TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ALGORITHMS

LIST OF ABREVIATIONS

ETS	École de Technologie Supeure
NFV	Network Function Virtualization
NF	Network Function
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
POP	Point Of Presence
ASC	Agence Spatiale Canadienne
SDN	Software-Defined Network
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
NAT	Network Address Translation
KVM	Kernal-based Virtual Machine
NIST	National Institute of Standards and Technology
OS	Operating System
VM	Virtual Machine
vCPU	Virtual Central Processing Unit
VNF	Virtual Network Function
SFC	Service Function Chain
SaaS	Software as a Service

PaaS	Platform as a Service
IaaS	Infrastructure as a Service
CP	Cloud Provider
SP	Service Provider
EC2	Elastic Compute Cloud
FIFO	First-In First-Out
I/O	Input/Output
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ICMP	Internet Control Message Protocol
VN	Virtual Network
DC	Data Center
SLA	Service Level Agreement
VLAN	Virtual Local Area Network
MPLS	Multiple Label Switching
VNF-FG	Virtual Network Function Forwarding Graph
ETSI	European Telecommunications Standards Institute
NP	Non-deterministic Polynomial time
SSC	Single Service Chain

NACHOS	Network-Aware Chains Orchestration
VNF-OP	Virtual Network Function Orchestration Problem
LOCF	Least Operational Cost First
LSCF	Least Synchronization Cost First
VNEP	Virtual Network Embedding Problem
NSCs	Network Service Chains

INTRODUCTION

Since its introduction as a concept by ETSI to decouple software from hardware by leveraging virtualization technology (Chiosi *et al.* (2012)), Network Function Virtualization (NFV) has been widely and rapidly adopted as more cost-effective and easy to manage replacement to traditional hardware-based middle-boxes. Such monolithic hardware solutions are not just expensive to obtain and maintain. They also make it difficult to re-allocate network functions such as firewalls, load balancers, intrusion detection systems (IDS), and network address translation (NAT). In contrast, NFV enables network functions to run on virtual machines (VMs) hosted by commodity servers as Virtual Network Functions (VNFs).

NFV offers numerous benefits to Cloud Service Providers (CSPs) including networking equipment cost reduction, power consumption minimization, scalability, elasticity, hardware reuse, easy multi-tenancy, and rapid configuration of new services (Chiosi *et al.* (2012)). NFV services are composed of a set of network functions traversing the path(s) from one or multiple sources to the destination called *Service Function Chains* (SFCs).

Resource allocation for service function chains is an intensely active research area. However, little attention has been given to optimally selecting the right type of cloud resources to be used to run the SFC components and to optimally embed these components into the infrastructure while taking into account operational costs (e.g., electricity cost, synchronization cost). This work aims at shedding the light on these challenges and putting forward efficient solutions to solve them.

Research Problem

In Cloud environments, Cloud Providers offer their infrastructure, which is composed of a single or multiple Data Centers (DCs) consisting of compute and networking equipment. NFV, as an emerging technology, follows the same concept and takes it to a higher level by leveraging

virtualization technology to virtualize network functions; Yet, it has introduced new challenges to network operators such as (i) finding the most appropriate placement of these VNFs into the Cloud infrastructure, (ii) chaining multiple of VNFs such that the order of the service chain is not violated, and (iii) evaluating the efficiency in terms of performance.

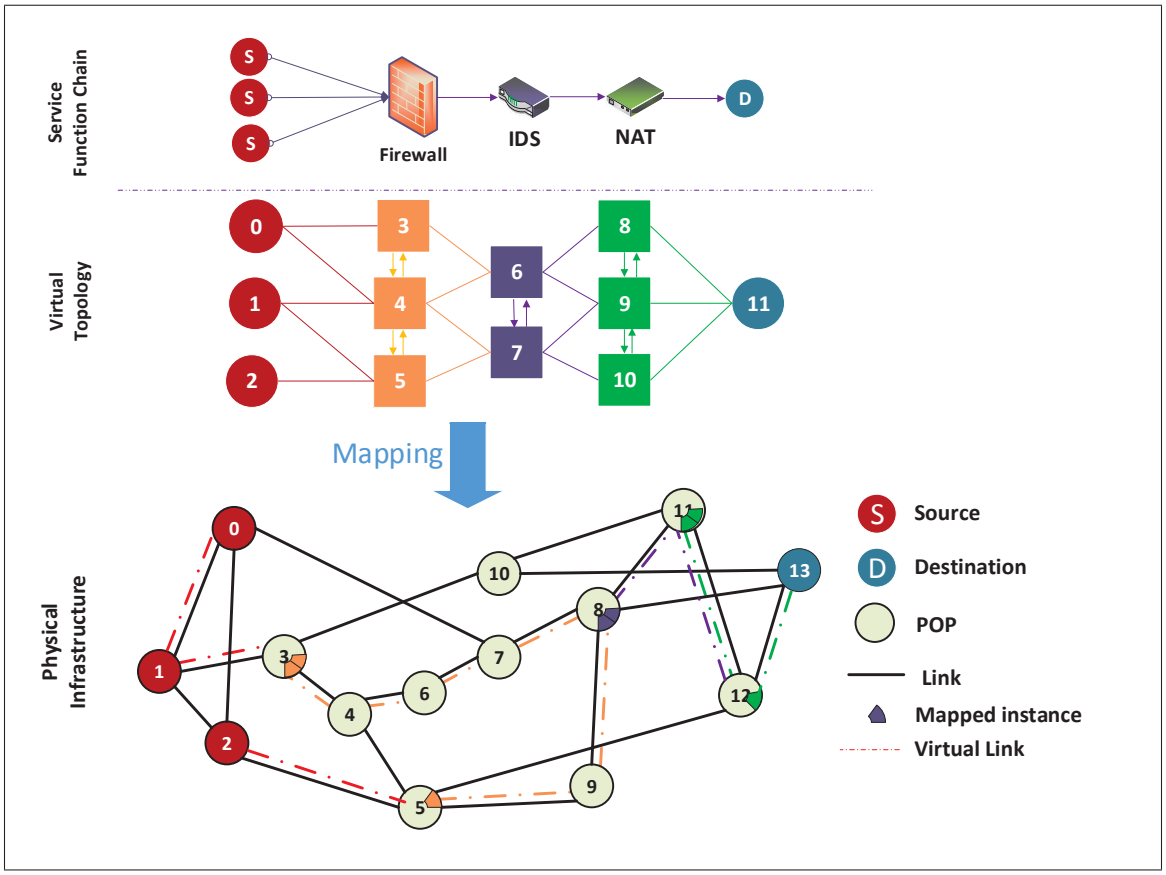


Figure 0.1 VNF instances placement and chaining problem

Basically, VNF placement problem is defined as it is the phase of finding the most appropriate location for VNF-SFC to be embedded in Cloud infrastructure. Thus, finding the best placement of a single VNF may not be as challenging as allocating resources (i.e. CPU, memory, storage, and bandwidth) for a set of VNFs representing a service request. Specifically, when an SFC consists of one VNF, it is a matter of finding a POP has enough resources (i.e. CPU, memory, and storage) to minimize the costs. However, when an SFC consists of multiple VNFs, the

placement problem becomes even more difficult since VNF-SFC is restricted to some certain order as well as link and VNF delays. To this end, we deduce that the longer the SFC is the more difficult it becomes for the operator to find a placement for the service request.

Realistically, POPs are usually distributed in wide geographical locations (i.e. Montreal, Toronto, and Houston). The energy price for each POP varies depending on its location; hence, the operational cost (i.e. cost of running VNFs) is not the same. Furthermore, from Service Providers' perspective, they aim at provisioning service requests and maximizing their total revenue. This can only be achieved by allocating resources for VNF-SFCs such that the service request is provisioned at a competitive and affordable cost to the end-users.

Besides VNF placement problem, a little effort has been made towards evaluating the capability of VNFs to process large amounts of traffic, especially with the growth of number of end-users adopting Cloud services. A single VNF might be sufficient to process a certain amount of requests before it becomes overloaded. This occurs especially when there is a surge of requests for some certain service. Let us take Netflix as an example. The majority of users may not watch videos on Netflix during business week days as much as they do on weekends. Hence, it is worth mentioning that when there is a sudden spike of requests, one VNF of Video Optimizer function may not be enough to process all the received requests. Therefore, the number of VNF instances must be estimated beforehand in order to allocate resources for VNF instances that can process requests more efficiently.

Research Objectives

In this thesis, we will formulate the VNF placement and chaining problem using integer linear program (ILP). We will then extend our solution and propose two heuristics that find a placement for SFCs that have multiple instances in the infrastructure to reduce operational cost taking into account the synchronization cost between each pair of instances.

- Guarantee service performance by satisfying end-to-end delay;
- Minimize NFV Service provider total costs by taking into account the different energy prices based on the geographically distributed locations;
- Evaluate VNF performance in terms of processing time, packet arrival rate, and packet loss;
- Determine the number of instances required to cope the traffic demand;
- Find the most appropriate placement of VNF-SFCs including the instances in such a way that it reduces the total operational and synchronization costs.

Publications

Throughout my Master's, we published two conference papers and submitted a journal paper. The list of papers is as follows:

- "*Profit-driven resource provisioning in NFV-based environments*", published at the IEEE International Conference on Communications (ICC) tht was held in Paris, France, May 2017.
- "*Price and Performance of Cloud-hosted Virtual Network Functions: Analysis and Future Challenges*" accepted at the IEEE International workshop on Performance Issues in Virtualized Environments and Software Defined Networking (PVE-SDN 2018) in Montreal, Canada in June 2018.
- "*Cost-Aware VNF Placement in Cloud Infrastructures*" submitted to the IEEE Transactions on Network and Service Management (IEEE TNSM).

Thesis Structure

This thesis is divided into four chapters as shown in Figure 0.2:

- The first chapter presents the technical background. This chapter is divided into two parts. The first part illustrates Cloud Computing, Virtualization, SDN and NFV concepts whereas the second part of this chapter describes the problem of VNF placement and chaining in much depth.

- The second chapter concentrates on the related work. Throughout this chapter, we will discuss the proposed solutions for VNF placement and chaining problem then compare our solutions to these proposals.

- The third chapter is dedicated to the solution design. This first part of this chapter will formulate VNF Placement and Chaining problem using ILP. We then propose greedy variant algorithms to find the best placement of VNFs in NSFNET considering a toplogy of POPs.

- In the fourth chapter, we present the VNF performance evaluation results and the simulation results for placement and chaining of VNF SFC as well as VNF instances.

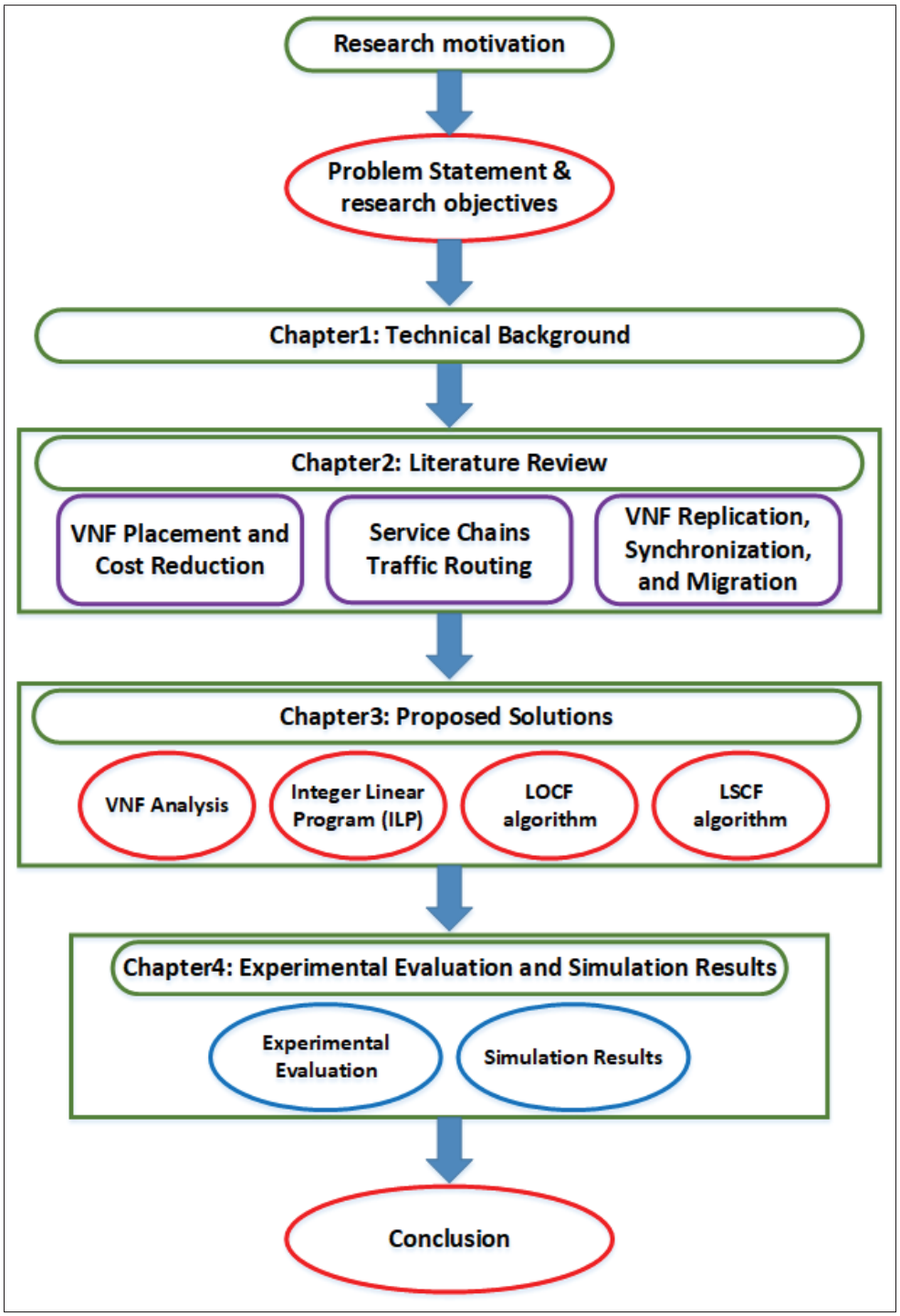


Figure 0.2 Thesis Structure

CHAPTER 1

TECHNICAL BACKGROUND

Throughout this chapter, we will highlight some key concepts concerning virtualization, VMs versus Containers, Network Virtualization, and NFV. We will also carry out some of the advantages and disadvantages of each of these technologies. Furthermore, we will discuss NFV Placement and Chaining problem and introduce its correspondence to Software Defined Networking (SDN) concept. In the following section, we will define and provide some key concepts about Cloud Computing.

1.1 Cloud Computing

1.1.1 Definition

According to NIST (Mell & Grance (2011)), Cloud Computing is defined as *"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. This cloud model is composed of five essential characteristics, three service models, and four deployment models".* Essentially, Cloud Computing is characterized by the following main features:

On-demand self-service: This indicates that provisioning of computing resources and features can be achieved automatically as required by consumers without the need to interact with service providers.

Broad network access: The availability of computing capabilities over accessible network through different client platforms such as cell phones, laptops, tablets, and workstations.

Resource pooling or Multitenancy: This refers to pooling of provider's computing resources in order to offer services to the end-users through a multitenancy-based model consisting var-

ious virtual and physical resources that are allocated and deallocated based on the customer's demand.

Rapid elasticity: Computing capabilities should be allocated and freed elastically and sometimes manually or automatically in order to scale in and out to meet the demand and use resources more efficiently (Mell & Grance (2011)).

Measured service: A pay-per-use model was designed to provide Cloud computing services. This pricing scheme varies depending on the service. For instance, a VM can be leased for a SaaS provider from an IaaS provider to serve clients for an hour. On the contrary, a SaaS provider can specify the service fee for its clients based on the number of clients the SaaS provider serves. Utility-based pricing is used to lower the cost of operating a service (Zhang *et al.* (2010)).

1.1.2 Cloud Computing Layered Model

The architectural model of Cloud Computing is divided into four main layers as depicted in Figure 1.1. Each of these layers performs some tasks, and they are as following:

- *The Application Layer*: This layer ranks on top of the hierarchy of layers and contains all Cloud applications. These applications are unlike traditional applications as they can be automatically scaled to improve performance as well as availability, and they also reduce operational cost.
- *The Platform Layer*: This layer is created atop infrastructure layer and is composed of operating systems and application frameworks. The main purpose of this layer is to reduce applications deployment directly into VM containers.
- *The Infrastructure Layer*: This layer is also called virtualization layer. In this layer, the infrastructure builds storage and computing resources by using hypervisors such as KVM, Vmware ESXi, or XEN to deploy and manage Cloud's virtual resources.

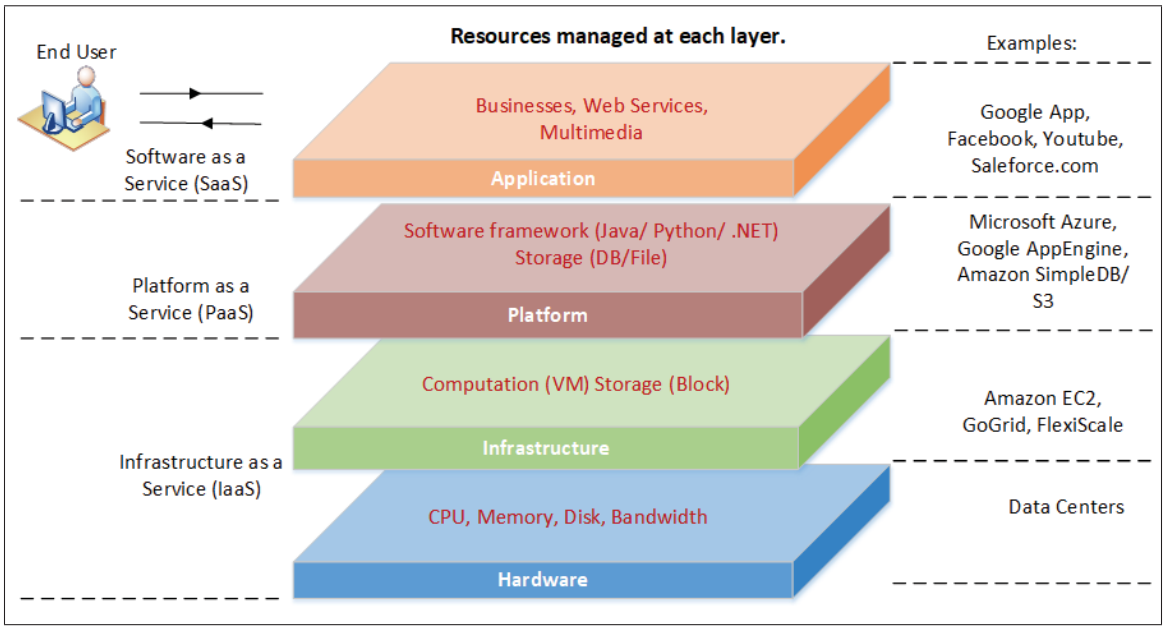


Figure 1.1 Cloud Computing Architecture (Zhang *et al.* (2010))

- *The hardware layer*: It is responsible for the management of physical resources (i.e. CPU, memory, storage), including servers, routers, switches. Typically, the hardware equipment is usually implemented in Data Centers or any point of presence (POP) (Zhang *et al.* (2010)).

