Performance and Survivability of Service Function Chains in Virtualized Environments

by

Zakaria ALOMARI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, AUGUST 22, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGEMENTS

I would like to convey my deep appreciation and sincere gratitude to those who have assisted me, in one way or another, through my passionate journey pursuing and realizing my PhD dream.

First of all, I would like to express my sincere and heartfelt gratitude to my advisor Professor **Mohamed Faten Zhani**, for having faith and trust in me, and for giving me this incredible opportunity to work under his supervision. His guidance, encouragement, and patience during the course of my study have enabled me to fulfill my PhD dream. Dear Faten, working with you has been the most rewarding experience of my life and without your support, this dissertation would not have been possible.

I would like to express my deep gratitude to the committee, Professor **Kim Khoa Nguyen**, **Abdelouahed Gherbi**, and **Nadjib Aitsaadi** for their efforts reviewing my thesis and providing insightful feedback.

I am deeply indebted to my beloved parents, **Aysheh** and **Mousa**, my brothers and sisters for their sincere prayers and wishes. In spite of the distance, they were an incessant source of encouragement and unconditional support during these years.

I remain indebted to my beloved wife **(Ala')**. Your comprehension, dedication, encouragement, endless support, and ceaseless love during the past years of my PhD journey have sustained me through the challenges and filled my life with joys and tenderness. A special thanks filled with love to my precious children **(Tala, Yahya, and Bana)** for cheering me up and bringing joy and happiness to my life.

Finally, I would like to give special thanks to all my friends. The time we spent together and the unconditional support will be always remembered.

Performance et survivabilité des chaînes de fonctions de service dans les environnements virtualisés

Zakaria ALOMARI

RÉSUMÉ

La virtualisation des fonctions réseau est une technologie émergente permettant le déploiement de services réseau de manière efficace et flexible sous la forme de chaînes de fonctions de service. Une chaîne de service est composée de fonctions réseau virtuel qui pourraient implémenter différentes fonctions (par exemple, IDS, pare-feu ou autres). Malgré leurs avantages, de nombreux défis sont à relever pour approvisionner ces chaînes, gérer leurs ressources et assurer leur continuité de service.

Dans cette thèse, nous abordons deux défis clés relatifs à la gestion des chaînes de fonctions de service, à savoir, 1) la synchronisation entre les fonctions réseau virtuels composant la chaîne, 2) la survivabilité de ces chaînes de service en cas de défaillance.

Le premier défi abordé dans cette thèse est de savoir comment assurer la synchronisation entre plusieurs instances d'une même fonction réseau avec des coûts minimaux et un délai acceptable. Ainsi, nous proposons une technique qui identifie le modèle de communication optimal entre les instances afin de minimiser les coûts et les délais de synchronisation. Nous proposons également une nouvelle fonction réseau de synchornisation capable d'effectuer la collecte de données et de minimiser davantage ces coûts. Nous formulons le problème de trouver le modèle de communication optimal, le nombre optimal et le placement des fonctions de synchronisation sous forme d'un programme linéaire entier visant à minimiser les coûts de synchronisation. Nous proposons aussi trois algorithmes gloutons pour faire face à des scénarios à grande échelle et nous exploîtons la migration des fonctions réseau pour réduire davantage les coûts. Les simulations montrent que les solutions proposées trouvent des solutions quasi-optimales.

Le deuxième défi que nous avons relevé est de savoir comment assurer la survivabilité des chaînes de fonctions de service en cas de pannes multiples et simultanées. Nous introduisons un cadre de gestion de la capacité de survivabilité pour gérer efficacement les sauvegardes des fonctions réseau. Nous modélisons mathématiquement le problème de survivabilité des chaîne de services sous la forme d'un programme linéaire entier qui détermine le nombre minimal de sauvegardes partagées et leurs emplacements optimaux dans l'infrastructure physique tout en garantissant un coût opérationnel minimal. Enfin, nous proposons deux algorithmes gloutons pour traiter les scénarios à grande échelle du problème et nous étudions l'utilisation de la prévsion du trafic pour réduire davantage les coûts. Les simulations démontrent comment les algorithmes proposés sont capables de fournir des solutions performantes.

Mots-clés: Chaîne de fonctions de service, Virtualisation des fonctions réseau, Synchronisation, Migration de VNFs, Survivabilité, Sauvegarde partagée.

Performance and Survivability of Service Function Chains in Virtualized Environments

Zakaria ALOMARI

ABSTRACT

Network Function Virtualization is an emergent technology enabling the deployment of network services in the form of service function chains in an efficient and flexible manner. A service chain is composed of virtual network function that could implement different functions (e.g., IDS, Firewall or others). Despite their benefits, many challenges are faced in order to provisioning such chains, manage their resources and ensure their service continuity.

In this thesis, we address tow key challenges pertaining to the management of service function chains, namely 1) the synchronization among virtual network functions composing the chain, 2) how to ensure the survivability of such service chains in case failures.

The first challenge addressed by this thesis is how to ensure the synchronization between multiple instances of the same network function with minimal costs and acceptable delay. Specifically, we propose a technique that identifies the optimal communication pattern between the instances in order to minimize the synchronization costs and delay. We also propose a novel network function named Synchronization Function able to carry out data collection and further minimize these costs. We then formulate the problem of finding the optimal communication pattern, the optimal number and placement of synchronization functions as an Integer Linear Program aiming at minimizing the synchronization costs. We also put forward three greedy algorithms to cope with large-scale scenarios of the problem and explore the use of network function migration to further reduce costs. Extensive simulations show that the proposed algorithms efficiently find near-optimal solutions with minimal computation time and provide better results compared to existing solutions.

The second challenge that we have addressed is how to ensure the survivability of service function chains in case of multiple and simultaneous failures. We introduce a survivability management framework to efficiently manage backups for network functions. We mathematically model the problem of service chain survivability as an Integer Linear Program that determines the minimal number of shared backups and their optimal location in the physical infrastructure while ensuring a minimal operational cost. Finally, we propose two greedy algorithms to deal with the problem's large-scale scenarios and we investigated the use of traffic prediction to further reduce costs. Simulations demonstrate how the proposed algorithms provide near-to-optimal solutions.

Keywords: Network Function Virtualization, Synchronization, VNF Migration, Shared Backup, Survivability.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Page

LIST OF ABBREVIATIONS

CAPEX	Capital Expenditure
OPEX	Operational Expenditure
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFV MANO	Network Functions Virtualization Management and Orchestration
SDN	Software Defined Network
NF	Network Function
VNF	Virtual Network Function
VM	Virtual Machine
SFC	Service Function Chain
IDS	Intrusion Detection System
NAT	Network Address Translation
SyncFn	Synchronization Function
NIST	National Institute of Standards and Technology
AWS	Amazon Web Services
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
API	Application Programming Interface

VMM	Virtual Machine Manager
ISP	Internet Service Providers
KVM	Kernel-based Virtual Machine
InP	Infrastructure Provider
SP	Service Provider
NV	Network Virtualization
VN	Virtual Network
ETSI	European Telecommunications Standards Institute
VNF-FG	Virtual Network Function-Forwarding Graph
CDN	Content Delivery Network
ILP	Integer Linear Program
SPT	Shortest Path Tree
SSF	Single Synchronization Function
MSF	Multiple Synchronization Functions
ARIMA	AutoRegressive Integrated Moving Average

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

ms	Millisecond
MB	Megabyte
GB	Gigabyte
min	Minute

INTRODUCTION

Until recently, the Internet successfully delivered and managed a variety of distributed services and applications. However, the Internet's enormous popularity has proven to be its greatest impediment to evolution and innovation. Indeed, with the rise of cloud computing services and the proliferation of new network applications, the amount of data is growing exponentially. However, the Internet architecture's rigidity prevents it from following this growth, resulting in what is known as the ossification problem. Furthermore, because of the multi-vendor nature of the architecture, adopting a new architecture or updating the existing one is a complicated undertaking that necessitates consensus among various stakeholders. As a result, updates to the current Internet are restricted to incremental solutions, i.e. investment in powerful enough equipment to handle this data, resulting an increase operational expenses (OPEX) and capital expenses (CAPEX).

The emergence of Network Function Virtualization (NFV) and Software Defined Networks (SDN) technologies is transforming the design and management of computer networks. These technologies give cloud providers more flexibility to provision network services and configure the network dynamically. This overcomes the limitations of traditional networks.

NFV technology adds a whole new dimension to the telecommunications industry market landscape through the ability to reduce capital investment and power consumption by consolidating Network Functions (NF) and delivering customised services as needed. Indeed, this technology revolutionizes networks by allowing to dynamically instantiate network functions in the form of software and hence there is no need for any specialised hardware. These software instances, known as Virtual Network Functions (VNFs), are instantiated and run on standard commercial servers without the need for new hardware.

Additionally, NFV simplifies service deployment by leveraging the concept of service chaining. To deliver a specific Internet service (e.g., VoIP, Web Service), providers design their services as a concatenation of Virtual Network Functions (VNFs), such as routers, intrusion detection systems (IDS), and network address translations (NAT), that run on virtual machines, process incoming traffic and direct it to its destination. This chain of virtual network functions is thus referred to as a service chain (Service Function Chain - SFC).

In recent years, many efforts have been dedicated to the problem of resource provisioning and management of these SFCs (Bari *et al.* (2012)), (Herrera & Botero (2016)), (Luizelli, Bays, Buriol, Barcellos & Gaspary (2015)), (Mijumbi *et al.* (2015)), (Qu, Assi & Shaban (2016)), (Racheg, Ghrada & Zhani (2017)). This problem entails determining the best placing for the service chain components while meeting resource requirements (CPU, memory and disk). Due to processing capacity limitation, the infrastructure provider may need to instantiate one or more instances of a particular network function when the amount of traffic increases (Gember *et al.* (2013)), (Ghrada, Zhani & Elkhatib (2018)), (Chua, Ward, Zhang, Sharma & Huberman (2016)), (Zhani & ElBakoury (2020)), (ETSI (2015)), (Salameh *et al.* (2019)). Most of network functions are stateful, which means that they keep a state that may be frequently read or updated (e.g., statistics like number of packets or bytes per flow). As a result, the instances of the same virtual network function should constantly share the same state to prevent incorrect operation (Khalid & Akella (2019)), (Khalid, Gember-Jacobson, Michael, Abhashkumar & Akella (2016)), (Gember-Jacobson *et al.* (2014)). Most of the existing studies consider data synchronization among network functions, but no one aims at minimizing the synchronization costs between the instances and ensuring that the synchronization delay does not exceed a certain bound to ensure normal operation of the function (Khalid & Akella (2019)), (Satapathy, Dave, Naik & Vutukuru (2017)), (Ma, Le, Russo & Lobo (2015)), (Peng, Leckie & Ramamohanarao (2007)).

Furthermore, another major challenges pertaining to service function chains is how to ensure their survivability in face of potential multiple and simultaneous failures that are popular in large-scale infrasctures (Zhani & Boutaba (2015)). The majority of existing research makes the assumption that the physical infrastructure is always available, which is unrealistic given the prevalence of failures in cloud network infrastructures (Lee, Ko, Suh, Jang & Pack (2017)), (ETSI (2015)), (Zhang, Zhani, Jabri & Boutaba (2014)). Due to the dependency between virtual network functions in the service chain, a failure of a single or multiple physical node in the network could easily bring down many VNFs and consequently break multiple SFCs and making their associated services unavailable. Any downtime, even for a brief period, could damage the service provider's reputation, and depending on the type of supplied service, could result in significant revenue losses (e.g., downtime could cost up to $5,600 per minute and up to $300,000 per hour according to (Cohen)).

0.1 Problem Statement

In this context, this thesis focuses in two the following two problems:

- Running the same network function in a distributed manner over multiple instances is particularly challenging. Indeed, when the network function is *stateful*, all instances should maintain the same state (e.g., statistics like number of packets or bytes per flow, list of available ports, per-connection port mapping). Ideally, all these instances should work like a single one, regardless of their number and location. Hence, they should regularly synchronize data among them. In this context, there is a compelling need to put forward synchronization schemes allowing to share data (maintain it consistent) while minimizing costs in terms of bandwidth consumed by the synchronization data and allowing to ensure an acceptable synchronization delay in order to guarantee normal operation of the network function.
- Once SFCs have been embedded into the physical infrastructure, the network operator is faced with the challenge of ensuring that these SFCs are survivable to failures. In other words, in order to avoid service interruption, they must be able to survive unexpected network failures. However, as previously stated, failures are common in physical infrastructure, and a single or multiple node failures could bring down a large number of VNFs, resulting in

the loss of several service chains. In this context, a key challenge for network operators is how many backup VNFs should be provisioned and where they should be placed so that they can take over in case of failure. Of course, the operational costs of instantiating and running these backups should be minimized.

