25-736612.pdf. [Accessed Dec. 16, 2018].

[46] OpenFlow Switch Specification, "Version 1.0. 0 (Wire Protocol 0x01)." Open Networking Foundation 2009.

[47] 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.

[48] "POX Homepage," [Online]. Available: http://www.noxrepo.org/pox/about-pox/.

[49] "Floodlight Homepage," [Online]. Available: www.projectfloodlight.org/floodlight/.

[50] "ONOS Homepage," [Online]. Available: https://onosproject.org/.

[51] "OpenDaylight Homepage," [Online]. Available: https://www.opendaylight.org/.

[52] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 655-685, 2016.

[53] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?," In OSDI, vol. 10, pp. 1-6. 2010.

[54] E. Salvadori, R. D. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in OpenFlow-based networks," in Proc. IEEE GLOBECOM, pp. 1–6, Dec. 2011.

[55] X. Jin, J. Gossels, J. Rexford, and D. Walker, "CoVisor: A compositional hypervisor for software-defined networks," in Proc. USENIX Symp. NSDI, pp. 87–101, 2015.

[56] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," OpenFlow Switch Consortium, Tech. Rep 1, no. 132, 2009.

[57] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T. Huang et al., "Carving research slices out of your production networks with OpenFlow," ACM SIGCOMM Computer Communication Review, vol. 40, no. 1, pp. 129-130, 2010.

[58] Amazon, "Amazon Web Services," [Online]. Available: http://aws.amazon.com/

[59] Google, "Google app Engine," [Online]. Available: http://code.google.com/appengine/

[60] Microsoft, "Windows Azure," [Online]. Available: http://www.microsoft.com/windowsazure/windowsazure/

[61] "OpenStack Homepage," [Online]. Available: https://www.openstack.org/.

[62] "Nimbus project Homepage," [Online]. Available: http://www.nimbusproject.org/.

[63] "Eucalyptus project Homepage," [Online]. Available: https://docs.eucalyptus.com/eucalyptus/latest/.

[64] T. Rosado, and J. Bernardino, "An overview of openstack architecture," In Proc. of the 18th International Database Engineering & Applications Symposium, pp. 366-367, 2014.

[65] KVM-contributors, "Kernel-based Virtual Machine," KVM, 7 November 2016. [Online]. Available: https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792.[Accessed 23 October 2017].

[66] "XRN Homepage," [Online]. Available: https://www.xenproject.org/.

[67] "vmware Homepage," [Online]. Available: https://www.vmware.com/.

[68] Libvirt virtualization API, "Libvirt Homepage," [Online]. Available: https://libvirt.org/.

[69] M. Satyanarayanan, "Fundamental challenges in mobile computing," In Proc. of the fifteenth annual ACM symposium on Principles of distributed computing, pp. 1-7, 1996.

[70] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," Future Generation Computer Systems, vol. 29, no. 1, pp. 84-106, 2013.

[71] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," Wireless communications and mobile computing, vol. 13, no. 18, pp. 1587-1611, 2013.

[72] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smart-phones last longer with code offload," In Proc. ACM MobiSys, 2010.

[73] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic Execution Between Mobile Device and Cloud," In Proc. ACM EuroSys, 2011.

[74] V. Balasubramanian, and Ahmed Karmouch, "An infrastructure as a Service for mobile Ad-hoc Cloud," In Proc. IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC 2017) , pp. 1-7. IEEE, 2017.

[75] G. H. Canepa, and D. Lee, "A virtual cloud computing provider for mobile devices," In Proc. 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond, p. 6, 2010.

[76] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," In Proc. Fourth IEEE International Conference on Utility and Cloud Computing (UCC 2011), pp. 281-286, 2011.

[77] K. Ha, P. Pillai, G. Lewis, S. Simanta, S. Clinch, N. Davies, and M. Satyanarayanan, "The impact of mobile multimedia applications on data center consolidation," In Proc. IEEE International Conference on Cloud Engineering (IC2E 2013), pp. 166-176, 2013.

[78] G. A. McGilvary, A. Barker, M. Atkinson, "Ad hoc cloud computing: From concept to realization", IEEE Cloud 2015.

[79] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," IEEE pervasive Computing, vol. 8, no. 4, 2009.

[80] M. Satyanarayanan, Z. Chen, K. Ha, W. Hu, W. Richter, and P. Pillai, "Cloudlets: at the leading edge of mobile-cloud convergence," In Proc. 6th International Conference on Mobile Computing, Applications and Services (MobiCASE 2014), pp. 1-9, 2014.