1.1.3 Cloud Service Models

The service model basically relies mainly on the architectural layer; however, it is divided into three models as following:

- *Software As a Service (SaaS)*: SaaS allows the end-users to access and manage their applications deployed in Cloud infrastructure either through a web-based or program interface. The end-users are allowed to access only but not control the underlying infrastructure of the Cloud including servers, storage, or operating systems. Examples of SaaS are Amazon EC2, GoGrid, and Flexiscale.

- *Platform As a Service (PaaS)*: PaaS allows end-users to deploy their defined or customized applications onto cloud infrastructure using programming languages and services supported by the provider. Examples of PaaS are Microsoft Azure and Google App Engine.

- *Infrastructure As a Service (IaaS)*: The end-users is empowered to provision physical resources (i.e. CPU, memory, and/or storage). The end-user cannot manage or control Cloud infrastructure; however, they only have control over operating systems, storage, and some deployed applications.

1.1.4 Cloud Computing Types

In this section we will introduce the main types of Cloud Computing, which include the following:

- *Private Cloud*: In this type, Cloud infrastructure is exclusively used by one corporate consisting multiple end-users. It can be operated either by the organization itself or a third-party. It can also be available on or off premises.

- *Community Cloud*: The infrastructure of this type of Cloud can be provided to a specific community of end-users that have common objectives. The infrastructure can be owned or operated by any of this community of end-users, a third-party, or a combination of both.

- *Public Cloud*: The infrastructure of this type of cloud can be owned and managed by an academic or business organization and offer a service for the public. This may exist on the cloud provider's infrastructure.

- *Hybrid Cloud*: As the name suggests, hybrid cloud infrastructure may compose two or more different cloud infrastructures (i.e. private, community, and/or public)(Mell & Grance (2011)).

1.2 Virtualization

1.2.1 Definition

Virtualization is a mechanism that hides the characteristics of compute physical resources in such a way that other applications, systems, and end-users use these resources without any direct interaction (IBM (2007)). As a practical example of that, when google map is requested from a cell phone or when you shop online for the most reasonable prices based on the location, virtualization is basically used. Hence, we can say that the concept of virtualization is quite wide as it can be applied to several devices, physical machines like servers, operating systems (OSs), applications and networks (IBM (2007)).

1.2.2 Virtual Machines

A virtual machine can be defined as the logical representation of the computer. It is mainly based on decoupling the hardware from the operating system to provide more flexibility and maximize the efficiency of utilizing underlying resources. In a virtualization environment, a single physical hardware is capable of accommodating multiple VMs sharing the same physical machine's resources. However, each VM is isolated and functions independently on the same physical host. Hence, in case of any failure in any of these VMs, this error does not affect other VMs (IBM (2007)).

The best form of machine virtualization can be demonstrated in Virtual Desktops and Virtual Servers. Furthermore, server virtualization enables network operators to consolidate their physical servers and partition them in order to host multiple secure virtual servers as depicted in Figure 1.2. Therefore, server virtualization helps reduce the hardware acquisition and also cuts down management costs. If resource requirement of one of the applications running on any of the virtual machines increases, this VM can be migrated to another physical server with more resources available to meet the resources requirements.

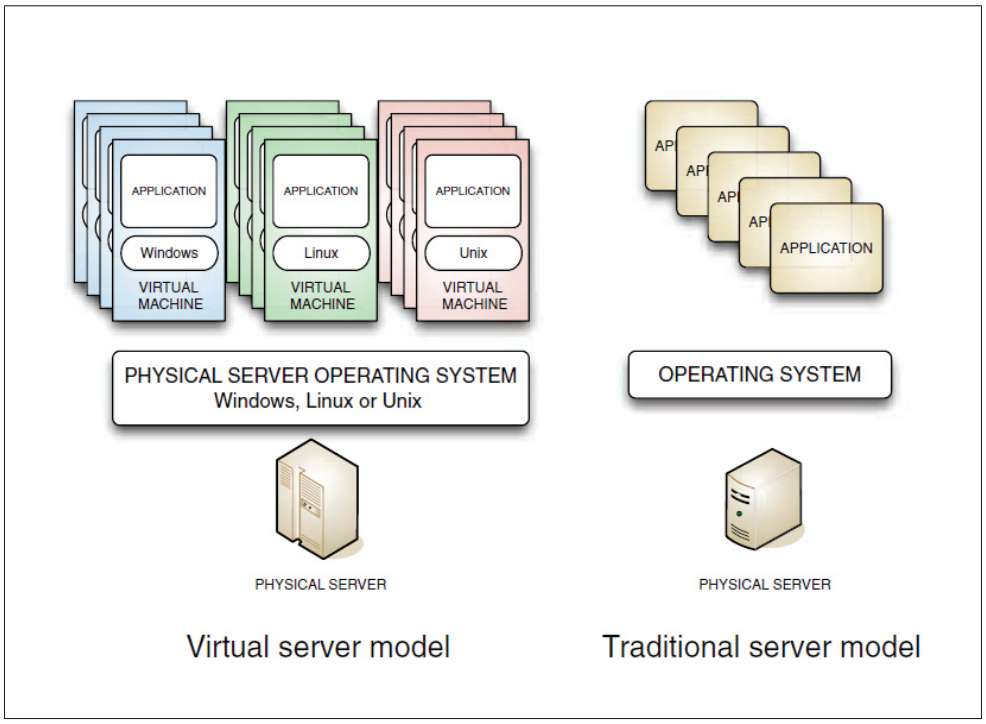


Figure 1.2 Server Models (IBM (2007))

Generally, VMs are controlled through a Hypervisor, which is a firmware or a software used to employ the physical machine's resources and manage multiple VMs. It is mainly responsible for creating, allocating resources (i.e. CPU, memory, storage), and destroying VMs when they are no longer needed. These hypervisors can either run on top of hardware such as VMWare ESXi, XEN, and KVM and this type of hypervisors is called **native or bare-metal** hypervisors. The other type runs on top of conventional operating system like any other programs and applications do, and this type is called **hosted** hypervisors.

1.2.3 Containers

As virtual machine (VM) technology has achieved a tremendous growth during the past decade, a new server virtualization form called "Containers" has been increasing rapidly in popularity. Containers is a promising concept aims at accelerating deployment cycles of the applications and fundamentally increasing the agility of Cloud services. They provide an isolation to the

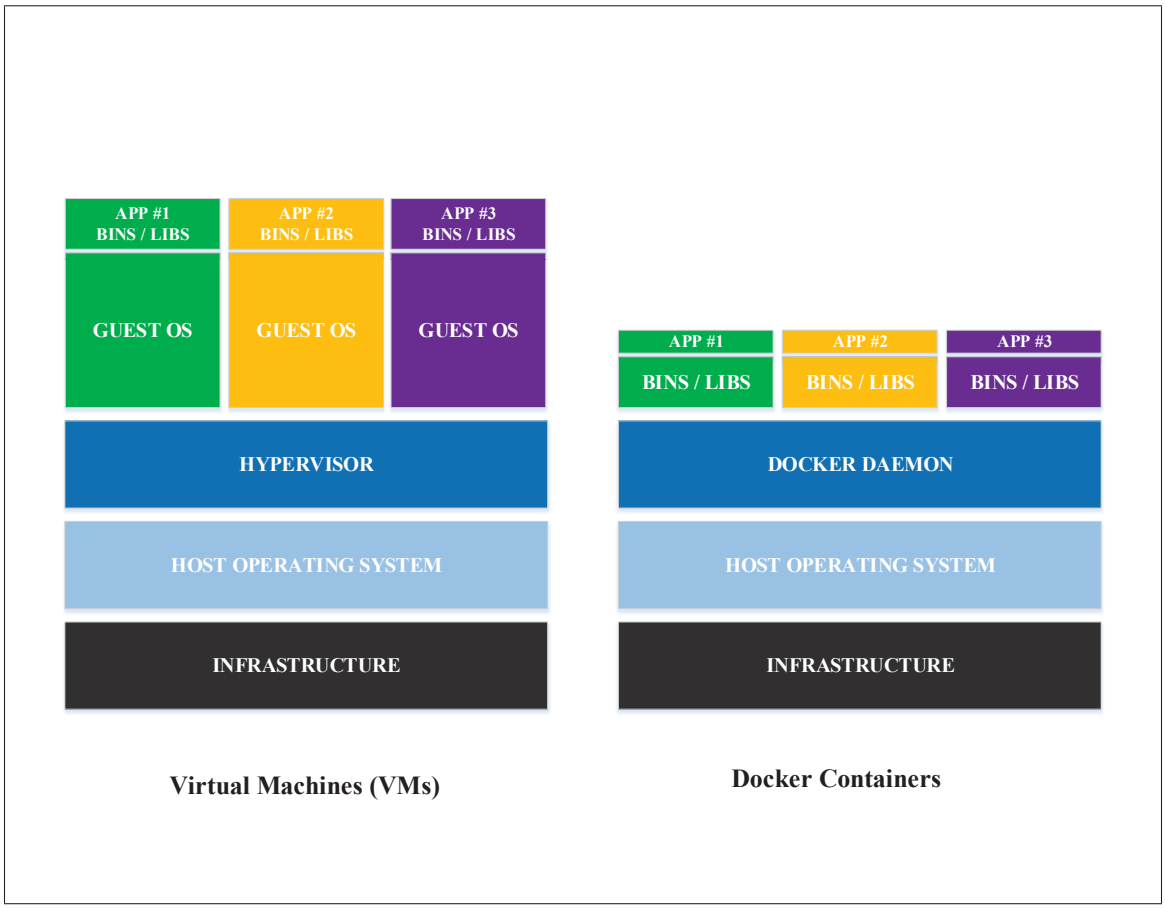


Figure 1.3 Comparison between VMs and Containers

applications from each other using OS-level virtualization characteristics (i.e. namespaces [resource isolation].) and Linux control groups[resource control] (Networks (2016)). For instance, Containers, in comparison to VMs, are much smaller in size and hence, the OS in each Container requires less booting time and applications launch faster.

Although Containers were first introduced in 2007, there are three main factors have made them be trendy in Cloud deployments these days. Moreover, exactly like VMs, Containers require a physical infrastructure and a host OS. To manage VMs, we need to install a hypervisor atop the OS whereas in Containers, we need a Docker engine (Docker Daemon), which is a service runs in the background of the host OS and manages everything required to run and interact with Docker Containers as depicted in Figure 1.3. VMs provide an isolation for the OS whereas

Docker Containers can be used to isolate different applications running on the same host OS. In other words, VMs and Containers have different use cases and have different isolation levels.

1.3 Network Virtualization

Network virtualization (NV) is basically defined as the abstraction of the network endpoints from the physical components or nodes (i.e. servers, switches, and routers) in the substrate network. Virtualizing network components enables network operators to design and manage their own virtual network(s) independently. In other words, network operators can deploy and manage their VN components such that it performs as if it is a network located in a different location (Doherty (2016)).

It is worth mentioning that network virtualization is not a new concept. In fact, it has been there for years and there are several forms of network virtualization including virtual private networks (VPNs), virtual LANs (VLANs), and Multiprotocol Label Switching (MPLS) (Doherty (2016)) (Carapinha & Jiménez (2009)). These technologies allow network operators to gather these physical nodes into logical groups so that it makes them seem to have functioned on top a single physical machine, which allows more efficient controllability in terms of network management, traffic control as well as security (Doherty (2016)).

In general, VNs consist of two main components, namely nodes and links virtualization (Carapinha & Jiménez (2009)). Link virtualization enables transporting a set of virtual links on a shared physical link. On the other hand, a virtual node can be accommodated on any physical node in the network that supports virtualization technologies. Saying that means each physical node can host multiple secure virtual nodes and manage them via a hypervisor(Carapinha & Jiménez (2009)).

1.4 Network Function Virtualization

Network Function Virtualization (NFV) (M. Chiosi, S. Wright, and others (2012)) is an emerging technology that decouples the software from the hardware of the traditional middle-boxes

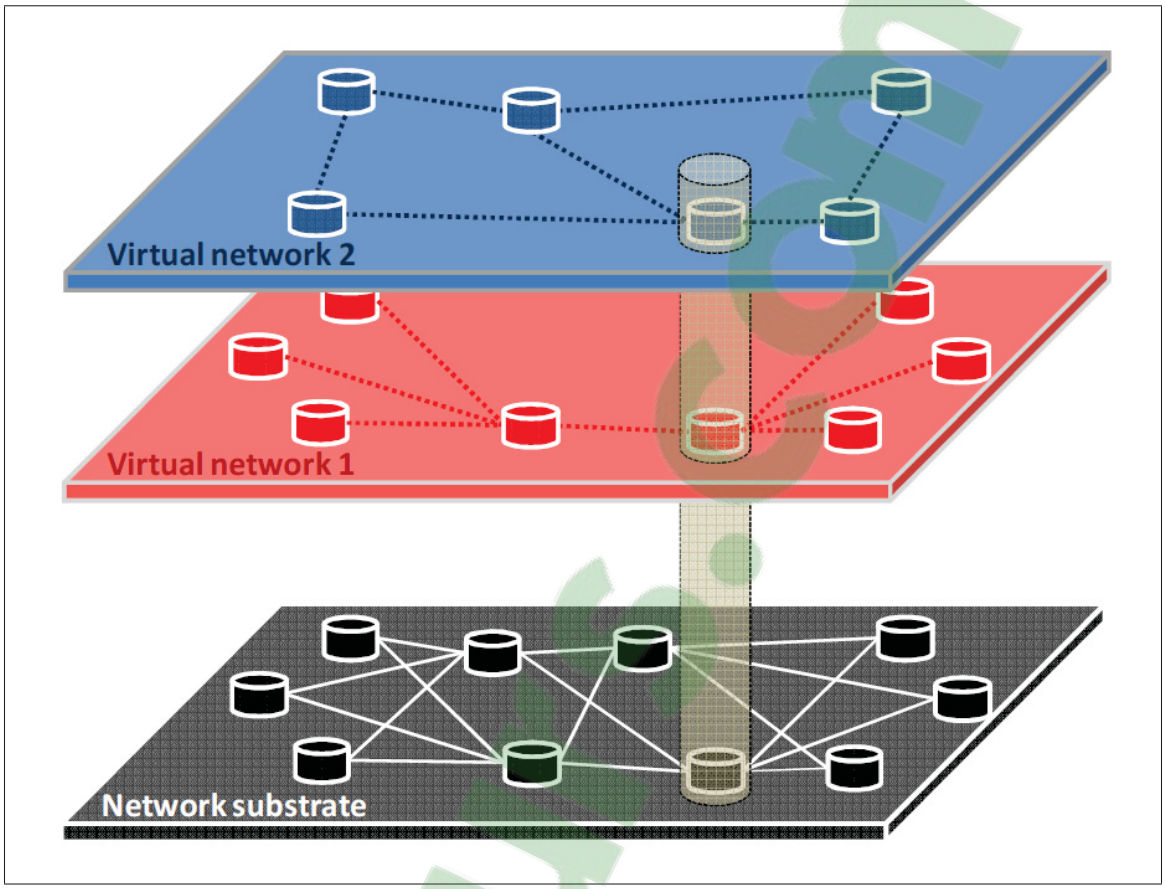


Figure 1.4 Network Virtualization Model (Carapinha & Jiménez (2009))

and enables network operators to design and manage their network services more flexibly. NFV leverages virtualization technology to provide several advantages such us equipment cost reduction, re-usability of resources, scalability, and ease to market and innovate. Each service consists of one or multiple virtual network functions (VNFs) that are placed in order and traversing the path from the source to the destination. This sequence is called a service function chain (SFC) or VNF Forwarding Graph (VNF-FG) (Lee *et al.* (2014)).