0.2 Objectives and Contributions

To address the aforementioned challenges, this thesis focuses on the following two objectives:

- *Minimizing VNF sychronization costs:* the first objective is to ensure data synchronization among multiple instances of the same network function while minimizing the synchronization costs and ensuring that the synchronization delay does not exceed a certain bound. We achieve this objective by proposing a technique to identify the optimal communication pattern between the instances in order to minimize the synchronization cost and to ensure a bounded synchronization delay. We also introduce the use a new function called *Synchronization Function* that ensures consistency among a set of instances and allows to further reduce the synchronization costs. We then mathematically model the problem of finding the optimal synchronization pattern and the optimal placement and number of synchronization functions as an integer linear program that minimizes the synchronization cost and ensures a bounded synchronization delay. We leverage VNF migration, where an instance of a VNF deployed on an NFV infrastructure can be migrated from one physical node to another in order to further reduce the synchronization cost, and to ensure a bounded synchronization delay. Last, we put forward four algorithms that are able to efficiently explore the solution space for large-scale scenarios within a reasonable timescale. These algorithms are based on finding the optimal communication pattern between the instances, but differ in how patterns are ultimately define.
- *Ensuring total yet cost-efficient survivability for service function chains:* our second objective aims at ensuring the survivability of service chains embedded into the physical infrastructure

by provisioning in advance enough backups before failure occurs (single or multiple node failures). We achieve this objective by proactively providing the minimum number of shared backups to reduce wasted resources and determining where those backup instances should be placed in the physical infrastructure, while ensuring a minimal operational cost (running, management, and synchronization costs). Thus, we introduce a Survivability Management Framework with survivability module to capture traffic fluctuations in the SFCs and use the acquired data to reduce the number of shared backups in infrastructure. We next mathematically model the problem of service chain survival as an integer linear program that determines the minimal number of shared backups required for each VNF type and their optimal location in the physical infrastructure. Additionally, we design a traffic prediction model based on ARIMA to forecast demand traffic rates in the future, which aims to minimize the operational cost. Finally, we propose two algorithms to deal with the problem's large-scale scenarios. These algorithms differ from each other in the way they provision backup instances.

0.3 Methodology

This Section describes the methodology adopted to reach each of the aforementioned objectives.

- *Minimizing VNF sychronization costs:* to achieve this first objective, our methodology consisted in the following steps:
  - We started by studying background concepts related to cloud computing and network function virtualization. We also investigated prior solutions that tackled the considered challenges in order to better understand the state of the art of state synchronization in the cloud.
  - We focused on the problem of finding the optimal synchronization pattern to be used to ensure data exchange between the VNF instances and would minimize synchronization costs. We mathematically modeled the problem as an Integer Linear program (ILP).

  As the problem is NP-hard, we also proposed three heuristic algorithms to deal with the large-scale instances of the problem.
  - We run several simulations considering different configuration scenarios to compare the performance of the proposed heuristics compared to proposed ILP solved by CPLEX.
  - We explored the possibility to migrate network function instances to further reduce costs. Hence, we put forward an algorithm to migrate virtual function in order to minimize costs while ensuring a minimal migration costs.
  - We run extensive simulations to show how migration further improves the performance of the proposed backup provisioning solutions.
- *Ensuring cost-efficient survivability for service function chains:* we achieved this second objective by performing the following steps:
  - We began by investigating the relevant literature pertaining to the survivability problem of service function chains and virtual networks to overcome single and multiple node failures.
  - We mathematically modeled the problem as an Integer Linear Program and also proposed two heuristic algorithms that allow to determine the minimal number of shared backups required for each VNF type and their location in the infrastructure while maintaining minimal operational costs incurred by running, managing, and synchronizing the backups.
  - Through several simulations, we compared the performance of the solutions provided by the proposed heuristics compared to the optimal solution found with the proposed ILP solved by CPLEX.
  - Finally, we investigated the use of traffic prediction models to forecast traffic rates and showed how such prediction could support the proposed heuristics to further minimize operational costs.

0.4 Publications

The main outcome of this thesis in terms of scientific publications are as follows: one published conference and two submitted journal papers (the first one is under the second-round revision and the second is submitted and is under review). These publications are reported in following:

- Zakaria Alomari, Mohamed Faten Zhani, Moayad Aloqaily and Ouns Bouachir. "On minimizing synchronization cost in NFV-based environments," In IEEE/ACM International Conference on Network and Service Management (CNSM), 2020.
- Zakaria Alomari, Mohamed Faten Zhani, Moayad Aloqaily and Ouns Bouachir. "Towards Optimal Synchronization in NFV-based Environments," In International Journal of Network Management (Wiley). Under review (Second Round). Revised manuscript submitted in July 2022.
- Zakaria Alomari, Mohamed Faten Zhani, Moayad Aloqaily and Ouns Bouachir. "On Ensuring Cost-Efficient Survivability of Service Function Chains in NFV Environments," In Journal of Network and Systems Management (Springer). Submitted in August 2022.

0.5 Thesis Organization

The remainder of this thesis is structured as follows. In Chapter 1, we present the background needed to understand the different concepts, technologies and techniques that we are using in this work, we then survey the literature pertaining to service function chaining, VNF synchronization and survivability, and we highlight about the originality of our contributions. Chapter 2 presents our solutions to minimize synchronization costs among network functions in NFV-based environments. Next, we present, in Chapter 3, our solutions to ensure a complete yet cost-effective survivability for service chains. Finally, we make some concluding remarks and highlight key research directions and future extensions of this thesis.

CHAPTER 1

BACKGROUND MATERIAL AND LITERATURE REVIEW

This chapter begins by introducing the fundamental principles that will help us grasp the problem under consideration in this research effort. Then, we analyse the key literature on the research difficulties at hand, emphasising the limitations of present solutions.

1.1 Background Material

We will introduce in this section the main concepts that will lead us to understand the problem studied in this research project. We will therefore present the concepts related to our field of research, i.e. cloud computing, virtualization, and data synchronization.

1.1.1 Cloud Computing

This section introduces the concept of cloud computing, its various characteristics, layers, service models, and deployment models.

1.1.1.1 Definition

As stated in NIST (Mell & Grance (2011)), Cloud Computing be defined as follows *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models"*.

1.1.1.2 Cloud Essential Characteristics:

The following are the five essential characteristics of a cloud computing environment (Mell & Grance (2011)):

- **On-demand self-service:** This demonstrate that the cloud services could be requested and provisioned rapidly as required by users, without requiring for manual configuration.
- **Broad network access:** Cloud services are available over accessible network anytime, anywhere access by various user platforms such as mobile phones, laptops, tablets, and workstations.
- **Shared resource pooling:** Cloud computing resources are intended to support a multi-tenant model. This means that a several users can share the same physical infrastructure although keeping privacy and security over their information. Further, cloud provider's resource pool suppose to be huge and flexible adequate to service several user requirements and to offer for economy of scale.
- **Rapid elasticity:** Cloud platforms are elastic. This means that organizations can quickly provision and de-provision any of the resources for the purpose of to scale in and out to satisfy the demand and utilize resources most effectively.
- **Measured service:** This implies that cloud resource usage gets be carefully monitored, measured, and reported by the cloud provider. The cost model is according to pay-as-you-go model, where the payment is changeable according to the actual consumption by the organization.

1.1.1.3 Cloud Computing Layered Model

In general, the architecture of a cloud computing environment could be categorised into four layers, as shown in Figure 1.1 (Zhang, Cheng & Boutaba (2010)). Each of the layers carry out several particular tasks, and they are as following:

- **The Application Layer:** This layer is at the top level of the hierarchy, and includes of the actual cloud applications. This cloud applications able to advantage the automatic-scaling features to fulfil greater availability as well as performance, in addition to that they also decrease operating cost.

- **The Platform Layer:** This layer is constructed on top of the infrastructure layer and includes of operating systems and application frameworks. The main goal of this layer is to reduce the difficulty of deploying applications directly into virtual machines containers.
- **The Infrastructure Layer:** This layer is also noted as the virtualization layer. In this layer, the infrastructure constructs a pool of computing resources by dividing the physical resources using virtualization technologies such as Vmware ESXi, XEN, KVM, and Hyper-V to deploy and manage cloud's virtual resources.
- **The Hardware Layer:** This layer is accountable for managing the physical resources of the cloud, containing servers, switches, power, cooling systems, and routers. Practically, the hardware equipment is generally implemented in data centers.

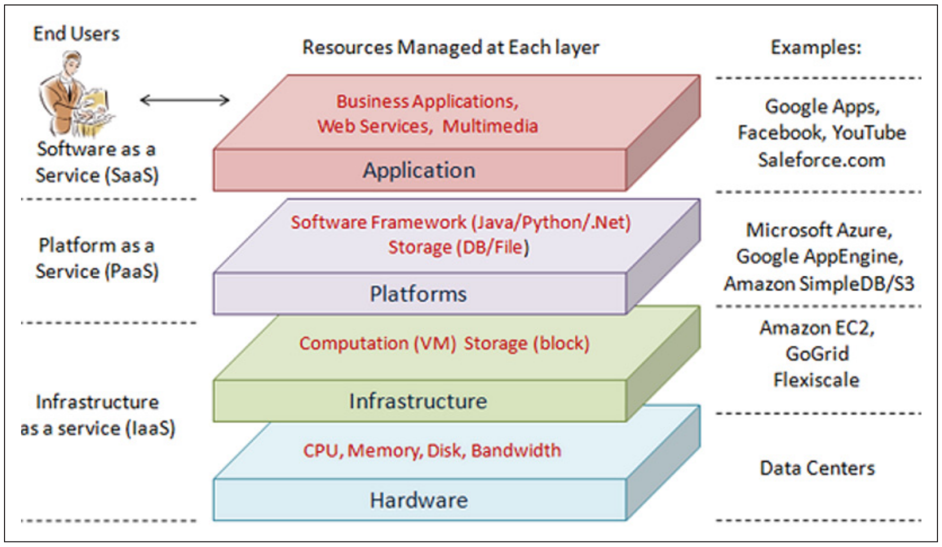


Figure 1.1 Cloud Computing Architecture
Zhang *et al.* (2010)

1.1.1.4 Cloud Service Models

The cloud service model essentially depends on the architectural layer, nevertheless, it has been divided into three types of services (Mell & Grance (2011)), as following:

- **Software as a Service (SaaS):** The end-user accesses a particular software application deployed in cloud infrastructure and managed by a standard web browser, therefore, this decreasing the require for on-device software downloads and updates. Further, the end-user able to access only without having any control the underlying infrastructure.
- **Platform as a Service (PaaS):** The end-user accesses a predetermined environment for software development, which used to build, test, and run applications. Usually this needed for application development. Thus, PaaS frees end-users from having to install hardware and software in their premises to develop or run anew applications.
- **Infrastructure as a Service (IaaS):** The end-user accesses via an Application Programming Interface (API), and basically rents the infrastructure (i.e. CPU, memory, storage). These resources are available to the end-users over virtualization technologies. However, the end-user is responsible to manage things like the operating system, apps, and storage while the cloud provider been responsible to manage and control cloud infrastructure.

1.1.1.5 Cloud Deployment Models

In this section, we will present different types of deployment models that user can subscribe to based on its needs (Mell & Grance (2011)), which incorporate the following:

- **Public Cloud:** The infrastructure of this type is generally established from information technology infrastructure not owned by the end user. This type can be owned and managed by some of the largest public cloud providers such as Microsoft Azure, Amazon Web Services (AWS), and Google Cloud. Further, a public cloud allows any subscriber to access it using an internet connection and access to the cloud space.
- **Private Cloud:** The infrastructure of this type is established for one organization comprising group end-users and restricted access to just that group. It can be operated on servers owned and maintained by the organization and available on the internet or over a private internal network.

- **Community Cloud:** The infrastructure of this type is shared and jointly accessed among several organizations that have similar cloud objectives. The infrastructure can be governed by any of the participating organizations or by a third-party managed service provider.
- **Hybrid Cloud:** The infrastructure of this type often contains a mixture of two or more different clouds, where the clouds incorporated are a combination of public, private, or community.

1.1.2 Virtualization

In this section, we will discuss the concept of virtualization, server virtualization, network virtualization as well as the network function virtualization.

1.1.2.1 Definition

Virtualization is a technology of creating a virtual version, representation of true underlying hardware or software, such as an operating system, computer program, servers, storage device, and networks. Therefore, virtualization improves IT resource utilization by dividing of physical resources into one or more execution environments, which means users, applications, and devices are able to cooperate with the virtual resource as if it were an actual single logical resource (IBM (2007)) (Landström, Bergström, Westerberg & Hammarwall (2016)).

Using virtualization in any environment provides several benefits such as optimization of workload, faster server provisioning and deployment, improved disaster recovery, and reduce hardware cost. The concept of virtualization is more generic word for many different types of computing, including the following: hardware virtualization, software virtualization, desktop virtualization, and storage virtualization (COUTINHO, SCHAAPMAN & TEMIN (2014)).