[81] A. Ceselli, M. Premoli, and S. Secci, "Cloudlet network design optimization," In Proc. IFIP Networking Conference (IFIP Networking 2015), pp. 1-9, 2015.

[82] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," In Proc. IEEE 16th International Enterprise Distributed Object Computing Conference (EDOC 2012), pp. 123-132, 2012.

[83] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," In Proc. Sixth International Conference on Advances in Future Internet, 2014.

[84] Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. "Fog computing and its role in the internet of things." In Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pp. 13-16. ACM, 2012.

[85] Vaquero, Luis M., and Luis Rodero-Merino. "Finding your way in the fog: Towards a comprehensive definition of fog computing." ACM SIGCOMM Computer Communication Review 44, no. 5 (2014): 27-32.

[86] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," In Proc. 2015 Workshop on Mobile Big Data. ACM, 2015.

[87] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," IEEE Communications Surveys and Tutorials, vol. 19, no. 4, pp. 2322-2358, 2017.

[88] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—A key technology towards 5G," ETSI White Paper, vol. 11, no. 11, pp. 1-16, 2015.

[89] "Mobile-Edge Computing," Introductory Technical White Paper, September, 2014.

[90] H. Amarasinghe and A. Karmouch, "SDN-based framework for Infrastructure-as-a-Service Clouds," In Proc. 9th IEEE International Conference on Cloud Computing (CLOUD), 2016. .

[91] N. Martinelli, "Connect the dots on OpenStack Neutron," [Online]. Available: http://superuser.openstack.org/articles/connect-the-dots-on-openstack-neutron-with-this-updated-manual.

[92] P. Gandhi, "Achieving OpenStack networking nirvana with SDN: Scalability, resiliency, automation," Big switch Networks, [Online]. Available: http://www.bigswitch.com/blog/2015/10/28/achieving-openstack-networking-nirvana-with-sdn-scalability-resiliency-automation.

[93] O. Tkachova, M. J. Salim, and A. R. Yahya, "An analysis of SDN-OpenStack integration," In Proc. 2015 Second International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T), pp. 60-62. IEEE, 2015.

[94] P. Singh, and S. Manickam, "Design and deployment of OpenStack-SDN based test-bed for EDoS," In Proc. 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), pp. 1-5. IEEE, 2015.

[95] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz et al. "ONOS: towards an open, distributed SDN OS." In Proc. 3rd workshop on Hot topics in software defined networking, pp. 1-6. ACM, 2014.

[96] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an SDN platform for cloud network services." IEEE Communications Magazine, vol. 51, no. 2, pp. 120-127, 2013.

[97] D. Theodorou et al., "NRS: A System for automated network virtualization in IaaS cloud infrastructures," In. Proc. 3rd international workshop on Virtualization technologies in distributed computing, (VTDC), June 2013.

[98] J. A. Wickboldt et al., "Retinking cloud platforms: Network-aware flexible resource allocation in IaaSClouds," In Proc. 13th IFIP/IEEE International symposium of integrated network management (IM 2013), pp. 450-456, Ghent, Belgium, 2013.

[99] T. Benson et al., "CloudNaaS: a cloud networking platform for enterprise applications," In Proc. ACM Symposium on Cloud Computing (SoCC), 2011.

[100] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," In Proc. 6th International Conference on emerging Networking Experiments and Technologies (CoNEXT), 2010.

[101] K. M. Metwally et al., "Two-phase ontology-based resource allocation approach for IaaS cloud service," In Proc. 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), 2015.

[102] K. M. Sim, "Agent-Based Cloud Computing," IEEE Transactions on Services Computing, vol. 5, no. 4, pp. 564–577, 2012.

[103] Z. Wang, Y. Han, T. Lin, Y. Xu, S. Ci, and H. Tang, "Topology-aware virtual network embedding based on closeness centrality." Frontiers of Computer Science, vol. 7, no. 3, pp. 446–457, 2013.

[104] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in Proc. ACM SIGCOMM 2011 Conference, ser. (SIGCOMM '11), New York, NY, USA, pp. 242–253, 2011.

[105] Mininet, "An Instant Virtual Network on your Laptop (or other PC)," April 2016 [Online]. Available: http://mininet.org

[106] Open vSwitch, "OVS: Production Quality, Multilayer Open Virtual Switch - Version 2.3.0," August 2015 [Online]. Available: http://openvswitch.org/.

[107] M. Ghijsen et al., “Towards an Infrastructure Description Language for Modeling Computing Infrastructures," in The 10th IEEE International Symposium on Parallel and Distributed Processing, 2012

[108] Apache Jena Ontology model specifications, [Online]. Available: https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/ontology/OntModelSpec.html.