ETSI (M. Chiosi, S. Wright, and others (2012)) states that *"Network Functions Virtualisation aims to transform the way that network operators architect networks by evolving standard IT virtualisation technology to consolidate many network equipment types onto industry standard high volume servers, switches and storage, which could be located in Datacentres, Network*

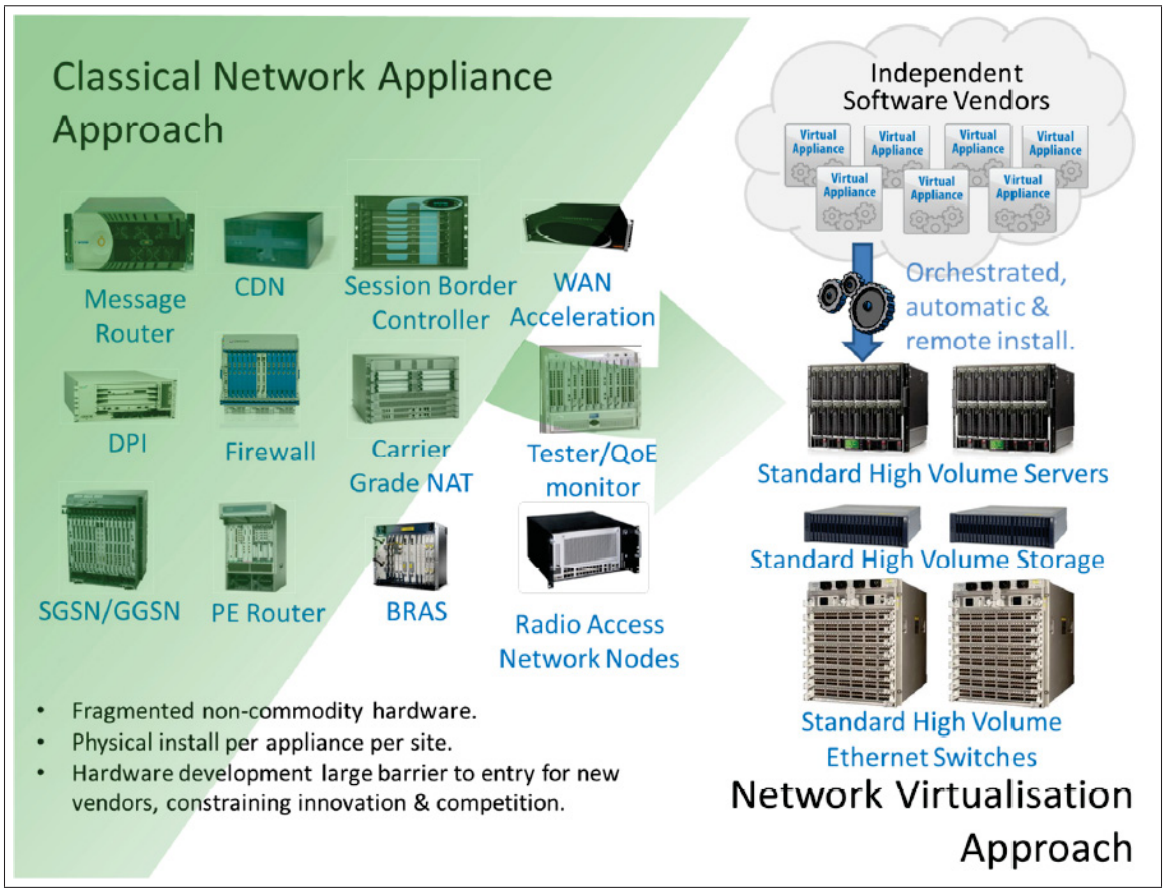


Figure 1.5 Network Function Virtualization Vision (M. Chiosi, S. Wright, and others (2012))

Nodes and in the end user premises". Separating hardware from software means that NFV offers several advantages to the industry including the following:

- NFV reduces the cost of compute and networking equipment by leveraging virtualization technology;
- Virtualizing network functions speeds up the time to market. This can be achieved by reducing the network operator cycle's innovation (M. Chiosi, S. Wright, and others (2012));
- NFV ensures the availability of multi-tenancy, which enables network operators to use a single platform for multiple applications and end-users. In other words, network operators can share physical resources across multiple services to different end-users;

- It targets a large population of end-users, and therefore, services can be scaled up and/or down as needed;
- Openness: It urges end-users and researchers in academia as well as small players to innovate services rapidly at lower risk (M. Chiosi, S. Wright, and others (2012)).

In spite of all the advantages that NFV provides to the industry, determining the number of required VNF instances and finding available resources in the infrastructure to host these instances are major challenges that will be discussed in the following section.

1.5 Placement and Chaining Problem Description

In this section, we demonstrate VNF instances placement problem taking into account different electricity prices of POPs that are located in different places. VNF Instances Placement problem is mainly divided into two sub-problems. The first is to determine the number of VNF instances required through to process some certain amount of traffic demand. Secondly, finding the most appropriate placement for VNF instances in the infrastructure such that the operational cost for requests and the synchronization costs between each pair of VNF instances of the same type are minimized. We will also discuss a few key ideas in this work before diving into details of the problem.

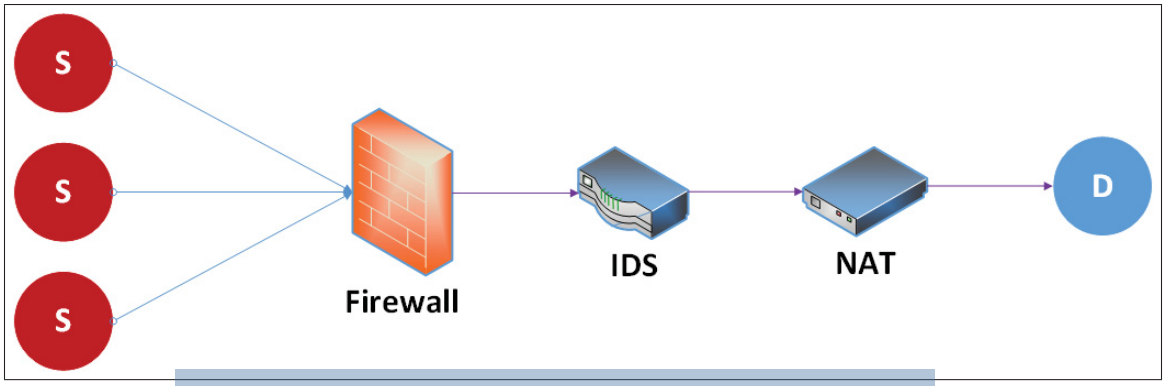


Figure 1.6 Service Function Chain

We firstly start by defining *service function chain* (SFC) or service request is a set of VNFs traverse the path from the sources to a single destination. SFCs are either composed of a single or multiple VNFs of different types as shown in Figure. (1.6) as it may begin with a load balancer followed by a firewall ending with an IDS. Requests received in the beginning of each SFC (at the first POP along the path to the destination) are processed in a First-In-First-Out (FIFO) manner as in (Alsubhi *et al.* (2012)).

VNFs are installed on top of commodity server and placed in point of presence (POP) facilities. Each POP consists of multiple homogeneous servers, switches and routers and they are distributed in different geographical locations and connected via links. Since these POPs are located in different regions, this means that POPs' energy prices vary depending on where the POP is located. VNFs in each SFC may contain one or more instances of the same type. For example, a service chain is composed of a firewall, load balancer, and NAT. A single VNF of each firewall and load balancer functions may suffice the amount of traffic received. However, two instances of NAT are needed to handle the forwarded traffic from the load balancer. This example shows that the number of instances in each VNFs varies depending on the processing capacity of the VNF in each request. Thus far, we learn that by determining the number of required instances for VNFs, we ensure the continuity of a service request. This can only be achieved by knowing the maximum processing capacity of the VNF at which resources usage is maximized while avoiding packet loss.

Once the requirements of the service request have been defined, Service Provider will be responsible for allocating resources for these VNFs into CLoud's infrastructure. Figure (4.7) depicts the mapping of VNF service chain into physical infrastructure. To ensure that each service request is reliable and there is no service disruption due to overloaded VNFs in the service chain, the number of VNF instances required to process some certain traffic demand needs to be determined.

Afterwards, embedding phase takes place. In this phase, physical resources need to be found in the POPs connecting the sources to the destination. Saying that means, each service request

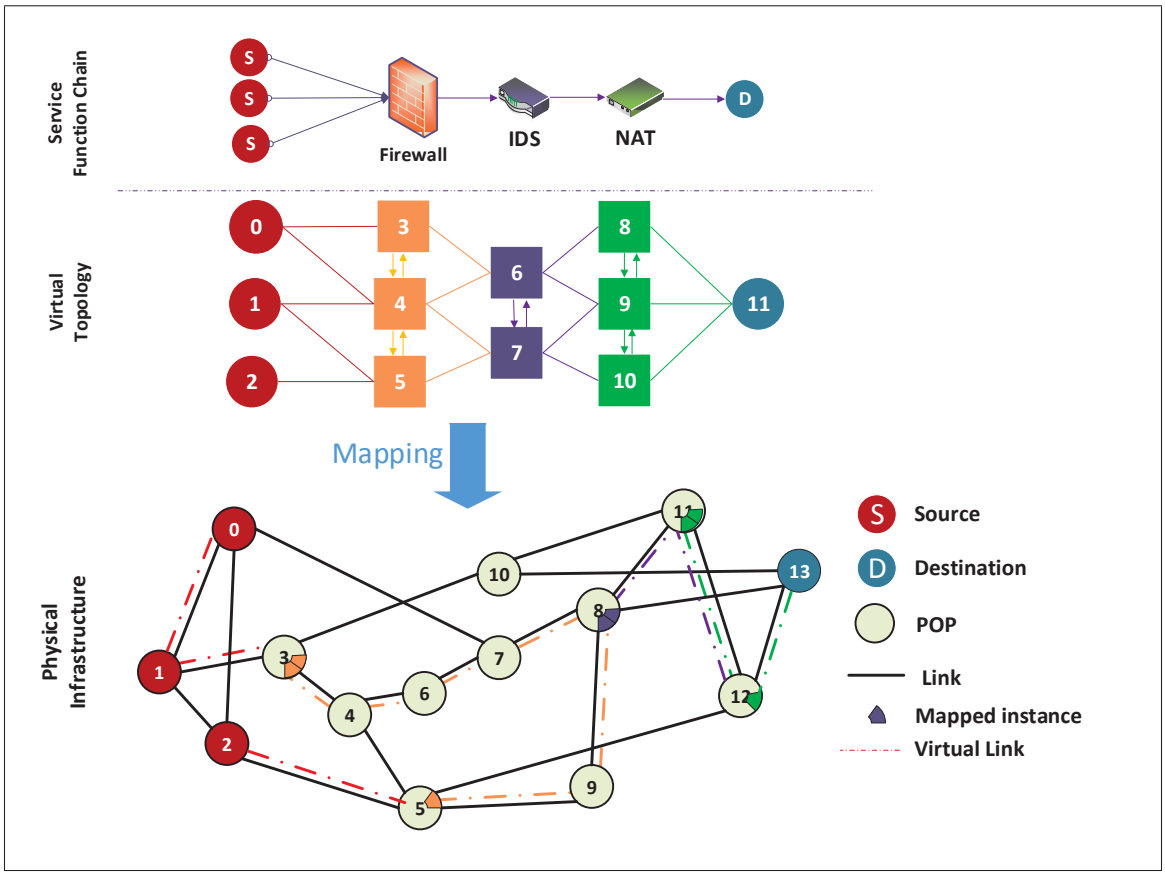


Figure 1.7 VNF Instances Embedding

has an amount of traffic and an SFC composed of VNFs should be instantiated in some certain order. Network Operators on the other hand, are responsible for determining the number of instances needed for each VNF in an SFC. Furthermore, since they have global-view of their infrastructure, finding the most appropriate location in the infrastructure to embed VNF instances in such a way that the embedding does not violate the SFC order nor it maximizes the operational cost and data rate exchanged between each pair of instances. Allocating resources for VNF instances and embed them into the physical infrastructure is a difficult task due to the fact that using a fixed number of VNF instances may under-utilize or exhaust physical resources. Therefore, there is a need to instantiate multiple instances based on the received traffic inbound. If this traffic is larger than the processing capacity of one instance, then another instance of the same VNF type is required in order to achieve continuity of SFC. To determine

the number of instances for each VNF to provision a service function chain, we take into account future traffic prediction, processing time, maximum end-to-end delay.

• **Future Traffic Demand:** since high traffic demand maximizes the utilization of VNF resources, this raises a question whether a single VNF is sufficient to serve multiple requests. If resource utilization, specifically CPU reaches its peak, then the VNF will drop future requests. For this reason, the performance of VNFs must be evaluated in order to estimate the number of instances required to serve the demand and minimize dropping requests due to lack of resources, and at the same time, to use the resources more efficiently.

• **End-to-End delay:** this refers to the total amount of time that a certain amount of traffic takes to traverse across the path, which is composed of set of VNF instances, from the sources to the destination. Moreover, each service chain has VNF instances varies in number depending on the incoming traffic at the source. This means that each VNF has different processing capacity varies in terms of processing time, which in turns affects the total delays.

• **Operational Cost:** This metric is defined as the cost of deployment and transfer of a VNF image from a location to another within the infrastructure and running it for a certain amount of time (i.e. request lifetime) where each request has different lifetime. VNF deployment depends on VNF type (i.e. Firewall, IDS, or Load balancer), Operatin System, and VNF license cost. Furthermore, the operational cost also takes into account different energy prices of POPs that are distributed across a wide geographical area.

• **VNF instances Synchronization:** VNF Synchronization is the rate at which data is exchanged between multiple VNF instances of the same type. This rate varies depending on the functionality of the network function. For example, a firewall has the least synchronization rate compared to IDS and NAT due to the fact that firewall rarely synchronizes the configuration with other adjacent firewall instances whereas IDS requires high synchronization rate between its peer instances in case an attack is detected in the network. Similarly in case of NAT where packets need to be processed and forwarded to end users. Consequently, iptable values change

frequently and therefore, to avoid the assignment of the same address to the host machines multiple times, NAT instances need to be synchronized. In this contribution, we assume that the synchronization comes at a cost; however, we do not dive into much details of how VNF instances are synchronized.

1.6 Conclusion

In this chapter, we have presented Cloud Computing and its modules, Virtualization, Network Virtualization and eventually introduced Network Function Virtualization (NFV). We have also listed the advantages and disadvantages of each technology, specifically, how virtualization cuts down Capital and Expenditure costs. NFV has been introduced in the recent few years. Although there are many proposals address VNF placement problem, none of them have evaluated VNF performance in order to determine the number of instances required to satisfy a certain amount of traffic. Not to mention that VNF placement becomes even more challenging taking into account the synchronization cost between each pair of VNF instances in the SFC.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we discuss some of the existing solutions concerning VNF Placement and Chaining problem. While presenting these solutions, we will point out to their objectives and limitations and then compare them to our contribution. We have classified the existing works into three main categories depending on their objectives. The three categories are: (i) VNF placement and cost reduction, (ii) service chains traffic routing, and (iii) VNF replication, synchronization, and migration.

2.1 Related Work

2.1.1 VNF placement and cost reduction

In this section, we discuss the most significant works that have been done towards VNF placement with an aim of costs reduction. We highlight the most significant VNF placement approaches taking into account different metrics.

In (Wang *et al.* (2017)), the authors address the optimal VNF deployment problem in cloud infrastructure. The authors in this work propose an online provisioning algorithm and a multi-armed bandit optimization scheme. The online provisioning algorithm minimizes the instantiation cost whereas the optimization scheme determines VNF instances placement on servers. The online provisioning algorithm is a randomized online heuristic adapted based on ski-rental algorithm (Karlin *et al.* (1994)) (an online buy/rent decision making algorithm), that dynamically estimates the number of VNF instances required to minimize the overall cost by potentially including the instantiation cost for new requests, or keeping the instances running, and paying for the operational cost. On the other hand, the multi-armed bandit optimization scheme includes a bandit-based online learning algorithm for VNF instances placement that reduces traffic congestion within data centers by predicting congested loads and avoiding them while placement has taken a place. The authors evaluate the performance of their online al-

gorithms running trace-driven simulation and small-scale experiments by generating two types of background traffic: 1) uniform background traffic, where a server sends a traffic rate of 200Mbpsto every other server with a range of [0,1] Gbps, and 2) permutation traffic where random traffic routes 2 Gbps between two servers with a range of [0,3] Gbps. The results show that the proposed algorithms achieve significant reduction in terms of competitive ratio, operational, and instantiation costs. The authors have addressed various types of workloads; however, this work does not clearly identify how background workloads affect VNF placement decision. They also consider multiple VNF instances without evaluating the processing capacity of VNFs. They assume that load balancers, firewalls, and NATs have the same processing capacity whereas IDS network functions process less traffic rate, which is not realistic.

Mijumbi *et al.* (2015b) address VNF embedding and scheduling onto virtual network problem. They formulate the online VNF embedding and scheduling problem using ILP and propose three greedy algorithms and a tabu search-based heuristic to solve this problem. The authors suggest that VMs are already mapped into the physical network and each VM can process multiple VNFs using containers assuming that the size of VNFs is considerably light. Not to mention that each VNF must be processed in a specific order. For scheduling, they assume that each service request has a deadline requirement that if exceeded, the request will be rejected. The greedy algorithms embed requests as they arrive and then rank them based on greedy criterion (i.e. least loading, least processing time, and shortest VM queues). Tabu Search aims at finding a solution and then attempts to optimize this solution by exploring the neighbours. The results show that the proposed tabu search-based algorithm outperforms the greedy algorithms in terms of acceptance rate, total cost, and total revenue. Using containers to accommodate VNFs in a single VM may be suitable for some functions, but not all of them. However, VMs offer isolation for each VNF despite the fact it utilizes more resources than containers. Furthermore, the proposed algorithms are incapable of adapting to the changes of network conditions. Also, hosting VNFs into one VM may lead to unpredictable over-utilization of resources (i.e. CPU, memory, and storage). Not to mention that this work does not take into account the need to embed multiple VNF instances to meet high traffic demand.

Bari *et al.* (2015b) address VNF Orchestration Problem (VNF-OP). In more details, the problem is divided into VNF deployment cost while provisioning service chain, energy cost for running VNFs, and traffic forwarding from/to VNFs. The authors formulate VNF-OP using ILP model with an objective of minimizing the deployment, energy, and traffic forwarding costs. They then further their solution by proposing a heuristic that finds a feasible embedding of service chains that minimizes the operational cost and resource usage in larger scale. Moreover, they compare their proposed heuristic with the ILP model using CPLEX and the results show that the proposed algorithm finds an embedding for service chains 1.3 times faster than the CPLEX solution. Not to mention that the execution time is 56-3500 times faster than CPLEX solution. The authors assume that one VNF for each function in the service chain is sufficient to handle traffic demand of any size, but this is not practical in production environment. Furthermore, the authors consider that each VNF has distinct resource requirements; however, this leads to a wastage of resources at times when low rate of traffic load is received. They also have not considered instantiating multiple instances instead of having one large VNF instances to reduce energy cost and power consumption.

Luizelli *et al.* (2015) propose an optimization model based on integer linear programming (ILP) that addresses VNF placement and chaining problem, and an algorithm that copes with large infrastructure. The problem is mainly divided into three phases: placement, assignment, and chaining. As for placement phase, it aims at determining the number of VNFs needed and where to place them. The assignment phase determines which VNF will be responsible of each flow. In the chaining phase, network paths are created to interconnect VNFs. The proposed model reduces the end-to-end delays and prevents resource over-provisioning. However, the placement does not take into consideration the shortest path between the source and the POP into which the VNFs are placed. The estimate number of instances for each VNF in a SFC is not considered in this work.