1.1.2.2 Server Virtualization

As stated in IBM Corporation (IBM (2007)), Server Virtualization be defined as follows: a virtualization technique that implies dividing a physical server to run several secure virtual servers through the use of virtualization software. This leads a possibility to merge physical servers, therefore server virtualization allows reducing the hardware purchase and also decreases management costs by eliminating infrastructure expansion at the server level. In other words, server virtualization allows a physical server to share computer resources with other virtual servers, by dividing of a physical server into smaller virtual servers to maximize the server resources as illustrated in Figure 1.2.

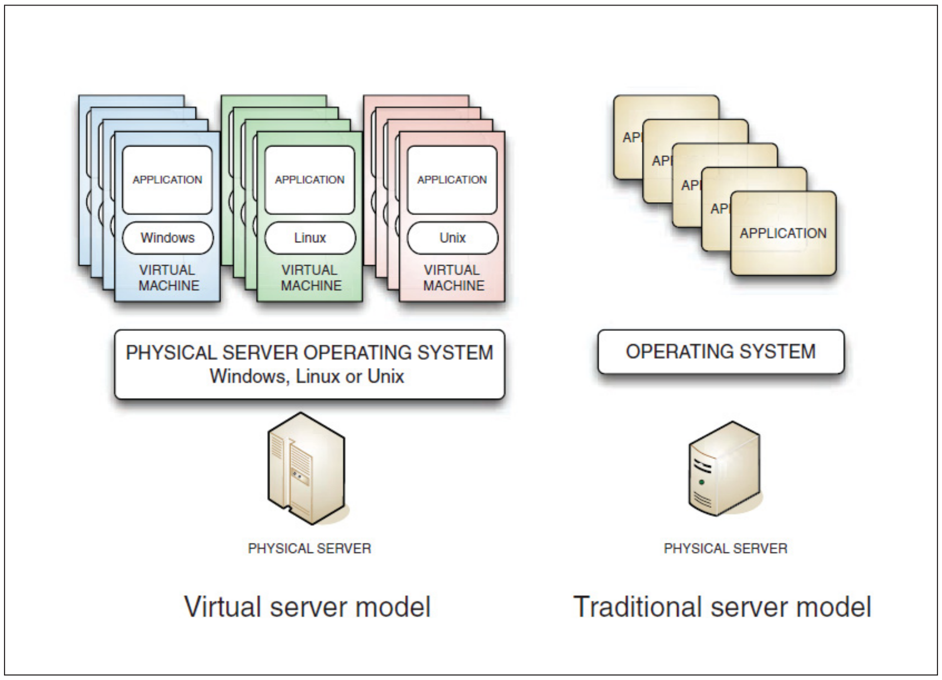


Figure 1.2 Server Models
IBM (2007)

Each virtual machine acts like it owned the whole physical machine, including a virtual CPU, virtual memory, virtual storage and so on. Further, each virtual machine has an operating system also called "guest operating system", those operating systems communicate with hardware by Virtual Machine Manager (VMM). Each guest operating system handles its own applications as it normally does in a non-virtual environment, exclude that it has been isolated in the physical machine by the VMM (IBM (2007)) (Tulloch (2010)). For instance, a virtual hard disk in VM is kept in a file located on the hard drive. Moreover, several VMs be able to exist within a single host at one time, therefore, they may be shared the same physical machine's resources. In addition to that, each VM is totally isolated from the host machine and other VMs. Thus, if a VM collapses, all remaining VMs are unaffected (IBM (2007)).

Generally, there exist several advantages by the use a server virtualization strategy. For example, it enables to deploy, manage, and operate several operating system instances at once on that single physical server, which means cost-effective way in IT. In addition to that, it saves space by decreasing the number of physical servers a user should have on its premises, as well as decrease energy expenditure because there exist fewer physical servers consuming power. If the resource needs of one of the applications running in a virtual server increases, so this VM shifts from one server to another with more available resources to balance workload. For instance, one common usage of this technology is in Web servers, Internet Service Providers (ISP) or website administrators may have different domain names, IP addresses, email administration, security systems, passwords and so on. Using virtual Web servers, these services are maintained as if they were in a dedicated server environment rather than requiring a separate computer for each Web server.

Moreover, VMs are managed by a Hypervisor and also can be called VMM. The Hypervisor is a firmware that controls several guest operating systems on a single computer. In reality, the guest operating system shares the hardware of the host computer and the Hypervisor manages the CPU, memory, storage, and other resources. As well as, the Hypervisor allocates what each operating system needs, and destruction guest operating system when they are no longer needed. Further, Hypervisors can be categorized to two various types as is revealed in Figure 1.3. The first type is Type-1 Hypervisor, this type is directly runs on the host's hardware to manage the hardware and control the guest VMs such as RTS Hypervisor, and Oracle VM. The second type is Type-2 Hypervisor, this type is a software interface that mimic the devices with which a system usually interacts. In that case, a guest operating system runs as an operation on the host, whereas the Hypervisor separates the guest operating system from the host operating system such as VMware Workstation and Kernel-based Virtual Machine (KVM) (Lowe (2012)) (Sridharan (2012)) (Mounika & Chinnaswamy (2016)).

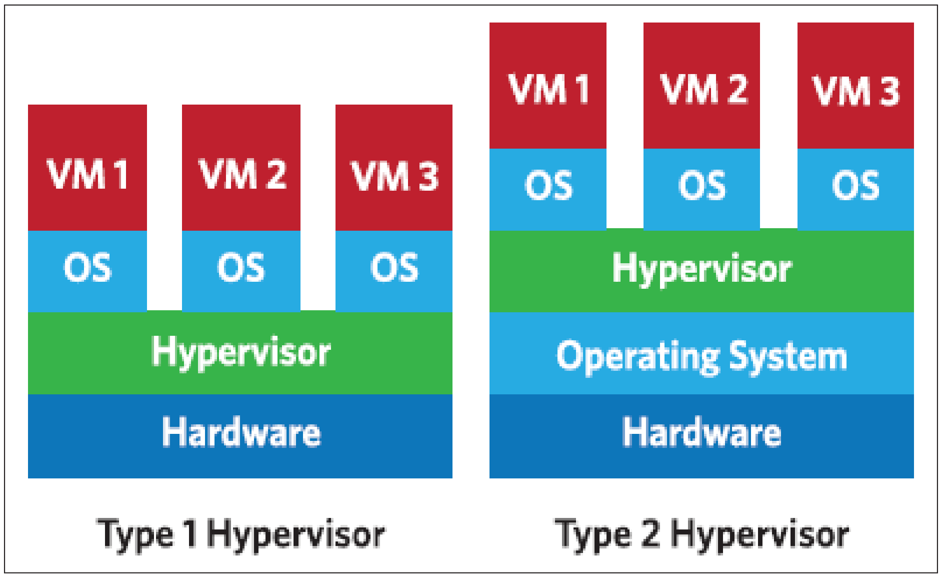


Figure 1.3 Difference between type 1 and type 2 hypervisors
Lowe (2012)

1.1.2.3 Network Virtualization

Network Virtualization (NV) is mainly defined as the capability to establish logical networks which are decoupled from the physical network, this means making a physical network show as several logical ones (Han, Gopalakrishnan, Ji & Lee (2015)) (Bays, Oliveira, Barcellos, Gaspary & Madeira (2015)). In this context, the roles of the traditional Internet Service Provider (ISP) is decoupled into two separate entities: Infrastructure Providers (InPs), who is accountable to manage the physical infrastructure, and Service Providers (SPs), who buy slices of resources (e.g., CPU, memory, bandwidth) from InP and then build a customized virtual network (VN) to offer service to end-users in Figure 1.4.

As shown in Figure 1.4, network virtualization enables to establish a logical network of the hardware and software networking resources. The physical network devices are in charge of the forwarding of packets. While, VN offers an abstraction to deploy and manage network services and underlying network resources. Therefore, a virtual node can be allocated on any physical node and virtual link can be mapped onto physical path (Carapinha & Jiménez (2009)).

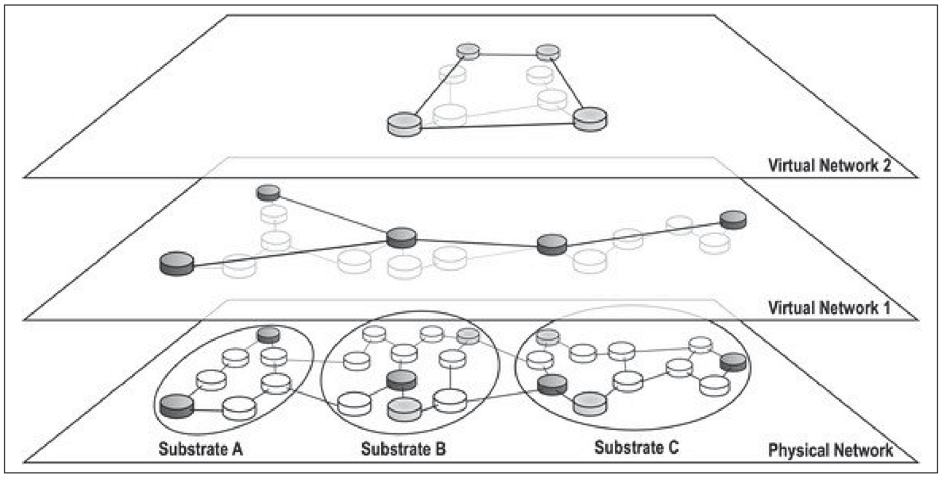


Figure 1.4 Network Virtualization Model
Bays *et al.* (2015)

1.1.2.4 Network Function Virtualization

According to the European Telecommunications Standards Institute (ETSI (2012)), Network Functions Virtualization (NFV) is an emerging technology that provides a new approach to design, deploy, and manage networking services. NFV is considered as a way to reduce cost and expedite service deployment for network operators. This is carried out by decoupling the network functions (NFs), such as IDSs, Firewalls, NATs, DNSs from the hardware of the traditional middle-boxes and packaged as virtual machines (VMs) on commodity hardware so they can run in software.

Individual Virtual Network Functions (VNFs) are an essential element of NFV architecture. They are handling specific NFs that run in one or more VMs on top of the hardware networking infrastructure. The network service chain consists of a set of VNFs that can process incoming traffic in a specific order. This sequence is called a service function chain (SFC) or VNF Forwarding Graph (VNF-FG) as depicted in Figure 1.5 (Beck & Botero (2015)).

Separating NFs from hardware yields numerous benefits for the network operator including: decreasing network power consumption, decreasing operator capital and operational cost,

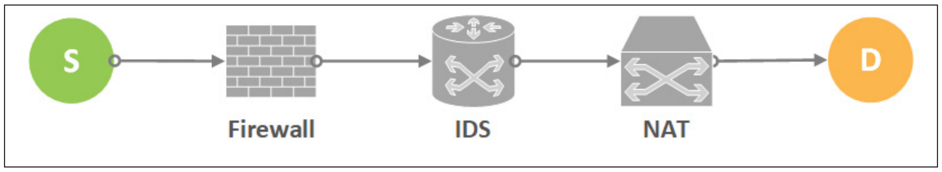


Figure 1.5 Service Function Chain
Beck & Botero (2015)

decreasing maintenance and hardware costs, greater flexibility to scale up, scale down or evolve services (ETSI (2012)).

In brief, network function virtualization has transformed the way service providers design, deploy, and operate their services. Indeed, NFV promises service providers greater flexibility to deploy network services more rapidly and at a lesser cost, hence improving service agility through a variety of variations in how network service provision is conducted compared to existing practise (Mijumbi *et al.* (2015)):

- **Decouple software from hardware:** separating network functions from the devices that run them allows the two to evolve independently of each other;
- **Flexible deployment of network function:** Detaching software from hardware allows infrastructure resources to be reallocated and shared. Therefore, components can be instantiated on any network device and their connections can be flexibly configured. This helps network operators deploy new network services faster on the same physical platform;
- **Dynamic scaling:** Decoupling network function functionality into instantiable software components provides greater flexibility to dynamically adapt VNF performance to operator needs.

Figure 1.6 presents the network function virtualization architecture which according to (ETSI (2020)) consists of three components: the network function virtualization infrastructure (NFVI), the virtual network functions (VNFs) and NFV management and network orchestration (NFV MANO).