[109] VirtualBox, "VirtualBox: general-purpose full virtualizer for x86 hardware - Version 4.3.22," January 2016 [Online]. Available: https://www.virtualbox.org/.

[110] IPTraf, "IPTraf: IP Network Monitoring Software - Version 3.0.0," September 2015 [Online]. Available: http://iptraf.seul.org/

[111] Zabbix, "Zabbix: The Enterprise-class Monitoring Solution - Version2.2 LTS," January 2016, [Online]. Available: http://www.zabbix.com/

[112] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," Pervasive and Mobile Computing, col. 6, no. 2, pp. 161-180, 2010.

[113] B. Motik, and I. Horrocks, "Problems with OWL syntax." In OWLED, vol. 216. 2006.

[114] E. Salvadori, R. D. Corin, M. Gerola, A. Broglio, and F. De Pellegrini, "Demonstrating generalized virtual topologies in an openflow network," In Proc. ACM SIGCOMM Computer communication review, vol. 41, no. 4, pp. 458-459, 2011.

[115] S. T. Sany, S. Shashidhar, M. P. Gilesh, and S. D. M. Kumar. "Performance evaluation and assessment of FlowVisor," In Proc. IEEE International Conference on Information Science (ICIS), pp. 222-227, 2016.

[116] Internet Trends 2015 – Code Conference by Mary Meeker, kpcb.com/InternetTrends

[117] D. Huang, T. Xing, and H. Wu. "Mobile cloud computing service models: a user-centric approach," IEEE Network, 2013.

[118] Q. Ning, C. Chen, R. Stoleru, and C. Chen, "Mobile storm: Distributed real-time stream processing for mobile clouds," In Proc. IEEE International Conference on Cloud Networking (CloudNet), pp. 139-145, 2015.

[119] U. Shaukat, E. Ahmed, Z. Anwara, F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," Journal of Network and Computer Applications (JNCA), vol. 62, pp. 18-40, 2016

[120] A. Jarray et al., "Column Generation based-Approach for IaaS Aware Networked Edge Data-Centers." Globecom IEEE, 2014.

[121] Y. Li, H. Wang, M. Liu, B. Zhang, and H. Mao, "Software defined networking for distributed mobility management," IEEE Globecom Workshops (GC Wkshps), 2013.

[122] H. Assasa, S. V. Yadhav, L. Westberg, "Service Mobility in Mobile Networks," In Proc. IEEE 8th International Conference on Cloud Computing (ICC), 2015.

[123] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," In Proc. IEEE 35th Annual International Conference on Computer Communications (INFOCOM), pp. 1-9, 2016.

[124] R. Bifulco, and R. Canonico, "Analysis of the handover procedure in Follow-Me Cloud," In Proc. IEEE 1st International Conference on Cloud Networking (CLOUDNET), pp. 185-187, 2012.

[125] Y. Wang, and J. Bi, "A solution for IP mobility support in software defined networks," In Proc. IEEE 23rd International Conference on Computer Communication and Networks (ICCCN), pp. 1-8, 2014.

[126] K. Tantayakul, R. Dhaou, and B. Paillassa, "Impact of SDN on Mobility Management," In Proc. IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), pp. 260-265, 2016.

[127] K. M. Metwally, A. Jarray, and A. Karmouch, "Two-phase ontology-based resource allocation approach for IaaS cloud service," In Proc. 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), pp. 790-795, 2015.

[128] J. Liu, Y. Li, and D. Jin, "SDN-based live VM migration across datacenters," ACM SIGCOMM Computer Communication Review, vol. 44, no. 4, pp. 583-584, 2014.

[129] V. Mann, A. Vishnoi, K. Kannan, and S. Kalyanaraman, "CrossRoads: Seamless VM mobility across data centers through software defined networking," In Proc. IEEE Network Operations and Management Symposium (NOMS), pp. 88-96, 2012.

[130] R. Bifulco, M. Brunner, R. Canonico, P. Hasselmeyer, and F. Mir, "Scalability of a mobile cloud management system." In Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pp. 17-22. ACM, 2012.

[131] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds." In Proc. IEEE Military Communications Conference (MILCOM), pp. 835-840, 2014.

[132] H. Liu, C. Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," In Proc. ACM 20th international symposium on High performance distributed computing, pp. 171-182, 2011.

[133] T. T. Duong, and D. Q. Tran, "Mobility prediction based on collective movement behaviors in public WLANs," In Proc. Science and Information Conference (SAI), pp. 1003-1010, 2015.