Clayman *et al.* (2014) propose a multi-layer architecture for management and orchestration that supports (SDN) and (NFV). This architecture consists of four layers: application, orchestration, abstraction, and infrastructure. The architecture and management scheme ensure automatic

network services allocation and VNF placement where VNFs and virtual links are dynamically placed when needed. The results show that each configurable placement engine demonstrate different behavior, and provide different placement strategies for virtual network functions. However, this study does not take into account other types of VNFs such as Firewall, DPI, and IDS for placement. In addition to that, chaining of VNFs is not considered in this work. Furthermore, the authors do not use a realistic number of hosts in the experiments to evaluate the demand of network functions in the production environment.

Sahhaf *et al.* (2015) address the issue of mapping NFV-based services onto physical network components. They propose an efficient cost-effective algorithm for virtual network embedding problem (VNEP) that maps Network Service Chains (NSCs) onto the physical infrastructure considering network function NF decompositions. This is done by translating a NF to a more refined NFs such as Firewall to iptable-based firewall and/or Flowtable-based firewall. The proposed algorithm reduces mapping cost based on service chaining requirements and increases request acceptance ratio. The results of this work show that the resource-aware decomposition selection optimizes embedding cost significantly and increases request acceptance ratio unlike in the scenarios where service decompositions were selected randomly. At the same time, this work does not consider the changing traffic demand to determine the number of VNF instances to utilize resources efficiently.

Bari *et al.* (2015a) propose nf.io, a VNF management and orchestration model that uses Linux file system as northbound API to implement a number of tasks such placement, chain composition and monitoring of VNFs. This proposal defines the linux file system used and features that can be exploited to implement some certain tasks; however, the proposed method does not address the case where multiple instances are instantiated nor determines the appropriate placement of VNFs in the infrastrcture.

Wang *et al.* (2016) address the problem of online deployment of scalable multiple VNF instances in order to process the fluctuating traffic rate received at VNFs to attain the minimum cost of provisioning resources. The authors propose two algorithms. One for the single service

chain and the other is for multiple service chains. The first algorithm is divided into two phases. The first phase is called pre-plan phase. In this phase, all VNF instances are prohibited from migrating between servers. In the second phase, the algorithm adapts to the randomized ski-rental algorithm to reach the optimal ratio. What distinguishes the second proposed algorithm from the first one is that it considers the existence of maximum input flow rate vector, which determines the maximum number of VNF instances to be deployed for each service chain. The results show that the randomized algorithm (Algorithm 1) can reduce up to 70% of the operational cost comparing to Receding Horizon Control (RHC) (Lin *et al.* (2012)). However, The author claims that the less number of instances deployed the more cost reduction can be achieved. Although this claim is true but is not realistic as the number of instances may scale up/down based on the demand.

Ghaznavi *et al.* (2015a) address Elastic Virtual Network Function Placement (EVNFP) problem, which involves server resources and bandwidth consumption as well as scaling in/out of VNFs and reassignment of VNF instances. To solve this problem, the authors introduce Simple Lazy Facility Location (SLFL), which aims at optimizing the placement VNF by minimizing installation, transportation, reassignment, and migration costs. The authors compare their proposed algorithm with the first-fit (Xia, Ming and Shirazipour, Meral and Zhang, Ying and Green, Howard and Takacs, Attila (2015)) and random placement (Carpio *et al.* (2017)) techniques. The results show that the proposed algorithm performs better in terms of acceptance ratio and operational cost compared with the two other algorithms. This work does not predict the future traffic load and hence, it is difficult to determine the number of instances needed prior to allocating resources especially if sudden surges occur. Furthermore, the authors in this work have limited their solution to only multiple instances of one VNF-type.

2.1.2 Service chains traffic routing

We hereby demonstrate the most important proposals in regards to service chains traffic routing with an aim of optimizing link bandwidth utilization and minimizing end-to-end delay.

Moens & De Turck (2014) address VNF Placement problem. The problem is mainly focused on VNF placement scenario where the base load is processed by hardware-based middle-boxes, and the remaining load is processed by VNFs when middle-boxes are fully utilized. The authors in this work propose Virtual Network Function Placement (VNF-P) algorithm for hybrid NFV networks resource allocation. The results show that the model works efficiently in small service provider networks in terms of execution speed. However, the authors in this work do not take into account to replace hardware-based middle-boxes by instantiation VNF instances to minimize the operational cost. Furthermore, the authors have not discussed in details how the processing capacity of a middle-box is determined.

Huang *et al.* (2015) tackle flow mingling problem for multiple service chains. The problem in this work is detailed in two sub-problems: Intra-chain contention and inter-chain contention problem. Intra-chain contention problem is basically caused when traffic flow of a single service chain utilizes the same set of links multiple times whereas the inter-chain contention is led to when multiple service chains pass through the same set of links several times. To balance traffic routing among multiple service chains, the authors propose Network-Aware Chains Orchestration (NACHOS) algorithm, which applies linear and dynamic programming in order to maximize network available bandwidth. Moreover, the dynamic programming solves this problem by using backward induction, which records the number of times a service chain passes through the same links. The authors compare NACHOS, with single service chain mechanism (SSC) with different datacenter network sizes to measure the utilization of network resources in terms of number of served users and SLA violations reduction rate. NACHOS shows its efficiency in terms of the acceptance rate and SLA violations reduction rate. However, this work focuses only on rerouting traffic inside data center networks without taking into account the placement of service chains. They also do not consider the instantiation of multiple instances and traffic routing mechanisms between synchronized instances.

Mehraghdam *et al.* (2014) propose a model for formalizing network functions chaining using a context-free language, which contains the order of network functions to be placed. They propose a heuristic that reduces the run-time of the process if there are multiple deployment

requests ordering possibilities. Moreover, the authors formulate an optimization model for the best placement of chained network functions in multiple data centers considering some metrics such as data rate, the number of used nodes, and latency. The results of this work show trade-offs between total remaining data rate, number of used nodes, and latency. Additionally, it shows the potential of finding a placement that optimizes the three metrics if there are enough resources. However, this work does not consider chaining NFs dynamically based on tenants' demand.

Beck & Botero (2015) address placement and chaining of VNFs and introduce CoordVNF to solve this problem. The proposed solution aims at reducing the utilization over links. It adapts to the changing conditions of the infrastructure and attempts to find a feasible chaining options for VNFs. VNF instances are only considered in some use cases when vDPI splits the traffic to TCP and non-TCP traffic. Dynamic allocation of resources and future traffic demand is not taken into consideration in this work. CoordVNF adopts backtracking concept where it repeatedly attempts to find VNF instances assignment options. If for some reason the algorithms is incapable of finding feasible solutions for embedding, it iteratively attempts to embed other chaining options. This proposal considers multiple instances only when a virtual deep packet inspection (vDPI) splits the traffic to TCP and non-TCP traffic.

In (Vizarreta *et al.* (2017)), the authors address the problem of VNF performance compared to the hardware-based middleboxes by trying to guarantee the minimum data rate, maximum end-to-end latency, port availability, and packet loss avoidance. The authors propose an ILP that finds placement of VNFs while guaranteeing SLA requirements (i.e. minimum bit rate and service availability). The main objective of this proposal is to minimize resources cost while taking into account the bandwidth, propagation and processing delays, as well as service availability. The authors propose three greedy heuristics. The first algorithm finds a feasible embedding for the service chain by exploring the paths and subsequently selects the QoS-constrained path. The second algorithm evaluates the candidate nodes into which the VNFs are embedded, and selects the node with the lowest additional cost (i.e. cost of license installation). The third algorithm finds the shortest path between the source and the destination of the

service chain while taking into account QoS constraints. Furthermore, the authors evaluate the performance of the proposed heuristic with a baseline algorithm which barely finds the shortest QoS-constrained paths between the source and destination. Results show that service availability can be guaranteed, but the risk of SLA violation can still be high when service availability is neglected.

2.1.3 VNF replication, synchronization, and migration

In this section, we go through the proposals that mainly focus on VNF replication in an attempt to improve service availability and reduce network costs (Carpio *et al.* (2017)), and VNF state migration as in (Peuster & Karl (2016)) and (Nobach *et al.* (2016)).

Carpio *et al.* (2017) formulate the VNF Placement using a mixed linear program (MIP) and compare it with a random fit placement algorithm (RFPA) to evaluate VNF placement considering the replications between VNF instances. VNF replicas are allocated on a cluster of servers within data centers. For scalability, they propose a genetic algorithm (GA) that finds the sub-optimal solution. The proposed GA consists of three phases: In the first phase, it finds the admissible paths and calculates the cost of the links where in the second phase, resources are allocated for the original VNFs. The algorithm, in the third phase, finds the nearest feasible placement of the VNF such that network costs are minimized. This work mainly focuses on replicating VNF instances; however, it does not take into account the difference in VNF types, which play a major role in the exchanged data rate.

Peuster & Karl (2016) address the problem of automatic migration of VNF state. This VNF state is responsible for replicating status of VNF instances of the same type in the service chain. The authors in this work propose E-State, a management scheme that logically exchanges the internal VNF state by using a distributed memory that is accessible by adjacent VNF instances when traffic assignment changes. For the proposed scheme to exchange VNF state between instances, requesting global state through which a VNF can obtain information from other instances. The authors compare the proposed framework with a centralized management system.

The results show the distributed management scheme outperforms the centralized framework in terms of the number of replicated instances. This work focuses only on state migration between VNF instances, but does not take into account a service chain composed of multiple instances. Furthermore, the number of instances is not determined to meet traffic demand.

Nobach *et al.* (2016) propose SliM, a statelet-based scheme that migrates VNF instances seamlessly to address VNF state migration between a source VNF to a destination instance. They claim that packet duplication mechanism may improve performance of the system. However, such mechanisms incur high additional costs. Their proposed solution, SliM, initiates the destination instance to receive statelets of the snapshot of the source VNF instance, and then sends these statelets over the corresponding stream. The authors evaluate SliM and compare it with a duplication-based model. Results show that SliM performs 3 times better than the duplication-based model in terms of link utilization. On the other hand, the authors in this work consider NAT VNF only among all other VNF types. They also do not take into account the distance between the source and destination instances and the communication cost incurred when available links are highly congested.

2.2 Comparison and Discussion

As illustrated in Table 2.1, the table demonstrates objectives for different approaches among the existing works. We briefly highlight these key differences and compare them to our approach in terms of number of instances required for VNFs, operational cost and synchronization cost reduction as well as service providers' profit maximization.

Our proposed solutions aim to determine the number of VNF instances needed to operate SFCs that meet the high traffic demand, reduce the total operational and synchronization costs taking into account different energy prices and exchanged data between the mapped VNF instances, and increase acceptance rate of mapped requests.

Table 2.1 Our solutions vs. existing solutions

Approach	Objectives				
	Determine Instances No.	**Minimize Operational Cost**	**Minimize Synchronization Cost**	**Maximize Acceptance rate**	**Maximize Profit**
Wang *et al.* (2017)	✓	✓	✗	✗	✗
Mijumbi *et al.* (2015b)	✗	✗	✗	✓	✓
Bari *et al.* (2015b)	✗	✓	✗	✓	✗
Luizelli *et al.* (2015)	✓	✓	✗	✗	✗
Mehraghdam *et al.* (2014)	✓	✗	✗	✗	✗
Sahhaf *et al.* (2015)	✗	✓	✗	✓	✗
Peuster & Karl (2016)	✗	✗	✓	✗	✗
Huang *et al.* (2015)	✗	✓	✗	✓	✗
Ghaznavi *et al.* (2015a)	✗	✓	✗	✓	✗
Our Proposed Solutions	✓	✓	✓	✓	✗

2.3 Conclusion

This chapter has conducted a study for the main approaches that address VNF placement and chaining. The literature has been classified based on the main focus of the work addressing three portions of VNF placement, namely VNF Placement and Chaining costs reduction, VNF Placement and Service Traffic Routing, and VNF Replication, Synchronization, and Migration. In the following chapter, we are going to describe the mathematical formulation of VNF instances placement and chaining and demonstrate the proposed solutions in details.

CHAPTER 3

SOLUTION DESIGN

In this chapter, we formulate the VNF Placement problem using Integer Linear Program (ILP) with an objective of minimizing the operational and synchronization cost of VNF instances. To cope with the complexity of the problem, we further our solution and propose two heuristics that aim at finding an embedding for service requests that minimizes the operational cost while taking into account the synchronization between instances of the same VNF type.

3.1 Problem Formulation

In this section, we formulate the problem of multiple VNF instances placement and chaining with an objective of minimizing VNF provider's operational costs which include instance costs, bandwidth and synchronization costs as well. The problem is formulated as an Integer Linear Program (ILP).

The physical infrastructure consists of multiple POPs located in different regions modeled by a directional graph $G = (M, P)$ where $M = \{0, 1, ..., |M|\}$ denotes the set of POPs and $P = \{(m,n) \in (M \times M) | m \text{ and } n \text{ are directly connected}\}$ is the set of physical links connecting them. Each POP $m \in M$ has a resource capacity C_m expressed in terms of the number of instances that it can host. We assume the size of one instance to be equal to 1 vCPU, 1 GiB of memory and 1 GB of disk. Each physical link $(m,n) \in P$ connecting POPs m and n has a bandwidth capacity B_{mn}.

Similarly, a service function chain is represented as a graph denoted by $V = (I, L)$ where $I = \{0, 1, ..., |I|\}$ is the set of virtual instances in the chain and L denotes the set of virtual links connecting them. Each VNF instance $i \in I$ has a resource requirement of 1 vCPU, 1 GiB of memory, and 1GB of storage. Each virtual link $(i, j) \in L$ has bandwidth requirement b_{ij}. It is worth noting that the endpoints (i.e., sources and destinations) are considered also instances. However, the requirements of such instances are 0 resources and they are constrained to be

Table 3.1 Table of notations

Symbol	Definition
$G = (M, P)$	Graph G consisting of the set of POPs M and set of physical links P
$V = (I, L)$	Graph V consisting of the set of VNF instances I and the set virtual links L
C_m	the available capacity at POP m in terms of number of instances
B_{mn}	Bandwidth capacity of the physical link connecting POPs m and n
$b_{i,j}$	Bandwidth requirement of the virtual link connecting instances i and j
δ_{im}	Deployment costs per unit of time for VNF instance i into POP m
$\Delta_{m,n}$	Bandwidth cost per bandwidth unit in physical link (m, n)
f_{im}	Boolean variable set to 1 if VNF instance i has to be embedded into POP m
s_{ij}	A constant indicates that instances i and j implement the same VNF and hence there is a synchronization cost between i and j
x_{im}	A boolean decision variable indicating whether or not instance i is embedded into POP m
$y_{ij,mn}$	A Boolean decision variable indicating whether or not the virtual link (i, j) is mapped into physical link (m, n)
$\mathbb{O}$	Operational cost
$\mathbb{S}$	Synchronization cost

mapped onto particular physical POPs are stated in the request. We provide more details about this constraint later on in this section.

Furthermore, we define two decision variables. The first one is denoted as $x_{im} \in \{0, 1\}$ and indicates whether or not VNF instance i is embedded into POP m. The second decision variable is denoted as $y_{ij,mn} \in \{0, 1\}$. If $y_{ij,mn} = 1$, this means that the virtual link (i, j) uses the physical link mn. It is worth noting that a virtual link is embedded through a physical path. As a result, several physical links could be used to embed the virtual link. In other words, if $y_{ij,mn} = 1$, the physical link (m, n) is part of the physical path used to embed the virtual link (i, j).

In the following, we provide more details about the different costs incurred when embedding the chain, the objective function and the set of constraints to be satisfied.

- **Synchronization cost:** the synchronization cost is the rate of exchanged data between two VNF instances i and j that implement the same VNF. This cost relies on the number of hops between the synchronized instances, and this cost can be expressed as following:

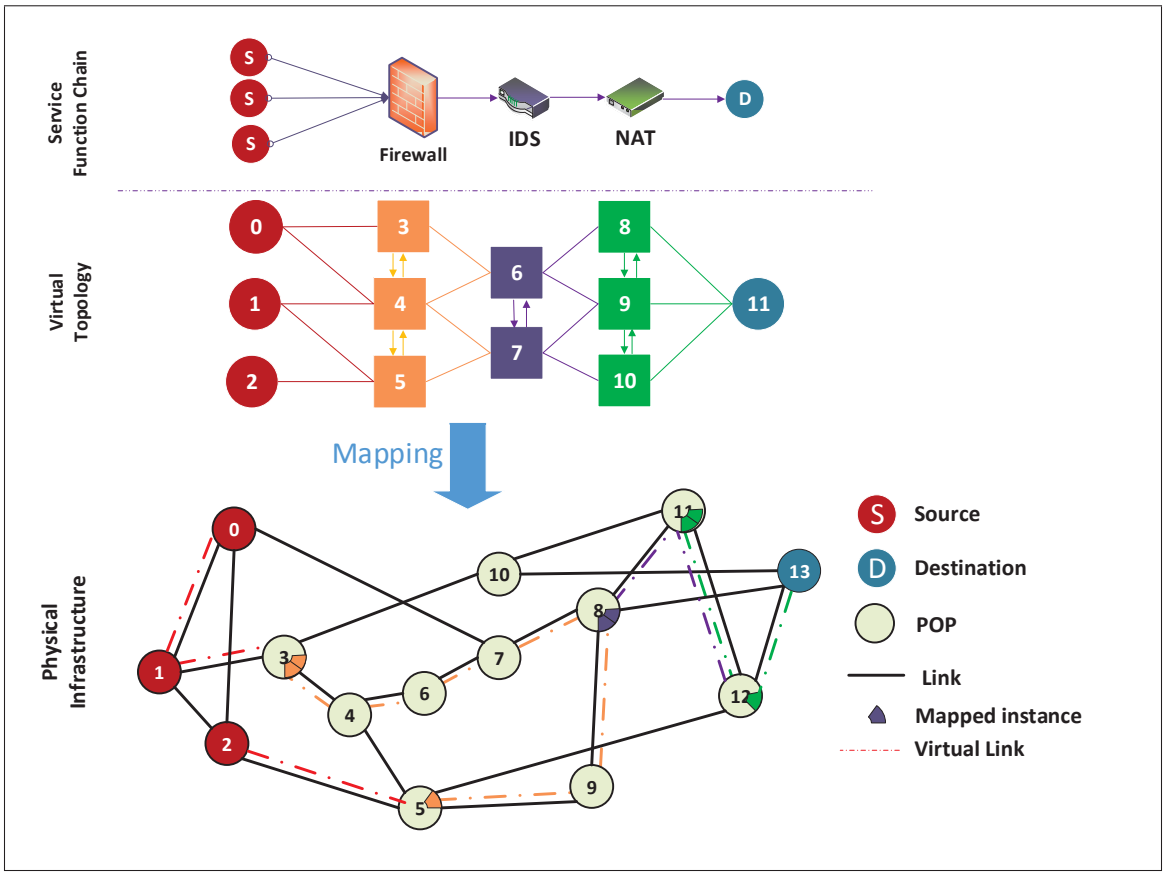


Figure 3.1 VNF Instances Embedding Problem

$$\mathbb{S} = \sum_{(i,j)\in L} \sum_{(m,n)\in P} y_{ij,mn} s_{ij}\, b_{ij}\, \Delta_{mn} \tag{3.1}$$

Where $y_{ij,mn}$ indicates whether or not the physical link (m,n) is used for embedding the virtual link (i,j). s_{ij} denotes whether or not instances i and j implement the same VNF. In this case, there is a synchronization cost computed as $b_{ij}\ \Delta_{mn}$, which is the cost of using bandwidth to exchange synchronization data between i and j.