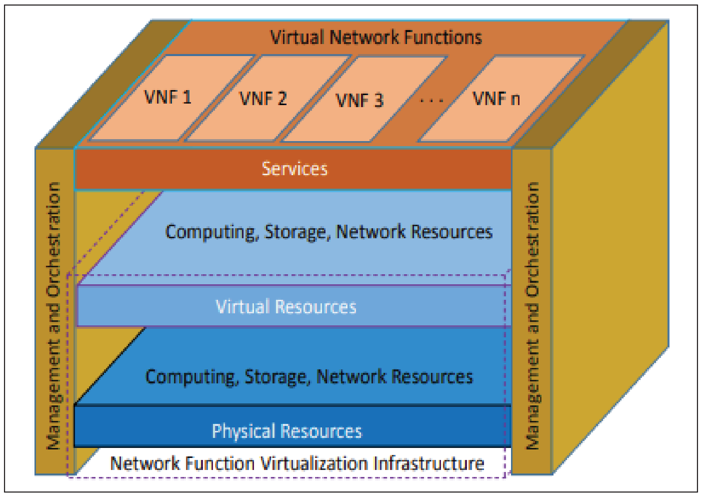

Virtual Network Functions
VNF 1
VNF 2
VNF 3
...
VNF n
Services
Computing, Storage, Network Resources
Virtual Resources
Computing, Storage, Network Resources
Physical Resources
Network Function Virtualization Infrastructure
Management and Orchestration
Management and Orchestration

Figure 1.6 Network Function Virtualization Architecture
Mijumbi *et al.* (2015)

- **Network Function Virtualization Infrastructure (NFVI):** NFVI is a combination of hardware (computing hardware) and software (abstractions of computing, storage, and network resources) that creates the environment in which VNFs run.
- **Virtual Network Functions (VNFs):** A VNF is a network function implementation that is deployed on virtual resources such as a virtual machine or a container.
- **NFV MANO:** NFV MANO provides the management and orchestration features required for VNF provisioning.

1.1.3 Data Synchronization

In this section the concept of data synchronization and data synchronization techniques between multiple devices has been introduced.

1.1.3.1 Definition

Data synchronization is a technique of establishing consistency among each source of data with its different endpoints and vice versa, as well as harmonization of data over time (Shodiq, Wongso, Pratama, Rhenardo *et al.* (2015)) (Faiz & Shanker (2016)). In other words, it points to the idea of keeping several copies of data-set (i.e., state) in coherence with one another, or to maintain data integrity.

1.1.3.2 Data Synchronization Techniques

The distributed systems raise the needs to synchronize data between several devices and data store locations. Thus, data synchronization processes contains a source and destination entity, and according to data appearance, is classified into two models of synchronization (Endo, Miyamoto, Kumagai & Fujii (2004)) (Hassani, Haghighi, Burstein & Davey (2017)):

1. *Client-Server Model:* The server synchronizes the data and distributes the copies to each client device. However, the data synchronization can be done in two ways: in a single direction or in both directions. Once data is changed on client device and forward to server, this is named upload-only synchronization. Once data changes only happen on the server, then client device has to retrieve changes from the server, this is named download-only synchronization. Whenever the above activities are carried out together, then it commonly referred as bi-directional data transfer, this is named bi directional synchronization (Malhotra & Chaudhary (2014)) (Faiz & Shanker (2016)).
2. *Peer-to-Peer Model:* The system does not rely onto any single centralized server. However, if a peer connects with other peers, it can synchronize data and collaborate directly. In other words, each peer acts as the server for the data stored on it. Thus, each peer is able of pushing data to other peers as well as getting the data by sending pull requests (Endo *et al.* (2004)) (Kubler, Främling & Derigent (2015)).

1.2 Literature Review

This section provides an overview of the existing literature focusing on the problems addressed in this work. Hence, in the first Section of this chapter, we review research work addressing issues related to state synchronization. We also present relevant work on VNF instance migration as our proposed solutions highly relies on migration. In the second section of this chapter, we provide an overview of representative work on the fault management and addressing the survivability problem in face of single or multiple failures. We also identify the key limitations of all the covered research efforts and highlight the originality of our solutions compared to the state of the art.

1.2.1 State Synchronization

Satapathy *et al.* (2017) propose a network state management system that introduces two different solutions to ensure state synchronization. In the first solution, an instance synchronizes the state at the receipt of each single packet, whereas in the second solution, the state is synchronized after receiving multiple packets. However, the first solution obviously has a high synchronization overhead whereas the second may incur additional delay to wait for multiple packets.

Peuster & Karl (2016) propose E-State management framework to synchronize the state of all instances of the same network function. They use a publish/subscribe communication pattern to share the state. Subscribers are the Instances who subscribe to a same state, and Publishers are the instances who publish the state. However, this solution does not take into account the synchronization costs and delay.

Rajagopalan, Williams, Jamjoom & Warfield (2013) propose a state management system, which classifies state inside network function into internal and external. Internal state is required for a given instance to run. External state is the state that is shared across all instances of the same type. This external state needs to be maintained consistently between instances of the same function. This system leverages a shared library used by instances to share the external state among them. This shared library supports a callback function, where each instance registers a callback function that is invoked automatically by the library when another instance updates its external state. This work also does not consider the synchronization costs and delay.

Khalid & Akella (2019) propose a NFV framework to support state synchronization among stateful network functions. They leverage a data store to ensure the state consistency among multiple instances of network function. Each instance records a callback function with the data store. This function is used by the data store to update the state of the instance on behalf of the others. This work does not consider synchronization costs and delay.

Xie *et al.* (2019) propose a framework named *Dual* to address the synchronization problem between instances of the same network function. An approximation algorithm is designed to minimize the synchronization traffic between the instances by placing the instances close to each other in the physical infrastructure. However, this work does not take into consideration the synchronization delay.

Ma *et al.* (2015) propose an algorithm for intrusion detection systems that can operate across multiple instances. In this algorithm, each instance directly shares state with the others and waits on other instances to complete processing. However, this work does not define a specific communication pattern for data synchronization and does not consider synchronization delay.

1.2.2 VNF Migration

Nobach, Rimac, Hilt & Hausheer (2016) propose Slim, a statelet-based framework for seamless instance migration in NFV infrastructure that is based on a novel interface added to the VNF. Slim sends a snapshot of the current state of the source VNF instance over the corresponding stream to the destination instance. The results show that SliM performs around three times better than full duplication in terms of link utilization. However, this solution does not take into account the distance between the source and destination instances and the communication costs raised due to congested links.

Cho, Taheri, Zomaya & Bouvry (2017) propose a VNF real-time migration algorithm. This algorithm aims at minimizing the delay among VNFs by migrating VNFs to different virtual machines in order to reduce the end-to-end delay of SFCs. The algorithm rearranges VNFs to reduce bidirectional network delay. However, this work does not take into account that a single VNF may be implemented by multiple instances, and hence does not consider synchronization among instances of the same type.

A recent study in Gember-Jacobson *et al.* (2014) hints that some types of VNFs, such as intrusion detection systems (IDSs), are sensitive to packet loss as they may miss-detect some attacks if the metadata for earlier packets in the flow is missed. For such types of VNFs, new approaches in Gember-Jacobson *et al.* (2014); Gember-Jacobson & Akella (2015); Nobach, Rimac, Hilt & Hausheer (2017) have been proposed for seamless migration of VNFs with strict requirements. The main goal is to conduct VNF migration loss-free while packet orders are also preserved. For example, in Gember-Jacobson *et al.* (2014), the SDN controller is used to buffer packets during the downtime incurred by migration and transfer them to the new instance once the VNF is up and running. However, this work did not consider the distance between the source and destination instances.

Table 1.1 provides a comparison between existing solutions addressing the state synchronization problem and the current work. Unlike previous work that looked only on the way synchronization can be carried out, the novelty of our solution lies in the idea of considering how to minimize the costs of this synchronization while ensuring that the synchronization delay does not exceed a certain bound. This ensures that any network function implemented in multiple instances operates normally and is not impacted by synchronization delays.

Table 1.1 Our solution versus existing solutions

References	**State synch.**	**Synch. delay**	**Synch. cost**	**Synch. pattern**	**VNF instance migration**
Satapathy *et al.* (2017)	√	×	×	×	×
Peuster & Karl (2016)	√	×	×	√	×
Rajagopalan *et al.* (2013)	√	×	×	×	×
Khalid & Akella (2019)	√	×	×	×	×
Xie *et al.* (2019)	√	×	×	√	×
Ma *et al.* (2015)	√	×	×	×	×
Our approach	√	√	√	√	√

1.2.3 Service Function Chain Survivability

This section provides an overview of recent studies that have addressed the survivability problem. The literature can be divided into two categories: reactive and proactive strategies

Zhani & Boutaba (2015). The proactive strategy allocate backup resources before the failure, whereas the reactive strategy allocates the resource after the failure.

A common proactive strategy was adopted in the literature to improve service survivability and consists in using redundant resources, wherein a portion of the SFC (or the virtual network) is duplicated to ensure redundancy Beck, Botero & Samelin (2016), Chowdhury *et al.* (2016), Ding, Yu & Luo (2017), Huang, Liang, Shen, Ma & Kan (2019), Yin *et al.* (2018), Xiao, Wang, Meng, Qiu & Li (2014). These redundant resources ensure service continuity in the event of a failure. While such a strategy is successful in guaranteeing the continuity of important services, the backup resources are costly and should remain operational even if they are not used. This is why minimizing the amount of backup resources was the purpose of a large body of existing works.

For instance, Beck *et al.* (2016) investigated both node and link failures and proposed a heuristic for provisioning backup VNFs and links in tandem with the VNF. This heuristic works by provisioning two chains for the same SFC, a primary one that should operate and a secondary one as a backup. Primary and secondary SFC's nodes and links should be distinct and disjoint, respectively. Backup links are provisioned to connect an SFC's secondary and primary nodes, ensuring that traffic is routed to the secondary SFC if a node in the primary SFC fails.

Chowdhury *et al.* (2016) presented a solution called Dedicated Protection for Virtual Network Embedding (DRONE) to ensure the survivability of virtual networks by deploying a dedicated backup to secure multiple primary nodes. Accordingly, each node and link in a virtual network request is associated with a dedicated backup resource in the physical network, ensuring that the virtual network can survive a single physical node failure.

Ding *et al.* (2017) suggested an effective strategy for selecting which VNFs require backup in order to increase SFC's overall reliability. The strategy selects which VNF(s) should have backup based on a measure called the Cost-aware Importance Measure, which is the ratio of SFC availability improvement to the cost to pay when providing backup for a VNF. The primary goal of this placement method is to maximize backup VNF(s) reliability while minimizing backup costs.

Yin *et al.* (2018) used path splitting to tackle the problem of survivable virtual network embedding against multi-failures with minimal resource consumption. In their solution, the number of backup paths and backup nodes is selected based on the desired level of protection and the shared risk group constraint. Xiao et al. Xiao *et al.* (2014) offered a topology-aware approach for virtual network resource allocation and fail-over remapping based on a set of pre-computed detour-paths.

To reduce backup costs, the majority of existing works choose an intermediate strategy, providing backup resources for only a portion of VNFs, sharing backup resources among numerous VNFs, or reactively allocating restoration resources only when a failure is detected. The solutions in Ding *et al.* (2017), Qing, Weifei & Julong (2017), Dinh & Kim (2019) are based on identifying a subset of critical VNFs to which backup resources should be given, in order to reduce the resources required to meet a specific degree of service reliability. For instance, Qing *et al.* (2017) proposed a protection optimization strategy to discover the set of nodes that, if protected, will maintain the majority of the SFC operational. These nodes are known as critical nodes, and they have to be backed up.

To reduce the likelihood of failure, the most crucial VNFs are assigned to dependable physical nodes Bijwe, Machida, Ishida & Koizumi (2017), Zhang, Wang, Qiu & Guo (2019). For instance, Bijwe *et al.* (2017) studied the prospect of reaching the requisite SFC reliability by enhancing the reliability of each individual VNF (i.e., creating adequate backups corresponding to each of the chain's individual VNFs) with the purpose of minimizing overall resource cost. Zhang *et al.* (2019) introduced a node-ranking algorithm to create a redundancy mechanism to safeguard the service from interruptions. The method minimizes backup resource use in response to a higher acceptance ratio of SFC requests.

The research in Aidi, Zhani & Elkhatib (2018), Fan *et al.* (2015), Fan *et al.* (2018), Li, Liang, Huang & Jia (2019), Yamada & Shinomiya (2019) focuses on finding the minimal number of backup VNF instances needed to ensure a certain level of reliability, as well as where they should be placed. In Aidi *et al.* (2018), Fan *et al.* (2015) sharing backup nodes between several VNF instances is permitted.

Aidi *et al.* (2018) suggested a framework for managing the survival of SFCs and backup VNFs in an efficient manner. They aimed to find the minimum number of needed backups to protect service function chains and determines their optimal location within the physical infrastructure.

Fan *et al.* (2015) presented an approach for selecting online backups. VNF chains are first mapped onto the physical infrastructure. If the mappings performed violate the reliability requirements, backup VNFs are chosen on the fly. Another relevant work is that of Fan *et al.* (2018) who suggested a method for providing SFC availability while taking into account heterogeneous failure processes; an optimization problem was developed to minimize the required backup resources while meeting a specified level of availability for an SFC request.

Li *et al.* (2019) studied reliable virtual network function service delivery in a multiaccess edge computing system by supplying primary and backup SFC instances in order to meet the reliability requirements of mobile users. Hmaity, Savi, Musumeci, Tornatore & Pattavina (2017) offered three strategies for service protection (end-to-end protection, virtual link protection, and virtual node protection), with the survival problem for each strategy represented and solved as an ILP problem with the goal of decreasing the number of active VNF instances, which is a measure of CapEx and OpEx costs.