[134] https://wiki.gentoo.org/wiki/Hostapd. 2015

[135] http://mediatomb.cc/. 2015

[136] H. Liu, H. Jin, C. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," Cluster computing, vol. 16, no. 2, pp. 249-264, 2013.

[137] A. Xiao, Y. Wang, L. Meng, X. Qiu and W. Li, "Topology-aware virtual network embedding to survive multiple node failures," In Proc. IEEE Global Communications Conference (GLOBECOM), 2014.

[138] M. M. A. Khan, N. Shahriar, R. Ahmed and R. Boutaba, "Multi-Path Link Embedding for Survivability in Virtual Networks," IEEE Transactions on Network and Service Management, vol. 13, no. 2, pp. 253 - 266, 2016.

[139] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. D. Turck and S. Latré, "Dynamic resource management in SDN-based virtualized networks," In Proc. 10th IFIP International Conference on Network and Service Management (CNSM), 2014.

[140] Y. Lin, H. Teng, C. Hsu, C. Liao and Y. Lai, "Fast Failover and Switchover for Link Failures and Congestion in Software Defined Networking," In Proc. IEEE International Conference on Communications (ICC), 2016.

[141] N. J. R. Potharaju, "When the network crumbles: an empirical study of cloud network failures and their impact on services," In Proc. 4th annual Symposium on Cloud Computing (SOCC '13), Santa Clara, California, 2013.

[142] K. Xi and H. J. Chao, "IP fast rerouting for single-link/node failure recovery," In Proc. Fourth International Conference on Broadband Communications, Networks and Systems (BROADNETS), Raleigh, NC, USA, 2007.

[143] T. Elhourani, A. Gopalan, S. Ramasubramanian, T. Elhourani, A. Gopalan and S. Ramasubramanian, "IP Fast Rerouting for Multi-Link Failures," IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 3014-3025, October 2016.

[144] O. Soualah, N. Aitsaadi and I. Fajjari, "A Novel Reactive Survivable Virtual Network Embedding Scheme Based on Game Theory," IEEE Transactions on Network and Service Management , vol. 14, no. 3, pp. 569-585, Sept. 2017.

[145] S. Sharma, D. Staessens, D. Colle, M. Pickavet and P. Demeester, "Enabling fast failure recovery in OpenFlow networks," In Proc. 8th IEEE International Workshop on the Design of Reliable Communication Networks (DRCN), 2011.

[146] X. Liu and D. Medhi, "Dynamic virtual network restoration with optimal standby virtual router selection," In Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS), 2016.

[147] M. Chowdhury, M. R. Rahman and R. Boutaba, "ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping," IEEE/ACM Transactions on Networking, vol. 20, no. 1, pp. 206-219, Feb. 2012.

[148] A. Jarray and A. Karmouch, "Decomposition Approaches for Virtual Network Embedding With One-Shot Node and Link Mapping," IEEE/ACM Transactions on Networking, vol. 23, no. 3, pp. 1012-1025, June 2015.

[149] N. L. M. v. Adrichem, C. Doerr and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks," In Proc. IEEE Network Operations and Management Symposium (NOMS), 2014.

[150] D. Eppstein, "Finding the k Shortest Paths," Society for Industrial and Applied Mathematics (SIAM) Journal on Computing, vol. 28, no. 2, pp. 652-673 , 1999.

[151] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden and M. Roughan, "The Internet Topology Zoo," The University of Adelaide, Australia, October 2011. [Online]. Available: http://www.topology-zoo.org. [Accessed March 2018].

[152] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an Inter-network," in Proc. of IEEE INFOCOM, 1996, pp. 594-602.

[153] B. Waxman, "Routing of multipoint connections," In IEEE journal on Selected Areas in Communications, vol. 6, no. 9, pp. 1617-1622, Dec. 1988.

[154] "IBM ILOG CPLEX Optimization Studio," Available: http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud, Accessed October 2016.

[155] "JavaScript Object Notation," Available: http://www.json.org/, Accessed October 2016.

[156] P. Gill, N. Jain and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," In Proc. of the ACM SIGCOMM 2011 conference, Toronto, Ontario, Canada, August 15-19, 2011.

[157] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali and C. Diot, "Characterization of failures in an operational IP backbone network," IEEE/ACM Transactions on Networking (TON), vol. 16, no. 4, pp. 749-762, August 2008.

[158] I. F. Akyildiz, A. Lee, P. Wang, M. Luo and W. Chou, "Research challenges for traffic engineering in software defined networks," IEEE Network, vol. 30, no. 3, pp. 52-58, June 2016.