- **VNF Instance operational cost:** it is the cost of running the VNF instances on the infrastructure. It can be expressed as follows:

$$\mathbb{O} = \sum_{m \in M} \sum_{i \in I} x_{im}\ \delta_{im}\ +\ \sum_{(i,j) \in L} \sum_{(m,n) \in P} y_{ij,mn}\ (1 - s_{ij})\ b_{ij}\ \Delta_{m,n} \tag{3.2}$$

where δ_{im} is the deployment cost (expressed in dollars per unit of time) of VNF instance i into POP m. It is worth noting that δ_{im} varies from one POP to another as it depends on several factors including the electricity price in the POP, the type of the VNF, the license, and the operating system. The first term of the equation (Eq.3.2) represents the total cost of deploying the VNF instances. The second term of the operational costs is the total cost of bandwidth consumed by the virtual links. $\Delta_{m,n}$ denotes the cost in dollars (per bandwidth unit and unit of time) for the physical link (m,n). It varies from one physical link to another.

- **Objective Function:** The objective function when embedding request r aims at minimizing the operational costs $\mathbb{O}$ and synchronization cost of the embedded VNF instances $\mathbb{S}$. This function can be expressed as:

$$J = \min_{\substack{(x_{im})_{i \in I, m \in M} \\ (y_{ij,mn})_{(i,j) \in L, (m,n) \in P}}} (\mathbb{O} + \mathbb{S}) \tag{3.3}$$

- **Endpoint embedding constraint:** this constraint ensures that endpoints instance (sources and destinations) are embedded into the POPs specified in the request. To do so we define the boolean variable f_{im}, which is an input to the ILP, which is equal 1 when the instance i is an endpoint that has to be embedded in POP m. To ensure the endpoint embedding constraints, we have:

$$x_{im} \geq f_{im} \quad \forall m \in M,\ \forall i \in I \tag{3.4}$$

- **Instance embedding constraint:** This constraint ensures that each VNF instance i is embedded one and only once. It can be expressed as:

$$\sum_{m \in M} x_{im} = 1 \qquad \forall i \in I \tag{3.5}$$

- **Resource capacity constraint**: this constraint ensures that any hosting POP has enough resources to host the VNF instances.

$$\sum_{i \in I} x_{im} \leq C_m \qquad \forall m \in M \tag{3.6}$$

where C_m represents the available capacity at POP m.

- **Bandwidth constraint:** we must also ensure that the bandwidth capacity required to embed all virtual links in a physical link does not exceed its available bandwidth. This can be expressed as follows:

$$\sum_{i,j \in L} y_{ij,mn}\, b_{ij} \leq B_{mn} \quad \forall\, (m,n) \in P \tag{3.7}$$

- **Flow conservation constraint:** we must also ensure that the incoming traffic to a POP is equal to its outgoing traffic unless this POP is source or destination. This constraint can be expressed as:

$$\sum_{(n,m) \in P} \sum_{(i,j) \in L} y_{ij,nm}\, b_{ij} - \sum_{(i,j) \in L} x_{jm}\, b_{ij} = \sum_{(m,n) \in P} \sum_{(i,j) \in L} y_{ij,mn}\, b_{ij} - \sum_{(i,j) \in L} x_{im}\, b_{ij} \quad \forall\, m \in M \tag{3.8}$$

The objective function is subject to the previous constraints. The service chain embedding problem is an *NP*-hard problem as it generalizes bin-packing problem; therefore, finding an optimal solution is not viable due to the large number of requests processed in the production environment (Zhani *et al.* (2013b)). Hence, we propose two heuristics in the following section to solve this problem and explore potential solutions.

3.2 Heuristic Solutions

In this section, we discuss the proposed solutions in details. To solve the problem of VNF instances Placement after the maximum processing capacity of each VNF in SFC is determined, we propose two algorithms. The first algorithm, called Least Operational Cost First (LOCF), attempts to minimize the operational cost while taking into consideration the synchronization

cost between each pair of VNF instances when placed either in the same path or different paths. The second algorithm, Least Synchronization Cost First (LSCF), prioritizes the objective of minimizing synchronization cost between instances for each request by embedding service requests including multiple VNF instances of the same type into the same POP. In the following section, we discuss further details of the two algorithms.

3.2.1 Least Operational Cost First (LOCF)

LOCF firstly parses all the paths between the source of the request is received and the destination. Our proposed solution aims at finding the most appropriate placement for an SFC composed of multiple instances such that it minimizes the operational cost while considering the synchronization between pairs of VNF instances.

Algorithm 3.1 Least Operational Cost First (LOCF)

```
1  At the receipt of the request at m
2  Find k S_Paths using Mod-DFS
3  Sort the paths based on the operational cost using Eq. 3.2
4  while S_Path ∈ S_Paths[ ] can embed the request do
5      if (inst(req)) ≤ (inst(S_Path)) then
6          Embed the request
7          Calc Sync using Eq. 3.1 & Update
8      end
9      Determine the number of layers for the request
10     For all layers
11     if (inst(layer) ≤ (inst(Path))) then
12         Embed inst(layer)
13         Calc Sync & Update
14     else
15         Embed inst(Path)
16         Calc Sync & Update
17     end
18 end
```

The LOCF algorithm starts by running a modified version of Depth-First Search (DFS) algorithm (Robert Tarjan (1972)) that is called Mod-DFS, which finds all paths from the source to

the destination of the request. Mod-DFS basically sorts the paths based on the operational cost from the least to the highest and then selects the path with thes least operational cost. Each path varies in terms of number of POPs and the capacity of these POPs. DFS starts the search for the paths from the source to the destination. It marks the current POP as visited and then it checks all the outgoing links connecting the current POP with all adjacent POPs. It examines whether each visited POP is the desired destination. If not, it progressively searches deeper until the destination POP is found. If it encounters a POP that is not connected with any more POPs, it backtracks going back to the most recent visited POP that it has not finished exploring.

At the receipt of the request, LOCF parses all the paths from the source to the destination. It checks whether or not the path has enough capacity to embed the whole request. In case the POPs across the selected path have sufficient resources to accommodate the entire request, then the embedding takes a place and the POPs' resources are updated. If the whole request cannot be embedded into the POP with the least operational cost, it splits the request into layers where each layer consists of a single instance of each VNF such that an embedding for an instance of each VNF i in the chain becomes feasible. In other words, the algorithm looks into the path and tests whether the current path has enough resources to embed at least one instance of a layer. If the whole request cannot be embedded; yet instances of one layer can be mapped, then the algorithm tries to find a feasible embedding for other layers of the request following the same way taking into account synchronization cost. These steps will be repeated until all instances of the request are embedded.

Following this, we discuss the second algorithm, which attempts to minimize the synchronization cost by allocating resources of POPs whose capacity is the highest in order to embed as many instances of the same type into these POPs.

3.2.2 Least Synchronization Cost First (LSCF)

This algorithm focuses on finding k paths from the source to the destination using DFS algorithm. Unlike, LOCF, the LSCF algorithm sorts the available source-destination paths based on their available capacity (in terms of number of instances) from the highest to the lowest. The goal behind that is to embed as many instances of the same VNF type near each other in an attempt to minimize the synchronization cost. If two or more paths with the same total capacity have been found, then the algorithm chooses the path with the least operational cost such that synchronization cost and operational cost are minimized.

Algorithm 3.2 Least Synchronization Cost First (LSCF)

```
1  At the receipt of request s
2  Parse and sort all k Paths[C_m]^highest between m and n //From the highest to the lowest
3  Select the path at which m with C_m^highest resides
4  while (m≤n) do
5      if (I_r ≤ I ∈ mn) then
6          Embed m←I_r
7          Calc OpCost with Eq. 3.2 & Update
8      end
9      Find m with SyncCost_least
10     if ((m==SyncCost_least)>1) then
11         if (I_r≤C_m) & (C_mrem ≥ I_r) then
12             Embed m←I_r
13             Calc OpCost with Eq. 3.2 & Update
14         else
15             Select m with SyncCost_least
16             Embed & Update
17         end
18     end
19 end
```

At the receipt of the request , LSCF starts by exploring k paths from the source to the destination in an attempt to find the POP m whose capacity is the highest in order to embed as many instances of the request as possible in that POP. If the POP is capable of accommodating a request, an embedding takes a place. If not, the algorithm examines whether the POPs in the

current path can embed a request such that synchronization is minimized. This is done by trying to embed VNF instances of the same type into the same POP. For example, we have a request composed of 5 instances, 2 instances of firewall, 1 IDS instance, and 2 NAT instances and there are two POPs across the path to the destination. Let us say the first POP can accommodate 4 instances and the following POP can accommodate 3 instances. The algorithm will embed the instances of firewall VNF and the IDS instance into the first available POP. Since the remaining capacity in first POP (i.e., 1 instance) is not sufficient to embed the 2 NAT instances, both instances will be placed in the second POP to avoid any synchronization cost.

3.3 Conclusion

In this chapter, we have presented the solution design, which consists of mathematical model to solve VNF placement problem using ILP and two heuristics that minimize the operational cost and synchronization cost. In the next chapter, we demonstrate and analyze Amazon EC2 VM insatnce pricing based on different resource capacity. We then evaluate the performance of three VNFs, namely Firewall, IDS, and NAT using different VM configuration. Finally, we demonstrate the results of our two proposed heuristics at the end of the chapter.

CHAPTER 4

EXPERIMENTAL AND SIMULATION RESULTS

4.1 Introduction

In this chapter, we present the experimental results and simulation results. Through experimental results, we evaluate the performance of three VNFs, namely, Firewall, IDS, and NAT using Shorewall as a firewall (Shorewall (2017)), Snort as an IDS (Snort (2017)), and Ubuntu-based NAT. We measure and compare the three VNFs using different VM configurations (Amazon (2017)) in terms of packet arrival rate, resources (CPU, memory, and storage), and packet loss rate in order to determine the maximum processing capacity of each VNF. In the simulation results, we evaluate the performance of the two proposed algorithms in terms of acceptance rate, POPs utilization, operational cost, as well as the embedding.

4.2 Analysis of VNF Performance and price

4.2.1 Price versus amount of resources

In this section, we analyze the prices of Amazon EC2 instances versus their size in terms of vCPU and memory. We then study the performance of different types of VNF instances using such instances in terms of a number of critical metrics: packet arrival rate, CPU utilization and packet loss rate.

The cost of operating Amazon EC2 instances (Amazon (2017)) is mainly reliant on their configuration (see Table 4.1), i.e. the amount of virtual resources (i.e. vCPU and memory) allocated to run these instances.

According to Amazon EC2 pricing list (Amazon (2017)), the lowest instance configuration is `t2.micro`, which operates with the least resources in terms of CPU and memory (i.e. 1 vCPU core and 1 GiB of memory). The price to run a single instance of `t2.micro`, for example,

Table 4.1 Excerpt of the configuration and price of the EC2 instances, as of November 2017.

Instance Configuration	vCPU Cores	Memory (GiB)	Price/hour ($)
t2.micro	1	1	$0.012
t2.small	1	2	$0.023
t2.medium	2	4	$0.047
t2.xlarge	4	16	$0.188

is $0.012 per hour whereas operating an instance of `t2.small` type, which is allocated a single vCPU and 2GiB RAM, is almost double the hourly rate at $0.023. This also means that having two `t2.micro` instances would cost a tenant $0.024 per hour. However, the tenant will have twice the processing capacity of a single `t2.small`. In other words, two `t2.micro` instances would *in theory* provide better performance than a single `t2.small` instance with merely the same price.

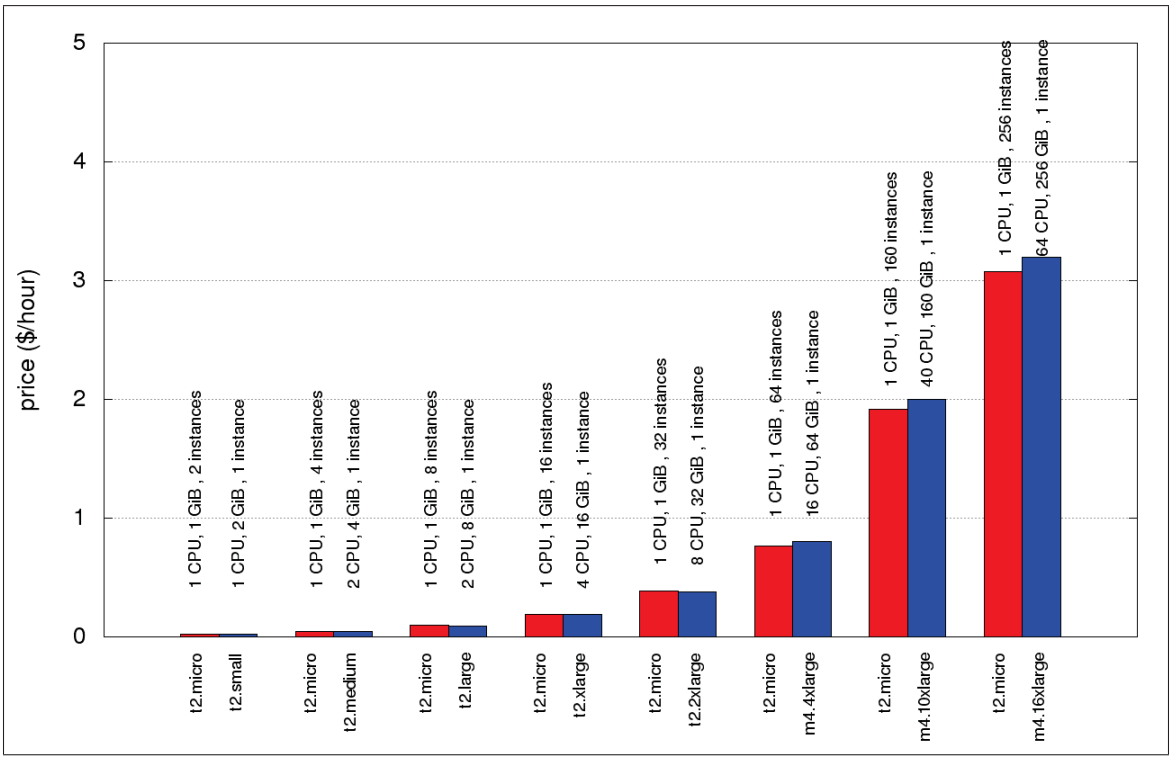


Figure 4.1 Instance capacity versus Price

To better analyze whether this holds true for other types of EC2 instances, we evaluate how many `t2.micro` instances it would take to provide the same amount of resources of larger EC2 instance types, and how they are compared in terms of price.

Figure 4.1 shows the cost of each EC2 instance as well as the number and price of the `t2.micro` instances that would provide an equivalent amount of resources in terms of CPU and memory. For example, the largest EC2 instance, `m4.16xlarge`, provides 64 vCPU cores and 256 GiB, and costs $3.2/hour. The same amount of memory could be provided by 256 `t2.micro` instances with a cost of $3.1/hour, but with significantly more vCPU cores (256 compared to 64). The same observation holds for all other types of instances suggesting that using smaller instances (i.e. `t2.mirco`) would, theoretically, provide reduced costs and increased performance especially in terms of CPU horsepower.

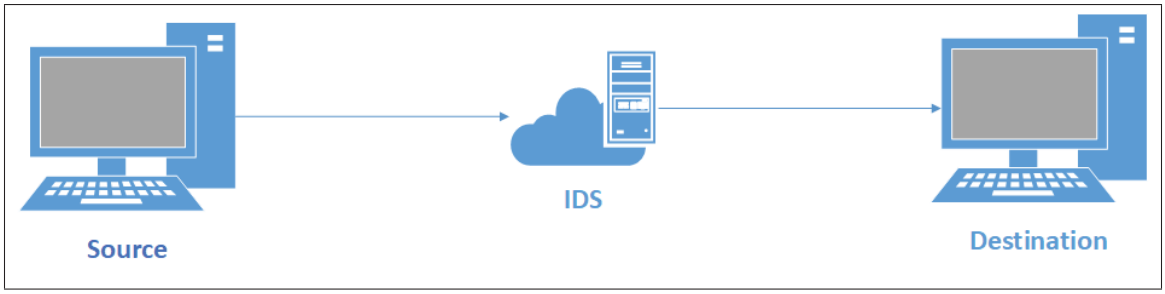


Figure 4.2 Simple Service Request

To evaluate the performance of VNFs, we install a software (i.e. Firewall, IDS, or NAT) on top of a t2.micro instance and generate traffic from the source to the destination node passing by a VNF to form a simple SFC as depicted in (Figure 4.2). While processing traffic, we examine the processing capacity of a VNF such that the physical resources are fully utilized to determine its maximum capacity measuring packet arrival rate, resources (CPU, memory, and disk) utilization, and packet loss rate. The results of these experiments will be demonstrated later on in this chapter.