Casazza, Bouet & Secci (2019) proposed a Variable Neighboring Search (VNS) algorithm with an elastic VNF protection model for SFC orchestration. The elastic architecture allows for the provision of VNF protection only when it is required, while also allowing for the transition from dedicated backup to shared backup protection, leading to effective resource use.

A two-step embedding approach that might survive the failure of a single facility node was examined by Yu, Anand, Qiao & Sun (2011) and Ayoubi, Chen & Assi (2016b). The first step is to redesign a virtual network request with redundant virtual nodes; the new virtual network is then integrated into the physical network with backup resource sharing in mind to reduce network resource expenses.

Finally, Bo, Huang, SUN, Chen & Liu (2014) suggested a greedy strategy for locating replacement substrate resources for a virtual network's impacted portion. In the event of insufficient resources, this strategy necessitates remapping the entire virtual network, resulting in protracted service outages. Ghaleb, Khalifa, Ayoubi, Shaban & Assi (2016) addressed multiple failures by introducing a heuristic to select a backup node. The method parses the solution space and speeds up the backup search process by using filtering strategies. Ayoubi, Assi, Narayanan & Shaban (2016a) established that the survivability-aware embedding is NP-hard and offered a polynomial time heuristic approach for restoring failed services while meeting QoS criteria for delays in the case of single-node failures.

Unlike previous works that looked to provision backups based on the number of original VNFs (usually they provision as many backups as primary instances or less) to handle single or multiple node failures, the novelty of our solution lies in the idea of analyzing and predicting the demand and then provisioning the backups based on the prediction, which further reduces the number of needed backups. This ensures that there are enough backups before the failure happens and enough to handle single or multiple simultaneous node failures. Furthermore, the proposed solution uses backups shared among VNFs of the same type regardless of the chains they belong to (which further reduces the amount of backup resources).

In addition, to the best of our knowledge, this effort is the first to consider minimizing the operational costs altogether (running, migration, and synchronization costs) and consider multiple and simultaneous failures.

CHAPTER 2

TOWARDS OPTIMAL SYNCHRONIZATION IN NFV-BASED ENVIRONMENTS

2.1 Introduction

The emergence of Network Function Virtualization (NFV) technology is currently transforming the way networks are designed, deployed, and managed as it allows operators to deploy network services at low cost and high flexibility Clemm, Zhani & Boutaba (2020); Vaquero *et al.* (2019); Varghese *et al.* (2019). The NFV technology allows to create *Virtual Network Functions* (VNFs) (e.g., Load balancer, Firewall, IDS, NAT) and connect them to build *Service Function Chains* (SFC) that will process the incoming traffic from the source to the destination.

As a matter of fact, the traffic may rise occasionally because of higher demand. In this case, the infrastructure provider might be required to implement the same VNF in multiple virtual machines instances due to limited processing capacity, resource costs or location constraints Gember *et al.* (2013); Ghrada *et al.* (2018); Chua *et al.* (2016); Zhani & ElBakoury (2020); ETSI (2015); Salameh *et al.* (2019). However, running the same network function in a distributed manner over multiple instances is challenging.

Indeed, network functions are either stateless or stateful. On one hand, when a network function is stateless, it can operate without maintaining any state Kablan, Alsudais, Keller & Le (2017); Khalid & Akella (2019). Hence, all instances implementing this function can operate independently and successfully without sharing any information. In this case, no data synchronization among them is required. On the other hand, when a network function is stateful, all instances should maintain the same state (e.g., statistics like number of packets or bytes per flow, list of available ports, per-connection port mapping) that should be frequently shared among them and regularly updated in a timely manner. Ideally, all these instances should work like a single one maintaining a unique state, regardless of their number and location Woo *et al.* (2018). Thus, they need to have the same state to operate normally, and, hence, timely and efficient data synchronization between the instances becomes mandatory Khalid & Akella (2019); Khalid *et al.* (2016); Gember-Jacobson *et al.* (2014). In this case, there is a compelling need to ensure a synchronization among them with minimal costs and acceptable delay.

In this context, synchronization costs could be defined as the number of messages exchanged between the instances of the same network function. Synchronization delay is the amount of time needed to exchange these messages among the involved instances in order to reach data consistency. Of course, if this delay is high, the performance and operation of the network function may be impacted as it may result in state inconsistency among the instances leading to late or inaccurate decisions Aloqaily, Kantarci & Mouftah (2014).

Several studies have recently focused on the state synchronization challenge. For instance, the authors in Khalid & Akella (2019); Satapathy *et al.* (2017) suggested to set up a common repository storing and managing state shared among instances. The authors in Ma *et al.* (2015); Peng *et al.* (2007) have considered a different technique where each instance broadcasts its state to the other instances. Unlike existing literature, our work does not only target to carry out synchronization but aims at minimizing the synchronization costs between the instances and ensuring that the synchronization delay does not exceed a certain bound to guarantee normal operation of the function. To the best of our knowledge, no previous work considered minimizing the state synchronization costs and delay of VNF instances.

In our work, we propose a technique that identifies the optimal communication pattern between the instances that minimizes the synchronization costs and ensures a bounded synchronization delay. We also introduce the use a new function called *Synchronization Function* that ensures consistency among a set of instances and allows to further reduce the synchronization costs. We first formulate the problem of finding the optimal communication pattern, the optimal placement for the synchronization functions, and number of synchronization functions as an Integer Linear Program (ILP) aiming at minimizing the synchronization costs. We also devise three algorithms, namely *the Shortest Path Tree Algorithm* (SPT), *the Single Synchronization Function Algorithm* (SSF), and *the Multiple Synchronization Function Algorithm* (MSF) that are able to efficiently explore the solution space for large-scale scenarios within a reasonable timescale. The algorithms are implemented, evaluated for multiple small- and large-scale scenarios and also compared, for small-scale scenarios, to the ILP, which was implemented in the CPLEX optimizer IBM (2020).

Further, we propose a new solution that leverages instance migration. Indeed, we explore how VNF instances could be migrated from one physical node to another in order to further reduce the synchronization costs, and to ensure that the bound on synchronization delay is respected. We hence propose a *Migration-based optimization Algorithm* that could be executed after running the MSF algorithm (denoted by M-MSF) and CPLEX optimizer (denoted by M-CPLEX) to further optimize instance placement. Through extensive experiments, we show that, thanks to migration, the proposed solutions are able to reduce by up to 40% the synchronization costs and by up to 12% the synchronization delay.

The remainder of this chapter is organized as follows. Section 2.2 presents some motivational scenarios to better highlight the importance of data synchronization among instances. Section 2.3 provides a detailed description of the problem of finding the optimal communication pattern to carry out synchronization. Section 3.5 presents the mathematical formulation of the addressed problem. We then describe the proposed heuristic solutions in Section 2.5. Section 2.6 presents the experimental results. Finally, in Section 3.8, we provide some concluding remarks.

2.2 Motivational Examples

Ideally, if a stateful network function is implemented in multiple instances, the instances should share the same state as they need to operate like a single instance regardless of their number and location. However, the state synchronization may be costly and has to be carried out in a bounded delay to ensure correct and quick operation. In the following, we provide two examples of VNFs and show how the synchronization could be costly and how it could impact the performance and operation of the network function.

Example 1 : Let us consider an Intrusion Detection System (IDS) as an example to demonstrate the importance of state consistency and synchronization delay among multiple instances implementing the same network function. Figure 2.1 shows an example of an IDS implemented in two instances. For instance, a Deny-of-Service attack is detected when the number of received TCP SYN packets reaches a threshold k. Each IDS instance maintains a counter for the number of received packets. The attacker may send the malicious traffic from two sources ($k/2$ from first source and $k/2$ from the second one), and hence each IDS instance process only half of the traffic. If the two IDS instances collaborate to synchronize the state (i.e., the number of packets) in a timely manner, the counters will be updated to k, and thus, the attack will be detected by both instances. Any delay in the synchronization will result in a late decision, which may impact the operation of the IDS and hence the network and service performance. The problem is even more challenging when multiple instances are deployed and when they are located in locations that are geographically distant from each other.

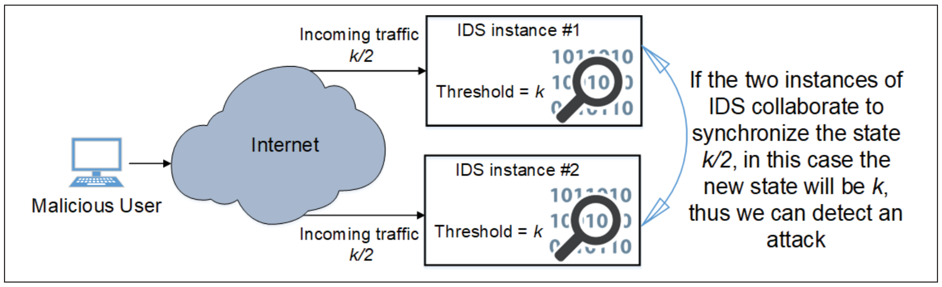

Incoming traffic
k/2
IDS instance #1
Threshold = k
Internet
Malicious User
IDS instance #2
Threshold = k
Incoming traffic
k/2
If the two instances of IDS collaborate to synchronize the state k/2, in this case the new state will be k, thus we can detect an attack

Figure 2.1 State synchronization between two IDS instances

Example 2 : In the context of Content Delivery Networks (CDN), multiple CDN instances may host large amounts of content. If these instances have to synchronize their content, this may consume significant amount of bandwidth. It is therefore of utmost importance to find the best synchronization pattern (e.g., paths for exchanged data) to carry out synchronization with minimal bandwidth consumption and minimal delay.

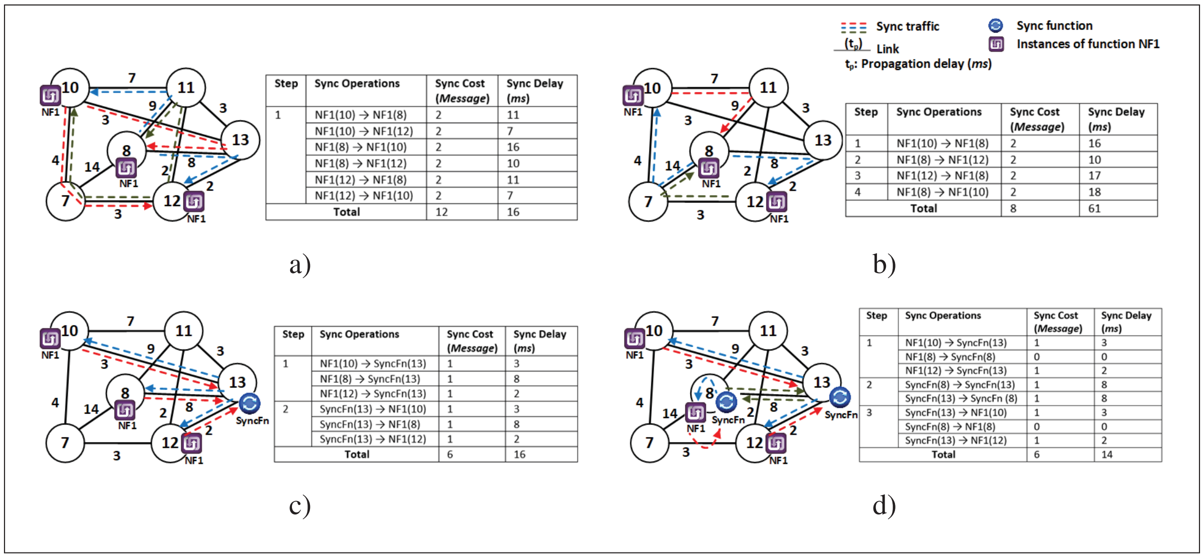
2.3 Problem Description

As previously mentioned, a VNF could be implemented in multiple instances. Each instance corresponds to a virtual machine or a container running on top of the hardware networking infrastructure ETSI (2015). These instances may be placed in geographically distributed locations (e.g., Figure 2.2) and there are different communication/synchronization patterns that could be defined to ensure they can exchange data and synchronize their state. A communication pattern defines the way data is exchanged (i.e., the order at which data is sent) as well as the paths that should be taken to deliver it to the involved instances. The goal of the following example is to show how the communication pattern impact the costs and the delay of the synchronization.

Figure 2.2 shows four different communication patterns (i.e., a, b, c, d) that could be used to carry out synchronization among instances of a function (NF1). We can see that there are three instances of function NF1 distributed in different physical nodes and synchronizing their state (see red, blue, and green dotted arrows).