[159] C. C. Meixner, C. Develder, M. Tornatore and B. Mukherjee, "A survey on resiliency techniques in cloud computing infrastructures and applications," IEEE Communications Surveys and Tutorials, vol. 18, no. 3, pp. 2244-2281, 2016.

[160] F. Hao, M. Kodialam and T. V. Lakshman, "Optimizing restoration with segment routing," in Procedings of 35th Annual IEEE International Conference on Computer Communications (IEEE INFOCOM), San Francisco, CA, USA, April 2016.

[161] H. Yu, C. Qiao, J. Wang, B. Wu and L. Li, "A Virtualization Layer Approach to Survivability," IEEE Transactions on Network and Service Management , vol. 11, no. 4, pp. 504-515, December 2014.

[162] S. Ayoubi, Y. Chen and C. Assi, "Towards Promoting Backup-Sharing in Survivable Virtual Network Design," IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 3218-3231, October 2016.

[163] A. Jarray and A. Karmouch, "Cost-Efficient Mapping for Fault-Tolerant Virtual Networks," IEEE Transactions on Computers, vol. 64, no. 3, pp. 668-681, 2015.

[164] S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra and F. Zeng, "Dedicated Protection for Survivable Virtual Network Embedding," IEEE Transactions on Network and Service Management, vol. 13, no. 4, pp. 913-926, December 2016.

[165] M. Furdek, N. Skorin-Kapov and L. Wosinska, "Attack-Aware Dedicated Path Protection in Optical Networks," Journal of Lightwave Technology, vol. 34, no. 4, pp. 1050-1061, February 2016.

[166] T. Guo, N. Wang, K. Moessner and R. Tafazolli, "Shared backup network provision for virtual network embedding," In Proc. of IEEE International Conference on Communications (ICC), Kyoto, Japan, June 2011.

[167] P. Thorat, R. Challa, S. M. Raza, D. S. Kim and H. Choo, "Proactive failure recovery scheme for data traffic in software defined networks," In Proc. IEEE NetSoft Conference and Workshops (NetSoft), 2016 .

[168] M. Kodialam and T. Lakshman, "Dynamic routing of restorable bandwidth-guaranteed tunnels using aggregated network resource usage information," IEEE/ACM Transactions on Networking, vol. 11, no. 3, pp. 399-410, June 2003.

[169] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger and D. Walker, "Abstractions for network update," In Proc. ACM SIGCOMM, Helsinki, Finland, August 2012.

[170] E. Zegura, K. Calvert and M. Donahoo, "A quantitative comparison of graph-based models for Internet topology," IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 770-783, Dec 1997.

[171] "NetworkX python based toolkit for complex networks," [Online]. Available: https://networkx.readthedocs.io/en/stable/ index.html.

[172] P. Erdos and R. Alfréd, "On the evolution of random graphs," Publ. Math. Inst. Hung. Acad, vol. 5, no. 1, pp. 17-60, 1960.

[173] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Department of Computer Science and Engineering, Washington University, Tech. Rep. WUCSE, vol. 35, pp. 1–11, St. Louis, MO, USA, 2006.

[174] A. Jarray and A. Karmouch, "Periodical auctioning for QoS aware virtual network embedding," In Proc. IEEE 20th International Workshop on Quality of Service, Coimbra, Portugal, 2012.

[175] C. Li, G. Li, P. Wai and V. Li, "A wavelength-switched time-slot routing scheme for wavelength-routed networks," In Proc. 2004 IEEE International Conference on Communications, Paris, France, 2004.

[176] H. Amarasinghe, A. Jarray and A. Karmouch, "Fault-tolerant IaaS management for networked cloud infrastructure with SDN," In Proc. IEEE International Conference on Communications (ICC), Paris, France, July 2017.

[177] H. Amarasinghe, A. Jarray and A. Karmouch,"Survivable IaaS Management with SDN," IEEE Transactions on Cloud Computing (under review).

[178] Statista market predictions, "Worldwide mobile app revenues in 2015, 2016 and 2020 (in billion U.S. dollars)," [Online] Available: https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/

[179] Yao, Jingjing, Ping Lu, and Zuqing Zhu. "Minimizing disaster backup window for geo-distributed multi-datacenter cloud systems." In Communications (ICC), 2014 IEEE International Conference on, pp. 3631-3635. IEEE, 2014.

[180] A. Jarray, J. Salazar, A. Karmouch, J. Elias and A. Mehaoua, "QoS-based cloud resources partitioning aware networked edge datacenters," In Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015.