4.2.2 Amount of resources versus VNF performance

In the previous subsection, we discussed VM prices according to `Amazon EC2` (Amazon (2017)) compared to the amount of resources. In this section, we evaluate the performance of three VNFs namely a firewall, an IDS, and a NAT in this subsection. We ran Shorewall as a firewall (Shorewall (2017)), Snort as a network IDS (Snort (2017)), and the Ubuntu-based NAT.

4.2.2.1 Experimental Setup

We start the evaluation by generating UDP traffic between a pair of instances, where the VNF is installed inline to face the incoming traffic, process it, and then forward it to the destination. We ran the same experiment to examine the processing capacity of all VNFs.

We generated traffic using `iperf` (iPerf (2017)), with this traffic increasing linearly with time. The purpose of increasing traffic load is to measure the CPU, memory, and disk utilization as well as packet loss for a VNF instance for different packet arrival rates. By doing this, we determine the highest packet processing capacity that each VNF can process per second in a particular instance.

4.2.2.2 Micro instance processing capacity

As previously highlighted, the `t2.micro` instance offers the most cost-effective access to VNF-hosting cloud instances. Thus, we look at how capable such instance is at hosting VNFs. For each VNF, we measure CPU, memory and disk utilization as well as packet loss as the received traffic increases over time. The results are depicted in Figures 4.3 through 4.5. For the firewall VNF running on a `t2.micro` instance, it becomes overloaded and rejects packets from around 11,000 packets per second as shown in Figure (4.3(a)). This corresponds to 90% CPU and memory utilization as depicted in Figure (4.3(b)). We also see in Figure 4.4 that an IDS running on a `t2.micro` instance behaves differently from a firewall VNF on the same instance. Packet losses start to occur when more than 18,000 packets start to arrive per second

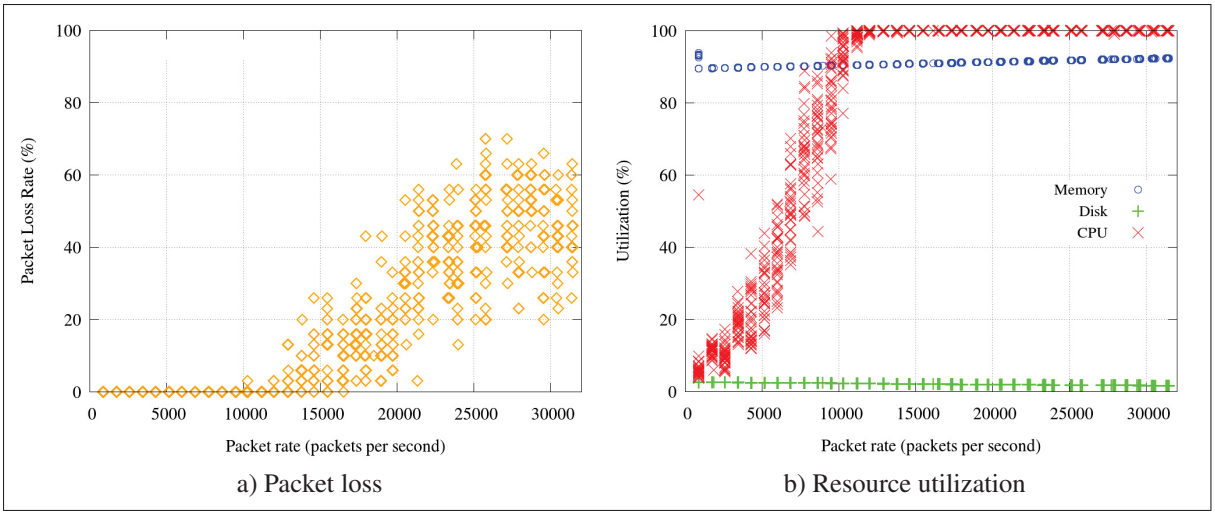


Figure 4.3 Firewall

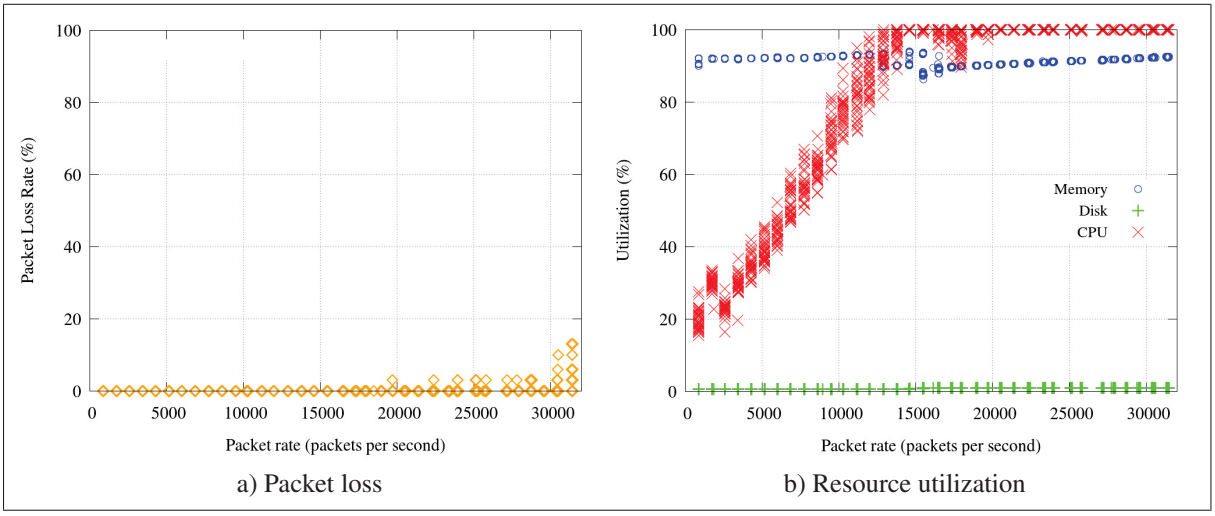
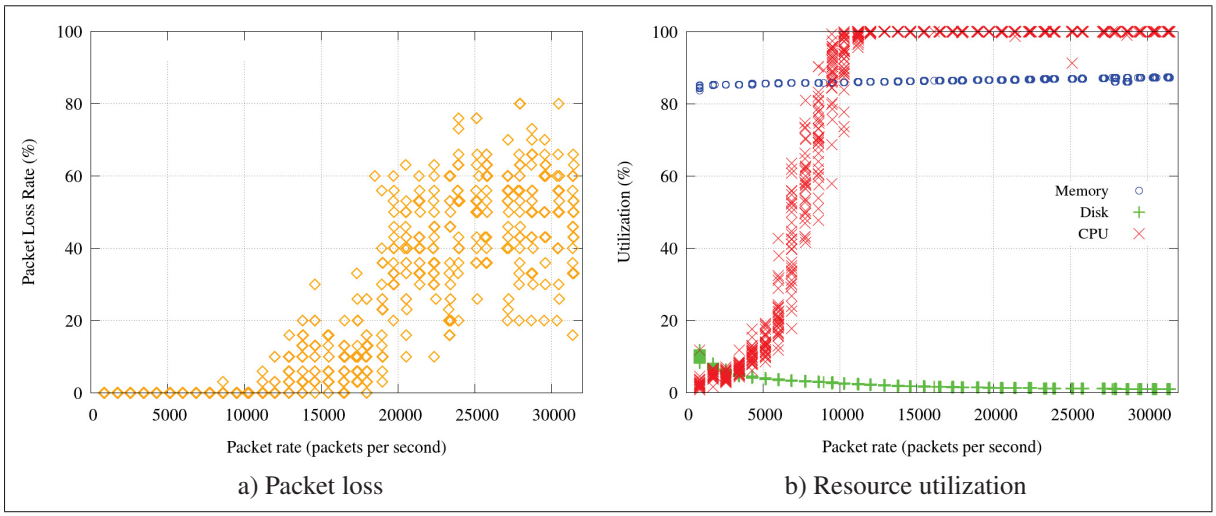


Figure 4.4 IDS

as in Figure (4.4(a)). This corresponds to 100% and 90% of CPU and memory utilization as can be seen in Figure (4.4(b)), respectively. Finally, the results for the NAT are reported in 4.5. We note that the performance of the `t2.micro`-hosted NAT starts to deteriorate around 9,000 packets per second arrival rate as in Figure (4.5(a)). This corresponds to 90% CPU and memory utilization as shown in Figure (4.5(b)).



a) Packet loss b) Resource utilization

Figure 4.5 NAT

4.2.2.3 Amount of resources versus performance

To compare the performance of instances of different sizes, we ran the same experiments for different instance types. For each of them, we identify the peak packet processing capacity (packets per second) as the maximum packet processing rate of the VNF when the CPU utilization reaches 90% or the packet loss reaches 10%, whichever first.

Figure 4.6 summarizes the results and compares the packet processing capacity of four types of instances, `t2.micro`, `t2.small`, `t2.medium` and `t2.xlarge` for the three VNFs. The first observation is that the packet processing capacity may significantly vary depending on the type of the network function deployed in the instance. For example, the figure shows that the `t2.micro` can process as high as 11,000 packets/second when used to host a firewall, 13,000 packets/second for the IDS, and as low as 9,000 packets/second for the NAT.

We also notice that increasing the size of the instance results in a higher processing capacity. However, surprisingly, the increase in processing capacity is not proportional to the number of CPU cores, nor to its price. For instance, int he case of a NAT Figure (4.6), a `t2.micro` instance (1 CPU, 1 GiB, $0.012/hour) can process 9,000 packets/second whereas a `t2.xlarge` (4 CPU, 16 GiB, $0.188/hour) can process 26,000 packets/second; only an 188% increase for

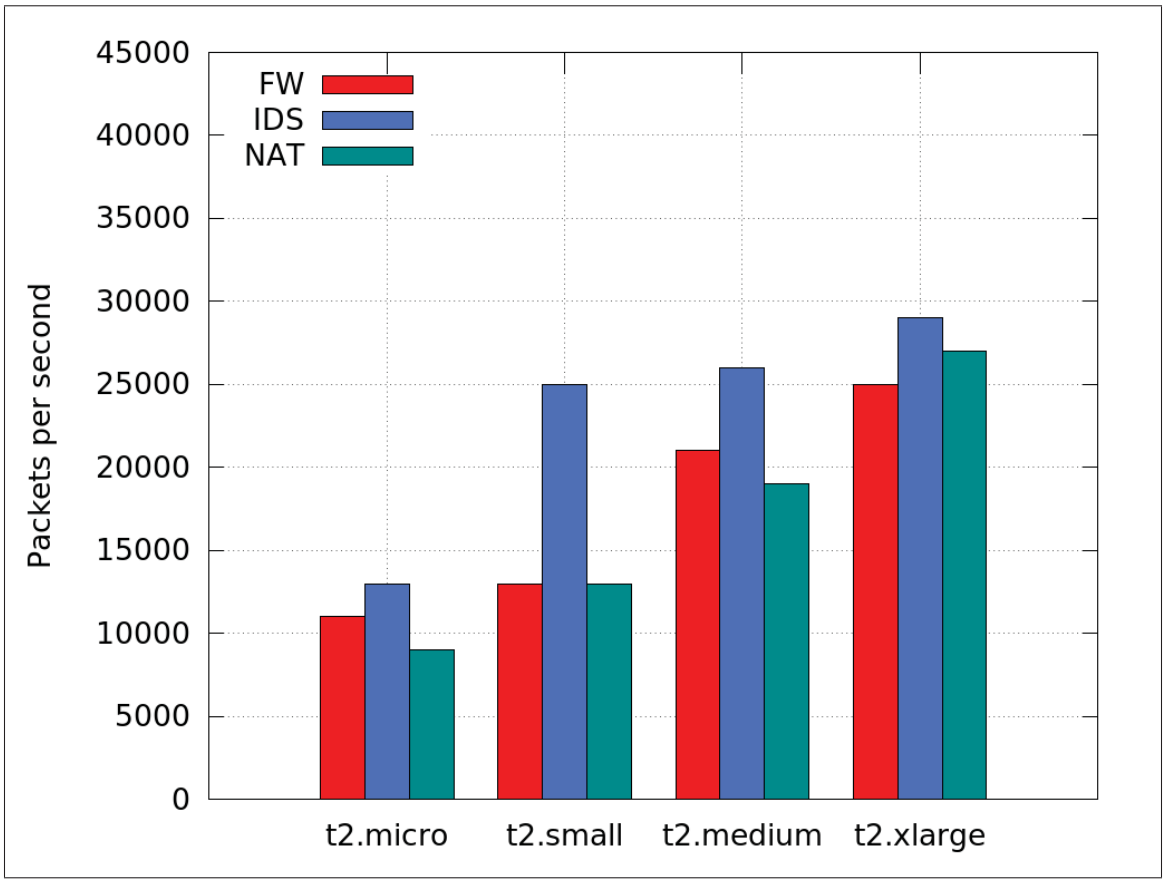


Figure 4.6 VNF peak packet service rate per instance type, as dictated by CPU utilization of 90% or packet loss of 10%, whichever first

more than 15x the price. In other words, even though `t2.xlarge` has 4 times more CPUs and 16 times more memory than `t2.micro`, it can process only 2.8 times more packets than a `t2.micro` instance. This suggests that deploying several `t2.micro` instance provides better processing performance. Finally, recall that the price of 16 `t2.micro` instances is less than a single `t2.xlarge` Figure (4.1), we can hence conclude that `t2.micro` instances are also more appealing in terms of cost-effectiveness.

4.2.2.4 Discussion

Abstracting from the above observations, we extract the following outcomes.

- **Input packet rate is crucial; CPU is the bottleneck**

  The incoming packet rate is the determining factor in the performance of VNFs. Consequently, anticipating such packet rate and its variance is a key element of dimensioning VNF embedding across any network.

  Based on the results plotted in Figures 4.3 and 4.5, VNF packet processing rates could deteriorate due to increased contention on the virtual CPU resource. This is the case for all tested VNF types. Of course, an interesting avenue of future work would be to ascertain if this holds for a wider range of VNFs.

- **Not all VNFs are the same**

  Every deployment of a VNF has its own performance 'breaking point', which varies across VNF types and we conjecture that it would vary also between different implementations of the same VNF. We observed that a low-demand VNF such as an IDS is easily hosted on a very modest cloud instance such as the EC2 `t2.micro`. However, to obtain similar packet rate capacity for a slightly more demanding VNF, such as a firewall, it requires hosting on a higher specification instance (in this case, `t2.medium`).

  We can hence conclude that the type of the VNF and its implementation (in other words, the software running on the instance) will have a major impact on the packet processing capacity of the VNF. It is therefore a major challenge to write customized code that can optimize the VNF operations and further leverage the resources available in the instance where the network function is running.

- **Instance specification is a loose indicator of performance**

  This observation has already been made by different research groups using both generic benchmarks and specific application profiles. This has been a motivator for our study, and indeed we have verified that such variance does exist even for NFV workloads. We found also the performance is not proportional to the amount of CPU and memory allocated for the VNF. This makes selecting the right instance based on the amount of the instance resources not accurate enough. As a result, the instance specification is only a loose indicator

of its performance. It is therefore of utmost importance to test in advance the VNF when running on a particular instance, and evaluate its packet processing capacity as it may vary depending on the instance type.

To conclude, we can say that current cloud provider offering models that provide generic VM instances with predefined resource capacity (i.e. CPU, memory and disk) are not perfectly suitable for VNFs. A better offering model would be to provide VNFs with a well-defined network function software (software name and version) and an average packet processing capacity (in terms of packets per second).

- **Micro instances are more cost-effective**

  The obtained results clearly show that, compared to large instances, deploying multiple micro instances provides not only much more resources (in terms of CPU, memory and disk), but also much more packet processing capacity with much less price. By combining results of 4.1 and 4.6, we can compare the performance, amount of resources and prices for a single `t2.xlarge` instance and 16 `t2.micro` instances. As shown in Table 4.2, for the same price, we can provision 16 `t2.micro` instances that provides 6 times the packet processing capacity of a single `t2.xlarge`.

Table 4.2 Firewall performance using a single t2.xlarge instance compared to 16 t2.micro instances

Instance Type	**vCPU Cores**	**Memory (GiB)**	**Price (US$/hour)**	**Packet Processing Capacity (pckts/s)**
t2.xlarge	4	16	0.1856	26500
t2.micro	16	16	0.1856	144,000

In the following section, we will illustrate the main simulation results in which the two proposed heuristics are evaluated in terms of the average acceptance rate, POPs utilization, operational cost, and synchronization cost.

4.3 Evaluation of the embedding solutions

In this section, we compare the performance of the proposed algorithms in terms of average acceptance rate, POPs utilization, operational cost, and embedding cost when the traffic load $\rho = \lambda/\mu$ is increasing, where λ is the arrival rate and μ is the service time or request lifetime.