Figure 2.2(a) illustrates how instances of function NF1 synchronize the same state with each other periodically at the same time. In this communication pattern, the synchronization operations are achieved in one step. For example, the instance that is located in node 10, NF1(10), synchronizes the same state with the other instances NF1(8) and NF1(12) by using the paths $10 \rightarrow 13 \rightarrow 8$ (the sync cost is 2 $messages$ and the delay is 11 ms) and $10 \rightarrow 7 \rightarrow 12$ (the sync cost is 2 $messages$ and the delay is 7 ms). Similarly, all other instances NF1(8) and NF1(12) will follow up the same procedure. As a result, the total synchronization cost between instances is the number of messages exchanged between them which is 12 $messages$ and the delay will be represented by the path that has the highest delay which is 16 ms.

Figure 2.2(b) shows that each instance synchronize the same state to the next instance sequentially. In this communication pattern, the synchronization operations are achieved in multiple steps. This means, the instances take turns sending and receiving the same state. The receiving instance reads the state addressed to it and then passes the state and any additional update to the next instance. This continues until the same state reaches the desired receiving instance. For example, the instance NF1(10) synchronizes the same state with the next instance NF1(8) by using the path $10 \rightarrow 11 \rightarrow 8$ (the sync cost is 2 $messages$ and the delay is 16 ms). Similarly, all other instances will follow the same procedure. The total synchronization cost for the instance NF1(8) is 4 $messages$ and the delay is 28 ms by using the paths $8 \rightarrow 13 \rightarrow 12$ (the sync cost is 2 *messages* and the delay is 10 *ms*) and 8 → 7 → 10 (the sync cost is 2 *messages* and the delay is 18 *ms*) respectively. The synchronization cost for the instance NF1(12) is 2 *messages* and the delay is 17 *ms*. As a result, the total synchronization costs between instances is the number of messages exchanged between them which is 8 *messages* and the delay is the amount of time needed to exchange these messages which is 61 *ms*.



a)

Step	Sync Operations	Sync Cost (*Message*)	Sync Delay (*ms*)
1	NF1(10) → NF1(8)	2	11
	NF1(10) → NF1(12)	2	7
	NF1(8) → NF1(10)	2	16
	NF1(8) → NF1(12)	2	10
	NF1(12) → NF1(8)	2	11
	NF1(12) → NF1(10)	2	7
Total		12	16

b)

Step	Sync Operations	Sync Cost (*Message*)	Sync Delay (*ms*)
1	NF1(10) → NF1(8)	2	16
2	NF1(8) → NF1(12)	2	10
3	NF1(12) → NF1(8)	2	17
4	NF1(8) → NF1(10)	2	18
Total		8	61

c)

Step	Sync Operations	Sync Cost (*Message*)	Sync Delay (*ms*)
1	NF1(10) → SyncFn(13)	1	3
	NF1(8) → SyncFn(13)	1	8
	NF1(12) → SyncFn(13)	1	2
2	SyncFn(13) → NF1(10)	1	3
	SyncFn(13) → NF1(8)	1	8
	SyncFn(13) → NF1(12)	1	2
Total		6	16

d)

Step	Sync Operations	Sync Cost (*Message*)	Sync Delay (*ms*)
1	NF1(10) → SyncFn(13)	1	3
	NF1(8) → SyncFn(8)	0	0
	NF1(12) → SyncFn(13)	1	2
2	SyncFn(8) → SyncFn(13)	1	8
	SyncFn(13) → SyncFn (8)	1	8
3	SyncFn(13) → NF1(10)	1	3
	SyncFn(8) → NF1(8)	0	0
	SyncFn(13) → NF1(12)	1	2
Total		6	14

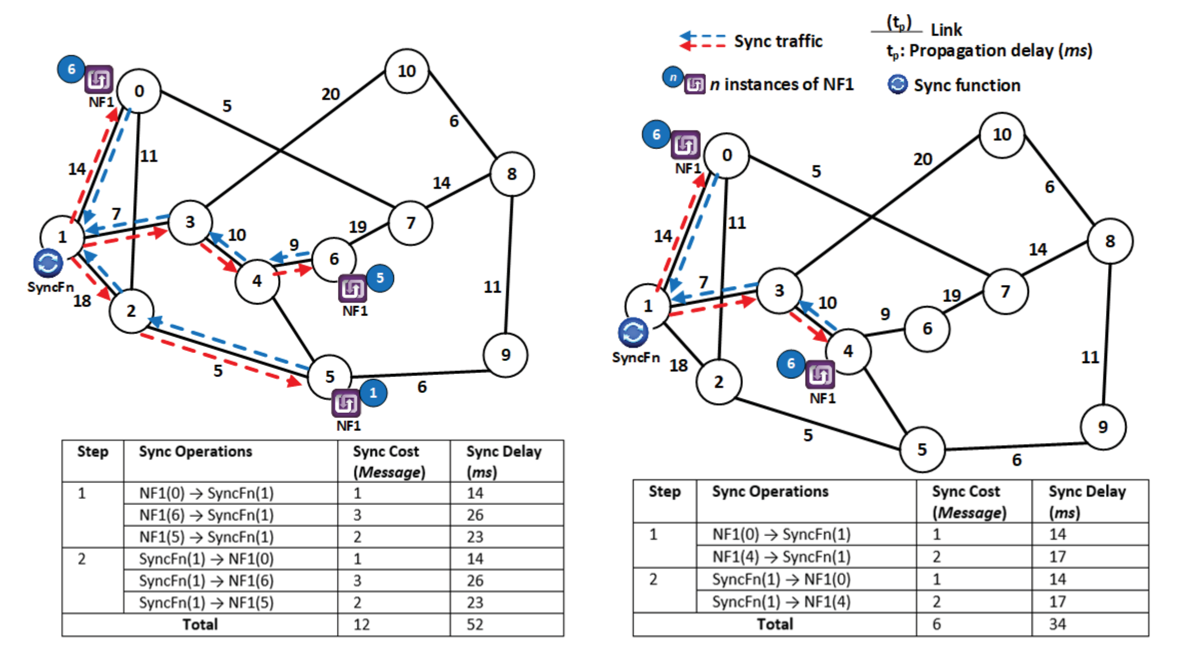
Figure 2.2 Different possible communication patterns to synchronize NF1 instances

Figure 2.2(c) illustrated a communication pattern that benefits from a single synchronization function. In this communication pattern, the synchronization operations are achieved in two steps. The first step, the instances synchronize the same state to the synchronization function periodically at the same time. The second step, the synchronization function consolidates and distributes the same state to the all instances periodically at the same time. For example, the instance NF1(10), NF1(8), and NF1(12) synchronize the same state to the synchronization function that is located in node 13, SyncFn(13). Then, the synchronization function SyncFn(13) consolidates and distributes the same state to all instances by using the same procedure. The synchronization cost and the delay for both steps is depicted in Figure 2.2(c). As a result, the total synchronization cost is the number of messages exchanged between instances and synchronization function which is 6 *messages* and the delay is the amount of time needed to exchange these messages which is 16 *ms*.

Figure 2.2(d) depicts a communication pattern that benefits from multiple synchronization functions. The synchronization function can work as follows: serve a set of instances, or serve a set of synchronization functions, or it can do both at the same time. In this communication pattern, the synchronization operations are achieved in three steps. The first step, the instances synchronize the same state to the closest synchronization function periodically at the same time. The second step, the synchronization functions synchronize the same state with each other periodically at the same time. In the third step, the synchronization functions distribute the same state to all instances periodically at the same time. For example, NF1(10) and NF1(12) synchronize the same state to SyncFn(13), also NF1(8) synchronizes the same state to SyncFn(8). Furthermore, SyncFn(8) and SyncFn(13) synchronize the same state with each other by using the same procedure. Finally, SyncFn(8) and SyncFn(13) distribute the same state to their instances by using the same procedure. The synchronization costs and the delay for all steps depicted in Figure 2.2(d). As a result, the total synchronization cost is the number of messages exchanged between instances and synchronization functions which is 6 $messages$ and the delay is the amount of time needed to exchange these messages which is 14 ms.

It is clear from the above-described example that the communication pattern significantly impacts the synchronization costs and delay and that synchronization functions could further reduce such costs and delay. In this chapter, we focus on how to find the optimal communication pattern to synchronize the instances, and the optimal placement for the synchronization functions in a way that minimizes synchronization costs (i.e., number of exchanged messages) while ensuring that the synchronization delay does not exceed a certain bound.

Figure 2.3 shows an example of instances of a function NF1 mapped into the physical infrastructure and it also shows the SyncFns that are provisioned to ensure the consistency among them, to minimize the synchronization costs and to ensure a bounded synchronization delay. As a practical example of that, we can see in Figure 2.3(a) that physical node 1 hosts one synchronization function, SyncFn(1), that is synchronized the state between instances of a function NF1 that are embedded in physical nodes 0, 5, and 6 (i.e., NF1(0), NF1(5), and NF1(6)). The instances NF1(0), NF1(5), and NF1(6) synchronize the same state to the synchronization function that is located in node 1, SyncFn(1). Then, the synchronization function SyncFn(1) consolidates and distributes the same state to all instances.



Step	Sync Operations	Sync Cost (*Message*)	Sync Delay (*ms*)
1	NF1(0) → SyncFn(1)	1	14
	NF1(6) → SyncFn(1)	3	26
	NF1(5) → SyncFn(1)	2	23
2	SyncFn(1) → NF1(0)	1	14
	SyncFn(1) → NF1(6)	3	26
	SyncFn(1) → NF1(5)	2	23
Total		12	52

a) Original mapping of the instances of a function NF1 and their SyncFns.

Step	Sync Operations	Sync Cost (*Message*)	Sync Delay (*ms*)
1	NF1(0) → SyncFn(1)	1	14
	NF1(4) → SyncFn(1)	2	17
2	SyncFn(1) → NF1(0)	1	14
	SyncFn(1) → NF1(4)	2	17
Total		6	34

b) Mapping after the migration of 5 instances from node 6 to 4 and 1 instance from 5 to 4.

Figure 2.3 Example showing how VNF instances migration could allow to further reduce the synchronization costs and delay

Some VNF instances may be far from the synchronization functions due to the lack of resources in the NFV infrastructure will cause the increase in synchronization costs and delay. For example, we can see in Figure 2.3(a) that there are instances of function NF1 hosted in node 6, NF1(6), that far from SyncFn(1). Furthermore, few number of instances hosted in physical node will cause the increase in synchronization costs and delay. For example, we can see in Figure 2.3(a) that there is one instance of function NF1 hosted in node 5, NF1(5), which increase the synchronization costs. Figure 2.3(b) shows that VNF instances migration could be leveraged to relocate the VNF instances that increased the synchronization costs and delay to other physical nodes, where they can get closer to SyncFns, therefore, can further reduce the synchronization costs and delay. The Figure 2.3(b) shows that the 5 instances of NF1 hosted in node 6 has been migrated to node 4, and thus, it is now closer to the SyncFn(1). In addition to that, the one instance of NF1 hosted in node 5 has been migrated to node 4, and thus, we aggregate them with other instances.

It is also worth noting that migration has a cost such as bandwidth, CPU consumption, and service interruption Boutaba, Zhang & Zhani (2014). We assume in this work that a migration could be carried out only if the number of hops between the original placement of VNF instances and the new one does not exceed a maximal number of hops (i.e., to ensure a minimal migration cost), and satisfy the delay (i.e., by migrating VNF instances to other physical nodes which can ensure the lowest synchronization delay and this delay should not exceed the threshold). For instance, the number of hops is limited to 1 in the example provided in Figure 2.3(b).

It is clear from the above-described example in Figure 2.3(b) that the VNF instances migration significantly impacts the synchronization costs and delay. To conclude, as illustrated in the aforementioned example, the main challenges addressed in this chapter is to leverage migration to further reduce the synchronization costs and to ensure a bounded synchronization delay.

2.4 Problem Formulation

In this section, we formulate the problem of finding the optimal communication pattern as an Integer Linear Program (ILP) aiming at minimizing the synchronization costs while ensuring the required synchronization delay does not exceed a certain bound. Furthermore, the ILP finds the optimal communication pattern to synchronize the instances, the number of the synchronization functions and their optimal placement of in the infrastructure. All symbols used in the formulation are summarized in Table 2.1.

The physical infrastructure consists of multiple nodes located in different geographical locations modeled by a graph $G = (N, L)$ where N indicates the set of physical nodes that are connected via physical links L. We also define w_{uv} is the value of the weight of the link (i.e., cost) between two physical nodes u and v. We define by d_{uv} the value of the synchronization delay of the link between two connected physical nodes u and v. We also denote by β the maximum synchronization delay that can be tolerated between any two VNF instances.

We define $I = \{1, ..., |I|\}$ as the set of physical nodes hosting the instances of the same network function that should be synchronized. Each node of the set I is considered as a source and a destination as it has to generate and receive data. Any subgraph that connects the nodes of the set I can be considered as a communication pattern. The weight of a subgraph is defined as the sum of the weights of the subgraph's links. This weight corresponds to the total synchronization costs of the communication pattern associated to that subgraph. As our ultimate objective is to minimize the total synchronization costs, the goal of the following ILP is to find the subgraph $\bar{G}$ extracted from the original graph G that necessarily includes all the nodes of I and having the minimum total weight. As such, the subgraph $\bar{G}$ corresponds to the optimal communication pattern.