4.3.1 Simulation setup

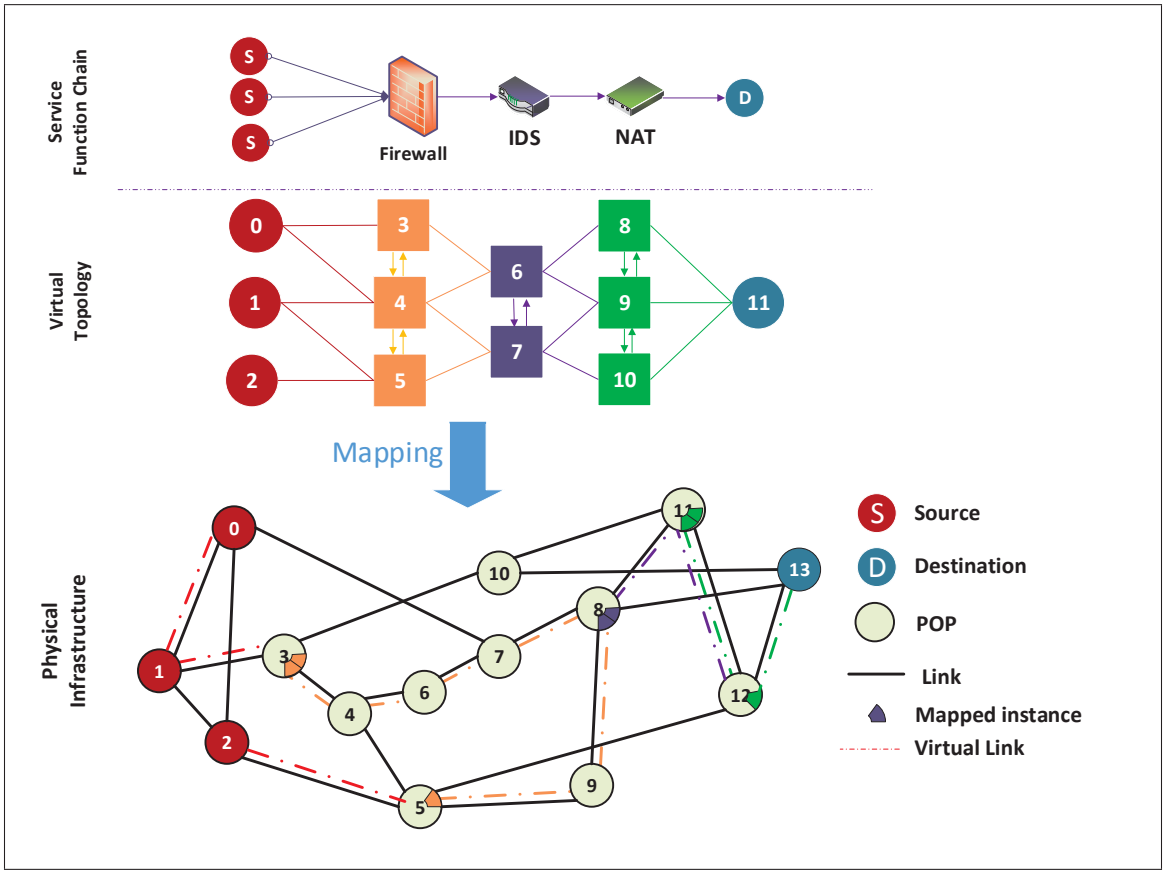
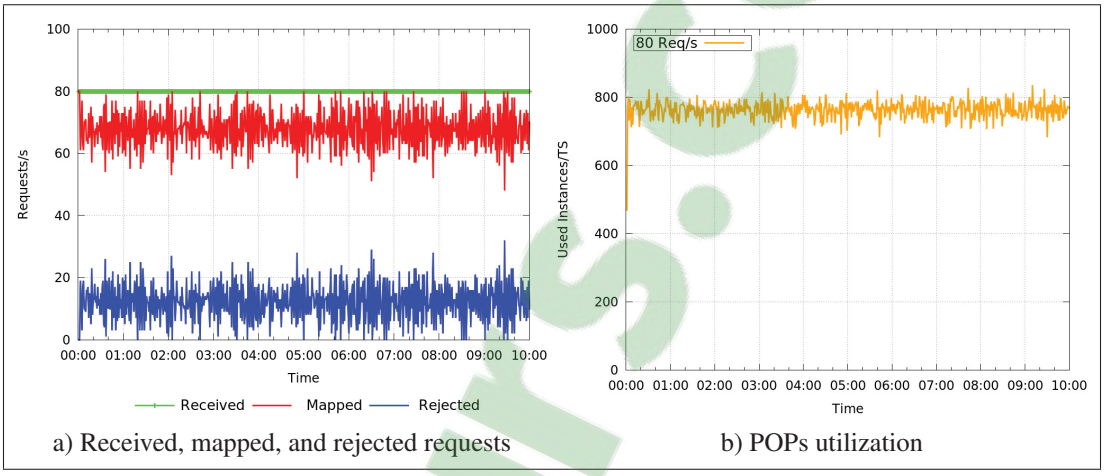


Figure 4.7 VNF instances embedding in the NSFNet topology

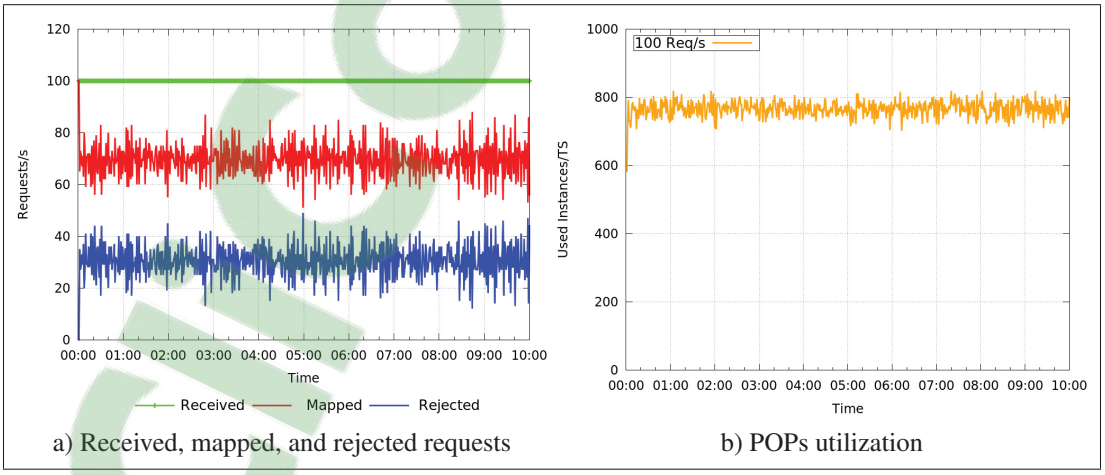
In order to evaluate the proposed solutions, we have developed a C-based simulator that simulates the entire environment consisting an infrastructure that is composed of POPs. We have considered NSFNET topology shown in Figure 4.7 with 14 POPs connected by 21 physical links.

Furthermore, service requests are generated with an average rate of (20,40,60,80,100) requests per second with a random demand between 4000-6000 packets per second. Each request has a random number of VNF instances and random lifetime varies between 4 to 6 seconds. Since electricity prices are different at each POP facility depending on the region it is located, the embedding price for each request is different and ranges between 0.85 to 4.77 dollars per hour according to VM pricing list proposed by Amazon EC2 (Amazon (2017)).



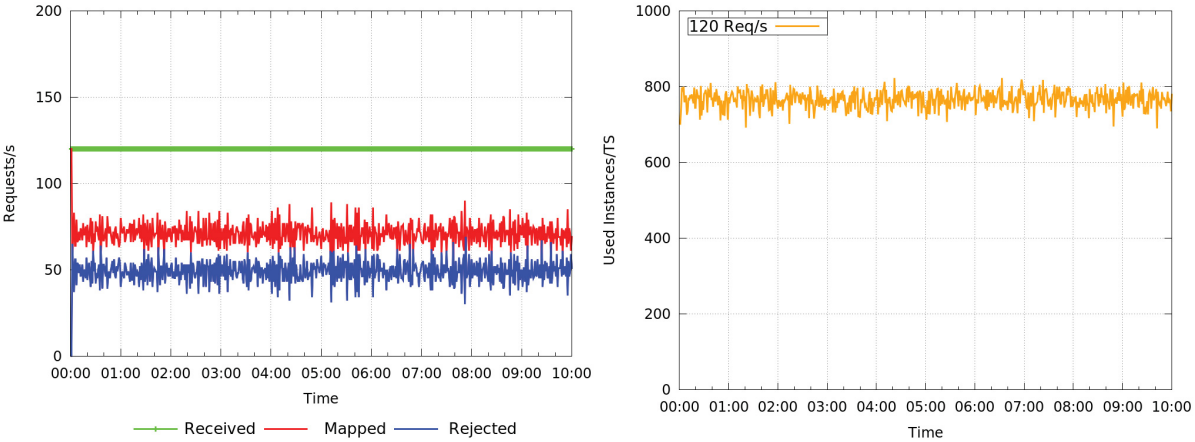
a) Received, mapped, and rejected requests b) POPs utilization

Figure 4.8 80 Requests/s



a) Received, mapped, and rejected requests b) POPs utilization

Figure 4.9 100 Requests/s



a) Received, mapped, and rejected requests

b) POPs utilization

Figure 4.10 120 Requests/s

The capacity of each POP is defined as the number of available micro instances at each facility, which varies between [40-110] instances. Each VNF has a standard CPU, memory, and storage requirements randomly defined between 0.01 and 0.04.

All simulations are conducted on a server with Intel Core i7 3.33 Ghz CPU and 32 GB of RAM running Ubuntu 14.04.LTS 64-bit OS.

We have observed that when 20, 40, and/or 60 requests are received at the source POP, all requests are mapped into the POPs across the paths from the source to the destination. This is an indication that the infrastructure is barely utilized. It is also worth mentioning that LOCF utilizes approximately 24.23%, 48,84%, and 72.6% of the infrastructure when the arrival rate λ of requests is 20, 40, and 60.

Figures (4.8),(4.9),(4.8) depict the received requests, mapped and rejected for different arrival rates λ as well as the total number of instances embedded into the system over time. Figures (4.8(a)) and (4.8(b)) show the number of requests received, mapped, and rejected as well as the used instances for embedding. We can see that around 70 out of 80 requests per second are mapped into the POPs. In Figure 4.8(b)) specifically, we can see that the average used instances is around 762 mapped instances. Similarly, in Figures (4.9(b)) and (4.10(b)) although

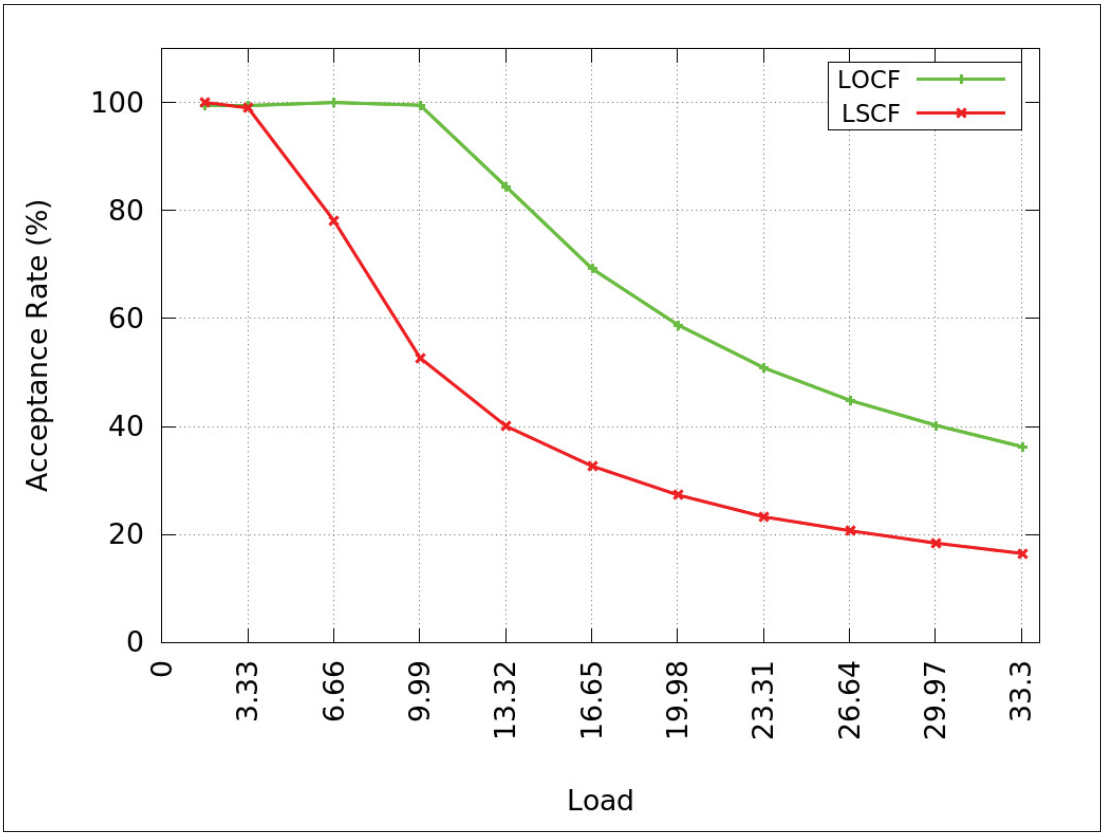


Figure 4.11 Average acceptance rate

the arrival rate $\lambda = (100, 120)$ as in the Figures (4.9(a)) and (4.10(a)). In the following section, we demonstrate a comparison between performance metrics including acceptance rate, POPs utilization, operational cost, and synchronization cost.

4.3.2 LOCF versus LSCF

In this section, we present the results comparing the two algorithms in terms of acceptance rate, POPs utilization, operational cost, and synchronization cost. Since LOCF aims at minimizing the operational cost while considering synchronization, and LSCF tires to embed instances as many of the same VNF type as possible in an attempt to reduce the synchronization cost between instances, we hereby summarize the results of ten experiments using different loads.

- **Acceptance Rate**

This represents the rate at which requests are mapped into the system upon its arrival at the source POP. The load ρ is shown in the horizental axis and is expressed as: $\rho = \frac{\lambda}{\mu}$ where λ denotes the arrival rate and μ denotes the average service time or request lifetime (time for which each request remains in the system).

Figure 4.11 shows a comparison between the performance of the two algorithms LOCF and LSCF. We can see here that LOCF outperforms LSCF when the rate of requests received is 10, 20, 40, or 60 requests per second then it degrades. LOCF accepts up to 46.82% more requests than LSCF. However, this difference decreases to around 20% when the load increases.

- **POPs Utilization**

The POPs utilization describes the resource utilization of these facilities, which is referred to as the number of instances that are currently embedded in the system. Resource utilization is highly dependent on the load of requests being processed and the lifetime of each request in the system.

Figure 4.12 depicts a comparison between LOCF and LSCF. Despite the fact that both algorithms cannot saturate the whole infrastructure, LSCF shows more resource utilization than LOCF. The reason why the proposed algorithms are incapable of using the full capacity of the infrastructure is that DFS tries to find POPs that connect the source with its destination. Some POPs may not come across the two end-points; therefore, these POPs cannot be considered for any potential embedding of VNF instances.

- **Operational Cost**

This represents the cost of running instances for a period of time. This cost varies depending on the POP facility's location and the duration of the service. In the following figure, we demonstrate the operational cost for different loads using LOCF and LSCF. Figure 4.13 illustrate the performance of the two proposed algorithms (LOCF and LSCF) and shows that LSCF uses higher operational cost than LOCF. This cost is reflected by the high embedded instances as shown in the previous Figure 4.12.

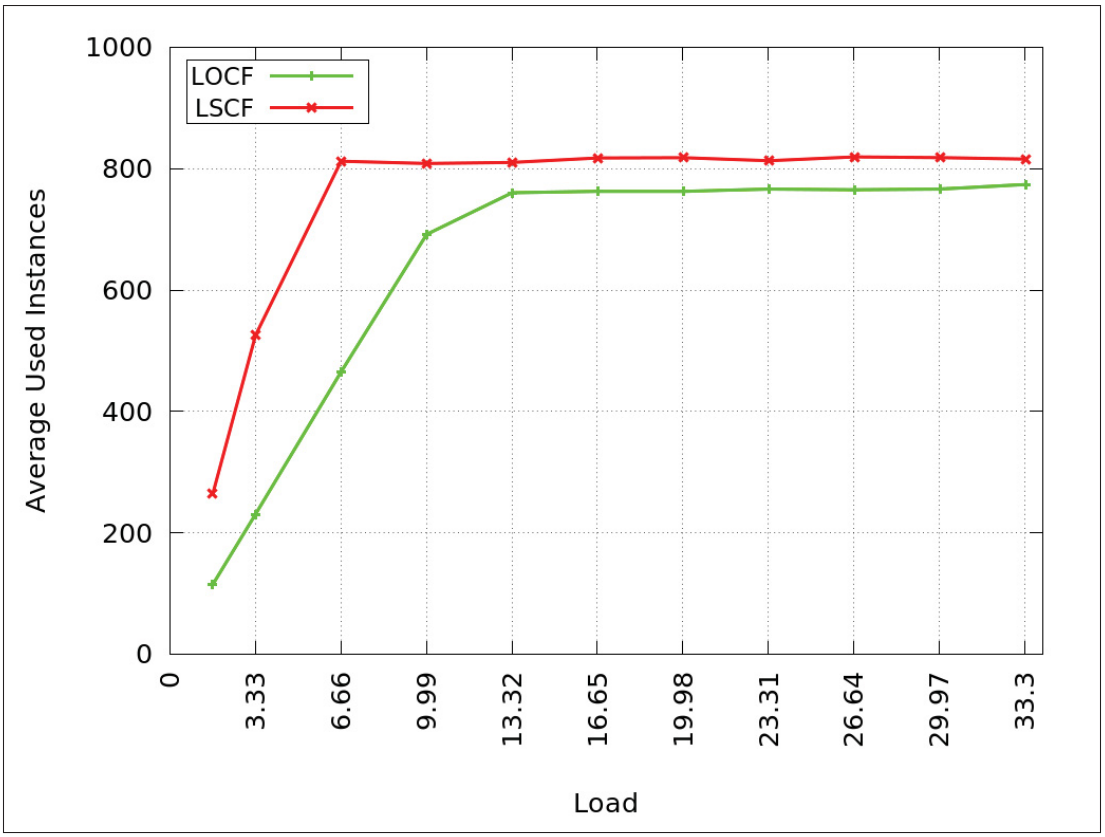


Figure 4.12 POPs utilization

- **Synchronization Cost**

  This demonstrates the synchronization cost for different loads using LOCF and LSCF. The synchronization cost is the cost of exchanging data between each pair of instances of the same type for a period of time (i.e. SFC lifetime). This cost is computed as the number of hops between per time slot the POPs hosting the instances of the same VNF.

  Figure 4.14 shows that LSCF, which mainly focuses on reducing synchronization cost and embedding VNF instances of requests into the POPs whose capacity is the highest, minimizes the synchronization cost compared to LOCF especially when the load increases.