The subgraph $\bar{G}$ can also be seen as a set of paths that connects all nodes of I while the weight of the resulting subgraph is minimal. The proposed ILP finds these paths while minimizing the weight of the generated subgraph. In the following, we define the decision variables and constraints of the ILP.

Decision Variables

We define two decision variables as follows:

- $x_{stuv} \in \{0, 1\}$ indicates whether the link uv ($u \in N$ and $v \in N$) is part of a path between a source $s \in I$ and a destination $t \in I$.
- $y_{uv} \in \{0, 1\}$ indicates whether the link uv is used in the path between any source $s \in I$ and any destination $t \in I$. In other words, y_{uv} is equal to 1 only if the link uv belongs to the subgraph $\bar{G}$.

Table 2.1 Symbols and their definitions

Symbol	Definition
N	Set of physical nodes in an infrastructure
L	Set of physical links in an infrastructure
w_{uv}	The value of the weight of the link between two physical nodes u and v
d_{uv}	The value of the synchronization delay of the link between two connected physical nodes u and v
β	The maximum synchronization delay that can be tolerated between any two VNF instances
I	The set of physical nodes hosting the instances of the same network function that should be synchronized
$\bar{G}$	The optimal communication pattern
x_{stuv}	Decision variable to indicate whether the link uv is part of a path between a source s and a destination t
y_{uv}	Decision variable to indicate whether the link uv is used in the path between any source s and any destination t

Objective Function

Our main objective is to minimize the synchronization costs among instances while ensuring the required synchronization delay does not exceed a certain bound. The objective function can be expressed as follows:

$$min \left(\sum_{u \in N} \sum_{v \in N} y_{uv} w_{uv} \right) \tag{2.1}$$

Constraints

While finding the optimal solution, several constraints must be satisfied. For instance, we need to ensure that there is a link leaving any source $s \in I$:

$$\sum_{t \in I} \sum_{v \in N} x_{stsv} - \sum_{u \in N} \sum_{t \in I} x_{stus} = 1 \qquad \forall s \in I \tag{2.2}$$

where the first term is the number of links leaving s, and the second one is the number of links entering to the node s.

We also need to ensure that there is a link entering any destination $t \in I$:

$$\sum_{u \in N} \sum_{s \in I} x_{stut} - \sum_{s \in I} \sum_{v \in N} x_{sttv} = 1 \qquad \forall t \in N \tag{2.3}$$

where the first term is the number of links entering t, and the second term is the number of links leaving t.

Furthermore, there must be a path between any source $s \in I$ and destination $t \in I$. In other words, we must ensure that for any node on the path between s and t, the number of links entering that node is equal to the number of links leaving it (as it is a path). This can be expressed as follows:

$$\sum_{u \in N} x_{stuz} - \sum_{v \in N} x_{stzv} = 0 \quad \forall s, t \in I \quad \forall z \in N \backslash \{s, t\} \tag{2.4}$$

where the first term is the number of links entering a node $z \in N \backslash \{s, t\}$ that lies in the path between the source $s \in I$ and the destination $t \in I$, and the second term is the number of links leaving $z \in N \backslash \{s, t\}$ in the same path.

Furthermore, we need to ensure that, if there are common links between the paths, they should be considered once in the constructed subgraph. In other words, a link uv should be counted only once in the generated subgraph. This can be expressed as follows:

$$y_{uv} \geq x_{stuv} \quad \forall s, t \in I \quad \forall u, v \in N \tag{2.5}$$

Additionally, the delay of the path between any source $s \in I$ and destination $t \in I$ should not exceed the maximum synchronization delay β. This constraint can be written as follows:

$$\sum_{u \in N} \sum_{v \in N} x_{stuv} d_{uv} \leq \beta \qquad \forall s, t \in I \tag{2.6}$$

ILP Output

The resulting values of the decision variable y_{uv} will define the links of subgraph $\bar{G}$ as well as its nodes. The variable x_{stuv} defines the path that should be used to synchronize data between nodes $s \in I$ and $t \in I$.

Furthermore, to find the number of the synchronization functions and their optimal placement, we assume that any physical node in $\bar{G}$ being connected to more than two links, called hereafter *a common node*, should host a synchronization function. This is because a node connected to only two links has only to forward the synchronized data from one link to the following. However, a node connected to more than two links is able to aggregate the data coming from two links and then forward it to the other ones. As a result, synchronization functions are placed in common nodes and have the same number as those nodes.

This problem is formally known as the Clustered Shortest-Path Tree Problem (CluSTP), and it has been demonstrated to be NP-hard Cosma, Pop & Zelina (2020); Bökler (2017). As a result, several efforts have been made to find effective heuristics for solving it Binh, Thanh & Thang (2019); Cosma, Pop & Zelina (2021). Because the problem of finding the communication pattern between multiple instances is a variant of the CluSTP problem, it is also NP-hard. To put it another way, we consider the communication pattern problem, which generalises the classical single-source shortest-path problem and seeks a spanning tree of a given graph with the property that each sub-graph induced by a cluster is connected and the total cost of the paths from a given source node to all other nodes in the sub-graph is minimised.

2.5 Proposed Heuristic

In this section, we describe three heuristics that could find the best communication patterns to synchronize data among the VNF instances. The first algorithm is called *Shortest Path Tree*

(SPT) as it tries to connect instances through the shortest paths among them and it does not use synchronization functions. The second algorithm is called *Single Synchronization Function Algorithm* (SSF) as it uses only one synchronization function in the physical infrastructure to serve multiple instances of the same network function. The third algorithm is called *Multiple Synchronization Functions Algorithm* (MSF) as it tries to place multiple synchronization functions, each serving a subset of instances of the same network function.

Furthermore, we put forward a new heuristic algorithm that leverages migration of VNF instances to move them closer to synchronization functions in order to further reduce the synchronization costs and ensure the requested synchronization delay. This algorithm, called Migration-based optimization, should be executed after running the MSF algorithm (denoted M-MSF) or the CPLEX optimizer (denoted M-CPLEX). In the following, we present the four proposed algorithms in detail:

2.5.1 Solution 1: Algorithm SPT

This algorithm is a modified version of the shortest path tree algorithm. Each instance synchronizes its state with the other ones periodically and at the same time using the shortest path tree that connect it to all the other instances.

We first process the network functions one by one. Assume we consider the first one, we consider its set of VNF instances. As shown in Algorithm 1, for each instance, we compute the shortest paths tree towards all the other instances. That is the tree for which the distance between the considered instance and all the others is minimal. Each of the nodes of the tree is configured to forward synchronization data originating from the root instance towards the leaves of the tree. This ensures that all data follow the shortest paths, and thereby synchronization costs is minimized. The messages coming from the instances are not modified throughout the path to the destination instances.

We also compute the worst-case synchronization delay, $t_{\bar{l}}$, which is the delay required to ensure synchronization messages were exchanged among all the instances. This delay corresponds to the path that has the highest delay in the tree (Line 5). This delay should not exceed the threshold of the synchronization delay β. The resulting synchronization costs for the tree and the total cost can be then easily computed (Line 6).

Finally, the whole process is repeated for the instances of each of the remaining network functions.

Algorithm 2.1 SPT

```
**Input:** $V$: set of VNFs
        $\beta$: delay constraint
        $t_{\bar{l}} \leftarrow 0$: worst-case synchronization delay
        $SynchCost \leftarrow 0$: synchronization delay

**Output:** The communication pattern

1 for $v \in V$ do
2     for each instance $i_v$ of type $v$ do
3         $SPTree(i_v) \leftarrow get\_shortest\_path\_tree(i_v)$
4         $Delay(i_v) \leftarrow compute\_synch\_delay(SPTree(i_v))$
5         $t_{\bar{l}} \leftarrow max(t_{\bar{l}}, Delay(i_v))$
6         $SynchCost \leftarrow SynchCost + compute\_synch\_cost(SPTree(i_v))$
7     end for
8 end for
```

2.5.2 Solution 2: Algorithm SSF

Algorithm 2 aims to find the optimal communication pattern to ensure synchronization among same-type instances but it is allowed to use a single synchronization function to support synchronization.

Similar to SPT, for each VNF v, we first compute the shortest path tree originating from each type-v instance to the others. Next, we identify the set of common nodes n_{cn} among all shortest path trees of the all the instances. Common nodes are the ones shared among all the shortest path trees. Each common node has a score $CNodeScore[n]$ defined as the number of shortest path trees using that particular node (Lines 2-7).

To minimize the synchronization costs, we choose the common node n that has the highest score $CNodeScore[n]$ and enough resources c_n to host the synchronization function (Lines 2-8). The rationale is that this node receives the highest amount of synchronization messages from all instances towards all instances. Hence, it is the best location to host a synchronization function (denoted by s_v) that can process these messages, e.g., merge their content, and then forwards such messages to the remaining instances.

We also compute t_v the longest delay between two instances of VNF type v going through the designated synchronization function s_v (Line 10). The delay t_v should not exceed the synchronization delay threshold β. The synchronization cost is also computed assuming all synchronization data goes through the synchronization function (Line 11). Finally, the whole process is repeated for the instances of each of the remaining network functions.

It is worth noting that we assume that all instances send messages to the synchronization function periodically and at the same time.

2.5.3 Solution 3: Algorithm MSF

In this third algorithm, our goal is to minimize the synchronization costs while it is allowed to use multiple synchronization functions to ensure data synchronization among instances of the same network function (as opposite to a single synchronization function for the SSF algorithm).

As shown in Algorithm 3, we consider network functions one by one. For a particular network function of type v, we identify first the shortest path tree $SPTree(i_v)$ originating from each instance i_v towards all the others of the same type.

Similar to SPT, for each VNF v, we first compute the shortest path tree $SPTree(i_v)$ originating from each type-v instance i_v to the others. Next, we identify the set of common nodes among all shortest path trees of the all the instances. Common nodes are the ones shared among all the shortest path trees. Each common node has a score $CNodeScore[n]$ defined as the number of shortest path trees using that particular node (Lines 2-7).

Algorithm 2.2 SSF

```
Input: $N$: set of physical nodes
       $V$: set of VNFs
       $c_n$: remaining capacity in each physical node $n \in N$
       $\beta$: delay constraint
       $SynchCost \leftarrow 0$: synchronization delay

Output: The optimal communication pattern

1  for $v \in V$ do
2      for each instance $i_v$ of type $v$ do
3          $SPTree(i_v) \leftarrow get\_shortest\_path\_tree(i_v)$
4          for $n \in SPTree(i_v)$ do
5              $CNodeScore[n] \leftarrow CNodeScore[n] + 1$
6          end for
7      end for
8      $s_v \leftarrow \{n | max_{n \in N}(CNodeScore[n])\ \ and\ \ c_n \geq 1\}$
9      $AllocateSynchFnct(s_v)$
10     $t_v \leftarrow compute\_synch\_delay(s_v)$
11     $SynchCost \leftarrow SynchCost + compute\_synch\_cost(sf_v)$
12 end for
```

For each instance i_v, the algorithm selects the closest common node (denoted by s_{i_v}), i.e., the one that has the lowest distance to the instance i_v in terms of number of hops h and delay $\bar{t}$ and has enough resources $c_{s_{i_v}}$ (i.e., that is the amount of available resources expressed in terms of number of VNF instances that could be hosted by the physical node) (Line 9). A synchronization function is hosted in this closest common node s_{i_v} to process synchronization data coming to the considered instance (Line 10).

We also assume that a physical node cannot host more than one synchronization function serving instances of the same network function in the same physical node. Thus, if the algorithm decides to place another synchronization function in the same physical node to process synchronization data for two difference instances, the two functions are merged into a single one.

The synchronization messages originating from instance i_v are then directed towards the associated synchronization function s_{i_v}. The synchronization functions send the consolidated synchronization messages using the shortest path trees towards the destination instances and eventually through other synchronization functions (if they are in the path). Finally, the highest synchronization delay t_v is then computed based on the resulting synchronization pattern and should not exceed the threshold β (Line 12). The synchronization costs is computed (Line 13). The whole process is repeated for the instances of another network function.