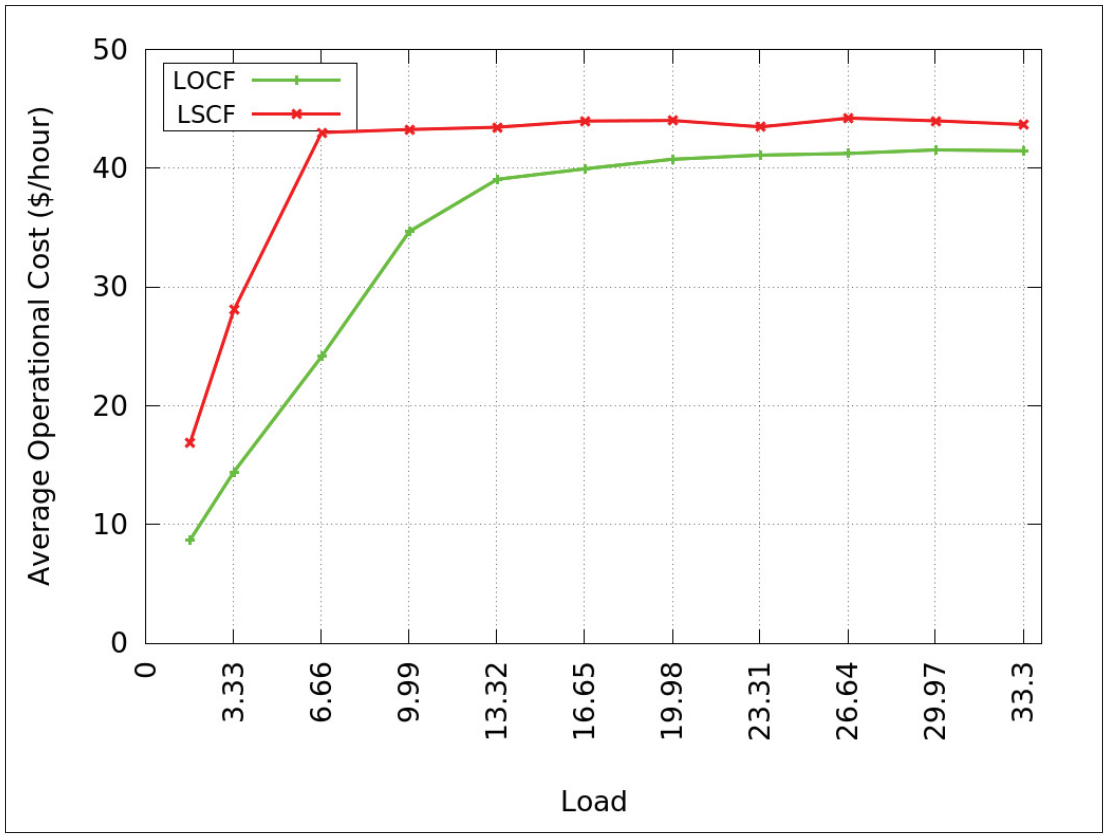


Figure 4.13 Average operational cost

4.4 Conclusion

In this chapter, we have presented the experiments and evaluation results as well as simulation results. By evaluating VNFs on different Amazon EC2 VM configurations in terms of price and performance (i.e. packet arrival rate, resource utilization, and packet loss rate), we deduce that `t2.micro` is the most cost-effective VM configuration. Hence, we have conducted our simulations to evaluate the performance of the two proposed heuristics in terms of acceptance rate, POPs utilization, operational cost, as well as synchronization cost. We have noticed that LOCF demonstrates better performance in terms of acceptance rate and operational cost whereas LSCF utilizes POPs better and reduces the synchronization cost even when load of requests is significantly high.

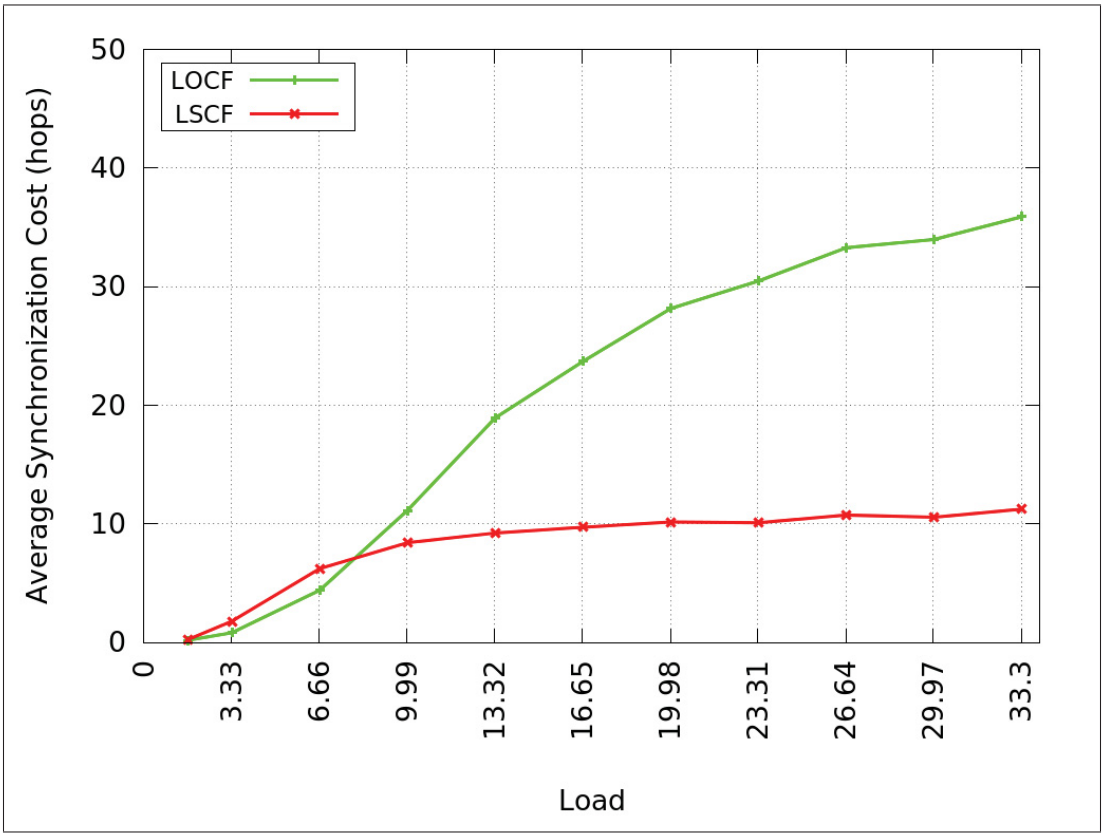


Figure 4.14 Average synchronization cost

CONCLUSION AND RECOMMENDATIONS

NFV is an emerging technology aims to decouple software from hardware and offers several benefits to the industry. Despite the fact that there is a massive progress in tackling NFV-related problems, none of the existing solutions address the placement of Service Function Chains (SFC) when multiple VNF instances are required to cope with the growing traffic demand. Hence, we address the problem of VNF instances placement in Cloud infrastructures with an objective of minimizing the operational cost taking into account the synchronization cost between pairs of VNF instances of the same type.

We formulate the problem using integer linear program (ILP) with an objective of minimizing the operational cost taking into account synchronization between VNF instances. Furthermore, We evaluate the performance of three VNFs, namely, Firewall, IDS, and NAT using different Amazon EC2 configuration (t2.micro, t2.small, t2.medium, and t2.xlarge) and compare them in terms of pricing and performance. In a comparison between t2.micro and t2.xlarge instances, we found that 16 instances of t2.micro equals one t2.xlarge instance operating for around the same price.

To solve VNF embedding puzzle, we have ran extensive experiments to evaluate the performance of the aforementioned VNF instances using Shorewall as a firewall, Snort as a network IDS, and Ubuntu-based NAT on top of t2.micro instances. We found that a firewall can process up to 11,000 packets/s and IDS can process up to 13,000 packets/s whereas NAT processes around 9,000 packets/s. By identifying the maximum processing capacity of each VNF, we can determine the number of required instances for each request.

For VNF instances placement, we propose two heuristics (i) Least Operational Cost First (LOCF) and (ii) Least Synchronization Cost First (LSCF), which consider VNF instances placement into an infrastructure composed of POPs with different energy prices; the first algorithm minimizes the operational cost using a modified DFS that explores all potential paths

between the source and the destination for embedding a request and sorts these paths based on the operational cost from the least to the highest. The second algorithm attempts to place as many VNF instances as possible into the POPs in the path whose capacity is the highest with a goal of minimizing synchronization cost between VNF instances. Both algorithms determine the number of VNF instances for each function in the service chain in order to minimize the embedding cost.

This work is highly significant for service providers as it helps them select the most appropriate VM flavor/configuration that minimizes the embedding cost and utilizes resources efficiently. However, synchronization between stateful VNFs need to be studied further in the future as the exchanged data rate between these VNF instances may incur high link utilization, which results in high communication cost.

BIBLIOGRAPHY

Alsubhi, K., Zhani, M. F. & Boutaba, R. (2012). Embedded Markov process based model for performance analysis of Intrusion Detection and Prevention Systems. *IEEE Global Communications Conference (GLOBECOM)*.

Amazon. (2017). Amazon EC2 Pricing On Demand. [accessed 20-Jan-2017].

Bari, M. F., Chowdhury, S. R., Ahmed, R. & Boutaba, R. (2015a). nf.io: A file system abstraction for NFV orchestration. *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*.

Bari, M., Chowdhury, S. R., Ahmed, R. & Boutaba, R. (2015b, Nov). On orchestrating virtual network functions. *IEEE International Conference on Network and Service Management (CNSM)*, pp. 50-56.

Beck, M. T. & Botero, J. F. (2015, Dec). Coordinated Allocation of Service Function Chains. *IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6.

Bonetto, E., Mellia, M. & Meo, M. (2012). Energy profiling of ISP points of presence. *IEEE Conference on Communications (ICC)*.

Bouet, M., Leguay, J. & Conan, V. (2015). Cost-based placement of vDPI functions in NFV infrastructures. *IEEE Conference on Network Softwarization (NetSoft)*.

Carapinha, J. & Jiménez, J. (2009). Network Virtualization: A View from the Bottom. *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, (VISA '09), 73–80. Consulted at http://doi.acm.org/10.1145/1592648.1592660.

Carpio, F., Dhahri, S. & Jukan, A. (2017). VNF placement with replication for Loac balancing in NFV networks. *IEEE International Conference on Communications (ICC)*, pp. 1–6.

Chiosi, M. et al. (2012). Network Functions Virtualisation. *SDN and OpenFlow World Congress*.

Clayman, S., Maini, E., Galis, A., Manzalini, A. & Mazzocca, N. (2014). The dynamic placement of virtual network functions. *IEEE Network Operations and Management Symposium (NOMS)*.

Cohen, R., Lewin-Eytan, L., Naor, J. S. & Raz, D. (2015). Near optimal placement of virtual network functions. *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1346–1354.

Dantas, R., Sadok, D., Flinta, C. & Johnsson, A. (2015, May). KVM virtualization impact on active round-trip time measurements. *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 810-813. doi: 10.1109/INM.2015.7140382.

Doherty, J. (2016). *SDN and NFV Simplified: A Visual Guide to Understanding Software Defined Networks and Network Function Virtualization*. Addison-Wesley Professional.

Engelmann, A. & Jukan, A. (2017). A Reliability Study of Parallelized VNF Chaining. *arXiv preprint arXiv:1711.08417*.

Feng, Y., Li, B. & Li, B. (2012). Jetway: Minimizing costs on inter-datacenter video traffic. *ACM international conference on Multimedia*.

Fischer, A., Botero, J. F., Beck, M. T., de Meer, H. & Hesselbach, X. (2013). Virtual Network Embedding: A Survey. 15.

Ghaznavi, M., Khan, A., Shahriar, N., Alsubhi, K., Ahmed, R. & Boutaba, R. (2015a). Elastic virtual network function placement. *IEEE 4th International Conference on Cloud Networking (CloudNet)*.

Ghaznavi, M., Khan, A., Shahriar, N., Alsubhi, K., Ahmed, R. & Boutaba, R. (2015b). Elastic virtual network function placement. *IEEE 4th International Conference on Cloud Networking (CloudNet)*, pp. 255–260.

Han, B., Gopalakrishnan, V., Ji, L. & Lee, S. (2015). Network Function Virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2), 90–97.

Huang, P.-H., Li, K.-W. & Wen, C.-P. (2015, Oct). NACHOS: Network-aware chains orchestration selection for NFV in SDN datacenter. *IEEE International Conference on Cloud Networking (CLOUDNET)*.

IBM. (2007). Virtualization in Education. Consulted at http://www-07.ibm.com/solutions/in/education/download/VirtualizationinEducation.pdf.

iPerf. (2017). iPerf - The ultimate speed test tool for TCP, UDP and SCTP . Consulted at https://iperf.fr/.

Karlin, A. R., Manasse, M. S., McGeoch, L. A. & Owicki, S. (1994). Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6), 542–571.

Lee, S., Pack, S., Shin, M. & Paik, E. (2014). Resource management for dynamic service chain adaptation. *IETF Draft*.

Lin, M., Liu, Z., Wierman, A. & Andrew, L. L. (2012). Online algorithms for geographical load balancing. *Green Computing Conference (IGCC), 2012 International*, pp. 1–10.

Luizelli, M., Bays, L., Buriol, L., Barcellos, M. & Gaspary, L. (2015). Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. *IEEE International Symposium on Integrated Network Management (IM)*.

M. Chiosi, S. Wright, and others. (2012). Network Functions Virtualisation (NFV). *SDN and OpenFlow World Congress*.

Mehraghdam, S., Keller, M. & Karl, H. (2014). Specifying and placing chains of virtual network functions. *IEEE International Conference on Cloud Networking (CloudNet)*.

Mell, P. M. & Grance, T. (2011). *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States: National Institute of Standards & Technology.

Miha, Amrhein, D., Anderson, P., De, A., Armstrong, J., B, E. A., Bartlett, J., Bruklis, R., Carlson, M., Cohen, R., Crawford, T. M., Deolaliker, V., Downing, P., Easton, A., Flores, R., Fourcade, G., Hanan, T., Herrington, V., Hosseinzadeh, B. & Hughes, S. (2010). Cloud Computing Use Cases A white paper. *Cloud Computing Use Case Discussion Group*, (v 4.0), 1–68. Consulted at http://cloudusecases.org/.

Mijumbi, R., Serrat, J., Gorricho, J., Bouten, N., De Turck, F. & Boutaba, R. (2015a). Network Function Virtualization: State-of-the-art and Research Challenges. *IEEE Communications Surveys Tutorials*, PP(99), 1-1.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. & Davy, S. (2015b, April). Design and evaluation of algorithms for mapping and scheduling of virtual network functions. *IEEE Conference on Network Softwarization (NetSoft)*, pp. 1-9.

Moens, H. & De Turck, F. (2014, Nov). VNF-P: A model for efficient placement of virtualized network functions. *International Conference on Network and Service Management (CNSM)*, pp. 418-423.

Networks, N. (2016). The next step in server virtualization : How containers are changing the cloud and application landscape.

Nobach, L., Rimac, I., Hilt, V. & Hausheer, D. (2016). SliM: Enabling efficient, seamless NFV state migration. *IEEE 24th International Conference on Network Protocols (ICNP)*, pp. 1–2.

Ostermann, S. et al. (2009). A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. *Conference on Cloud Computing (CloudComp)*. doi: 10.1007/978-3-642-12636-9_9.

Peuster, M. & Karl, H. (2016). E-State: Distributed state management in elastic network function deployments. *IEEE NetSoft Conference and Workshops (NetSoft)*.

Robert Tarjan. (1972). Depth first search and linear graph algorithms. *SIAM JOURNAL ON COMPUTING*, 1(2).

Sahhaf, S., Tavernier, W., Colle, D. & Pickavet, M. (2015). Network service chaining with efficient network function mapping based on service decompositions. *IEEE Conference on Network Softwarization (NetSoft)*.

Samreen, F., Elkhatib, Y., Rowe, M. & Blair, G. S. (2016). Daleel: Simplifying Cloud Instance Selection Using Machine Learning. *Network Operations and Management Symposium (NOMS)*. doi: 10.1109/NOMS.2016.7502858.

Shan, L., Xie, L., Li, Z. & Xu, A. (2007). Partial predeflection a novel contention resolution scheme for optical burst switching networks.

Shorewall. (2017). Shorewall Firewall. [accessed 22-Jun-2017].

Snort. (2017). Snort. [accessed 19-Feb-2017].

Vizarreta, P., Condoluci, M., Machuca, C. M., Mahmoodi, T. & Kellerer, W. (2017). QoS-driven function placement reducing expenditures in NFV deployments. *IEEE International Conference on Communications (ICC)*, pp. 1–7.

Wang, G. & Ng, T. E. (2010). The impact of virtualization on network performance of amazon ec2 data center. *Infocom, 2010 proceedings IEEE*, pp. 1–9.

Wang, X., Wu, C., Le, F., Liu, A., Li, Z. & Lau, F. (2016). Online vnf scaling in datacenters. *IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 140–147.

Wang, X., Wu, C., Le, F. & Lau, F. C. (2017). Online Learning-Assisted VNF Service Chain Scaling with Network Uncertainties. *IEEE 10th International Conference on Cloud Computing (CLOUD), 2017*, pp. 205–213.

Xia, Ming and Shirazipour, Meral and Zhang, Ying and Green, Howard and Takacs, Attila. (2015). Network function placement for NFV chaining in packet/optical datacenters. *Journal of Lightwave Technology*, 33(8), 1565–1570.

Zhang, Q., Zhani, M. F., Boutaba, R. & Hellerstein, J. L. (2013). Harmony: Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud. *IEEE International Conference on Distributed Computing Systems (ICDCS)*.

Zhang, Q., Cheng, L. & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1), 7–18.

Zhani, M. F., Zhang, Q., Simon, G. & Boutaba, R. (2013a). VDC Planner: Dynamic migration-aware Virtual Data Center embedding for clouds. *IFIP/IEEE International Symposium on Integrated Network Management (IM)*.

Zhani, M. F., Zhang, Q., Simona, G. & Boutaba, R. (2013b). VDC Planner: Dynamic migration-aware virtual data center embedding for clouds. *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pp. 18–25.