Algorithm 2.3 MSF

```
Input: $N$: set of physical nodes
       $V$: set of VNFs
       $c_n$: remaining capacity in each physical node $n \in N$
       $\beta$: delay constraint
       $SynchCost \leftarrow 0$: synchronization delay

Output: The optimal communication pattern

1  for $v \in V$ do
2      for each instance $i_v$ of type $v$ do
3          $SPTree(i_v) \leftarrow get\_shortest\_path\_tree(i_v)$
4          for $n \in SPTree(i_v)$ do
5              $CNodeScore[n] \leftarrow CNodeScore[n] + 1$
6          end for
7      end for
8      for each instance $i_v$ of type $v$ do
9          $s_{i_v} \leftarrow \{n \mid CNodeScore[n] > 1 \text{ and } min_{n \in N}(hops(i_v, n)) \text{ and } c_n \geq 1\}$
10         $AllocateSynchFnct(s_{i_v})$
11     end for
12     $t_v \leftarrow compute\_synch\_delay()$
13     $SynchCost \leftarrow SynchCost + compute\_synch\_cost(v)$
14 end for
```

2.5.4 Solution 4: Migration-based optimization

This algorithm is performed after one of the previously presented solutions in order to further reduce synchronization costs and delay. The algorithm finds first the VNF instances whose placement is incurring high synchronization cost and delay. In other words, it identifies the VNF instances that are far from synchronization functions, and hence they should be migrated to be located closer to these functions. Afterwards, the algorithm finds the candidate physical nodes able to host the identified instances to be migrated one by one. The candidate physical nodes should satisfy four constraints: (1) the number of hops between the new and the old location should not be more than one hop, (2) the overall synchronization cost after the migration should be the lowest possible and less than it was before the migration, (3) the synchronization delay should remain less than the threshold β and the total end-to-end delay of the service chain is less than the threshold σ, and (4) the capacity (i.e., free resource) of the candidate physical node should be enough to host the VNF instance.

Algorithm 2.4 Migration-based optimization

```
Input: Output of MSF or CPLEX for all VNFs
       $N$ : set of physical nodes
       $V$ : set of VNFs
       $Host(i_v)$ : physical node hosting instance $i_v$ of type $v$
Output: The optimal communication pattern after migration

for $v \in V$ do
    for each instance $i_v$ of type $v$ do
        $Candidate(host(i_v)) \leftarrow Identify_Candidate(host(i_v))$
        $Destination \leftarrow \{n \mid min_{n \in Candidate(host(i_v))}(compute\_synch\_cost(v, i_v, n))\}$
        $Migrate(i_v, destination)$
    end for
    $t_v \leftarrow compute\_synch\_delay()$
    $compute\_synch\_cost(v)$
end for
```

After running the MSF algorithm or CPLEX optimizer, we identify the set $Candidate(n)$ defined as the set of candidate physical nodes that have to satisfy the following constraints: 1) they could be reached from n within at most one hop, 2) they are closer than n to the synchronization function serving instance of type v hosted in n, 3) they satisfy the delay threshold σ if the instance is moved there, and 4) they have enough resources to host the new instance. The instance is then migrated to $n \in Candidate(n)$ such that the synchronization cost is minimized first and then the synchronization delay is minimized (Line 5). Note that function $compute\_synch\_cost(v, i_v, n)$

computes the synchronization cost for VNF instances of type v assuming i_v is placed in n rather than its original placement.

Finally, after migrating the VNF instances, the highest synchronization delay t_v is computed based on the resulting synchronization pattern and it should not exceed the threshold β (Line 7). The synchronization cost is computed (Line 8).

The whole process is repeated for the instances of the other network functions.

2.6 Simulation and Results

In this section, we evaluate the performance of the proposed algorithms through extensive simulations. We basically compare the performance of those algorithms (i.e., SPT, SSF, and MSF) with the optimal solution provided by CPLEX and compare the obtained synchronization cost, delay, as well as the algorithms' execution time. We also compare the performance of MSF with the solution provided by MSF after migration (M-MSF) and the optimal solution provided by CPLEX with the optimal solution provided by CPLEX after migration (M-CPLEX) and compared the obtained synchronization cost and delay.

2.6.1 Simulation setup

In order to evaluate the performance of the proposed solutions, we developed a C++ simulator that simulates the entire environment including the physical infrastructure and the embedded network functions and we implemented the four proposed algorithms. We considered a topology that was randomly generated with 24 physical nodes connected through 52 physical links. We assume that each physical node has different computing capacity randomly generated between 40 and 120 virtual machines. We assume that all virtual machines have the same resource capacities. We also assume that links have different propagation delays that were randomly generated between 5 ms to 20 ms. Other delays like queuing and transmission delays are not taken into account as they are not significant compared to the propagation delay especially in wide-area networks Barford, Donoho, Flesia & Yegneswaran (2002); Jurski & Wozniak (2009).

Furthermore, in our experiments, we assume that the VNF instances are already embedded. We considered eight different scenarios where, in each of them, the instances of the same network functions were randomly distributed in the infrastructure. We also fixed the required synchronization delay among the instances of the same VNF to be 110 ms.

2.6.2 Synchronization cost

Figure 2.4 shows the average synchronization cost found with the proposed algorithms compared to the optimal solution generated by CPLEX for each scenario. We can see that, for the eight scenarios, SSF and MSF outperforms SPT and generate a slightly higher number of messages compared to the optimal solution provided by CPLEX. This indicates that the communication patterns provided by SSF and MSF are close to the optimal ones.

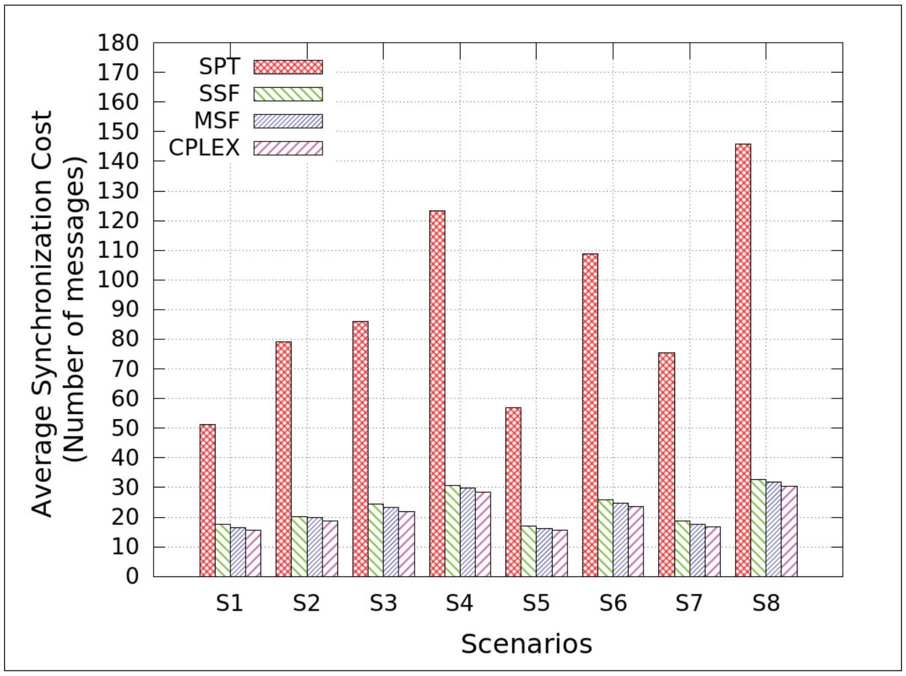


Figure 2.4 Average synchronization cost

Indeed, when SPT is used, each instance synchronizes its state with the other instances using the shortest path tree. In this case, the synchronization messages are duplicated at each node of the tree. Unlike SPT, SSF and MSF use synchronization functions consolidate received messages and then distribute the generated messages to all instances. As a result, the number of exchanged messages is further reduced.

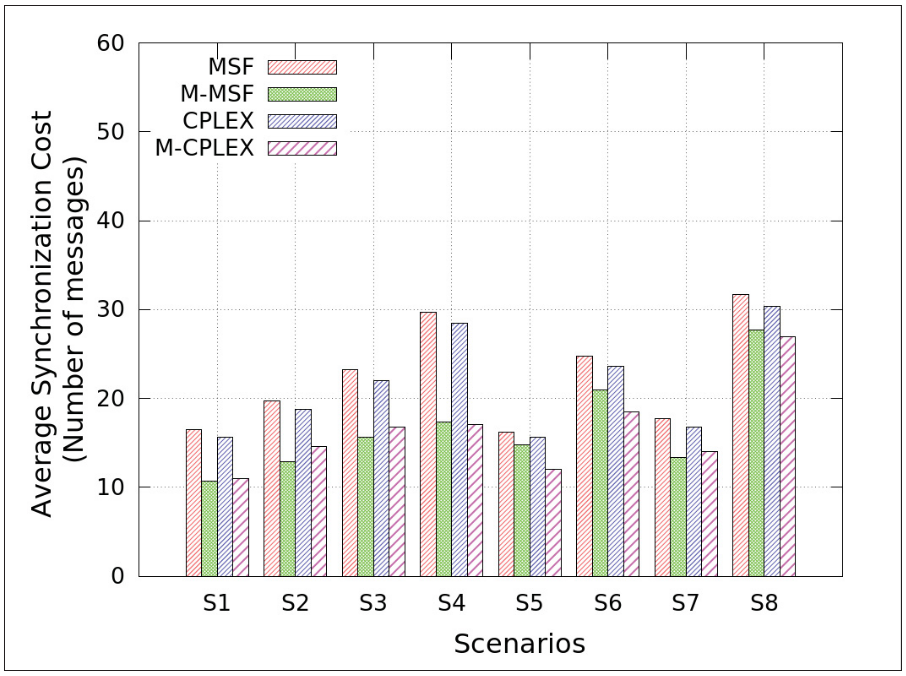


Figure 2.5 Average synchronization cost with and without migration

These costs are further reduced when migration is activated. For instance, Figure 2.5 compares the average synchronization cost found with MSF, CPLEX, with the ones found with migration M-MSF and M-CPLEX for all scenarios. We can notice that for all scenarios M-MSF and M-CPLEX outperform MSF and CPLEX. This indicates that the VNF instance migration mechanism has succeeded to significantly reduce synchronization costs. Indeed, M-MSF and M-CPLEX relocate the VNF instances closer to the synchronization functions, and therefore allow to further reduce the number of hops to reach them and the number of synchronization messages exchanged between the nodes.

2.6.3 Synchronization delay

Figure 2.6 compares the average synchronization delay found with the proposed algorithms and the optimal solution generated by CPLEX for each scenario. It clearly shows that SSF and MSF outperform SPT and provide a near optimal results. To explain these results, the synchronization delay can be controlled by the number of hops between the instances. Therefore, the more we minimize the number of hops, the more the synchronization delay decreases. Hence, SSF and MSF succeeded in reducing the synchronization delay by provisioning synchronization functions that minimize the number of hops between instances. The results show that, for SPT, SSF and MSF algorithms as well as CPLEX, the synchronization delay is below the total synchronization delay threshold β specified as input to all solutions (β is equal to 110 ms in our experiments).

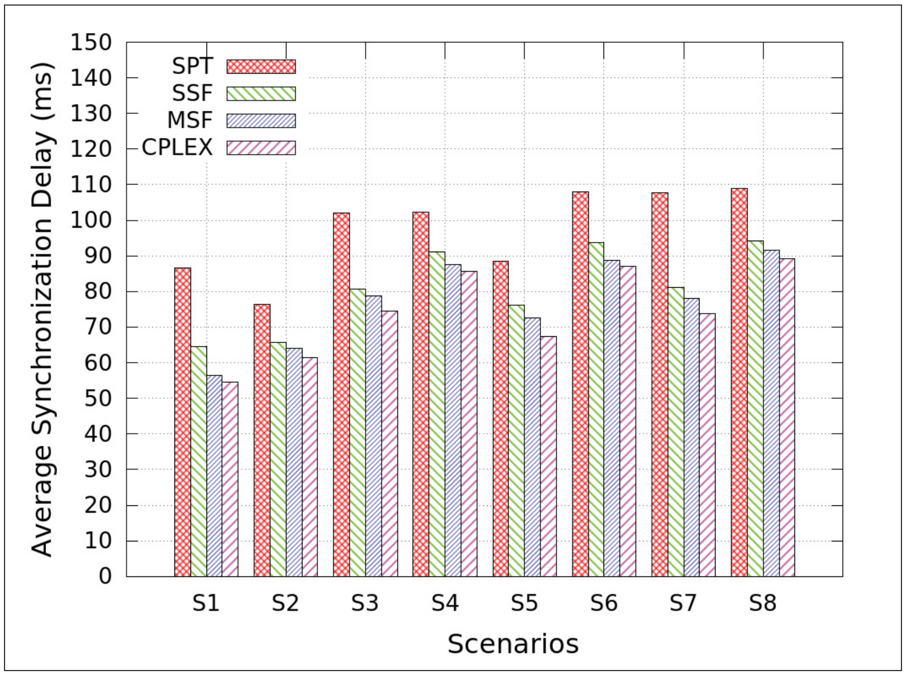


Figure 2.6 Average synchronization delay

We also study the benefit of instance migration. Figure 2.7 compares the average synchronization delay found with MSF and the optimal solution generated by CPLEX with the M-MSF and the optimal solution generated by M-CPLEX for each scenario. We can see that, for all scenarios, M-MSF and M-CPLEX outperform MSF and CPLEX. This is because M-MSF and M-CPLEX migrate the VNF instances to physical nodes ensuring lower synchronization delay.

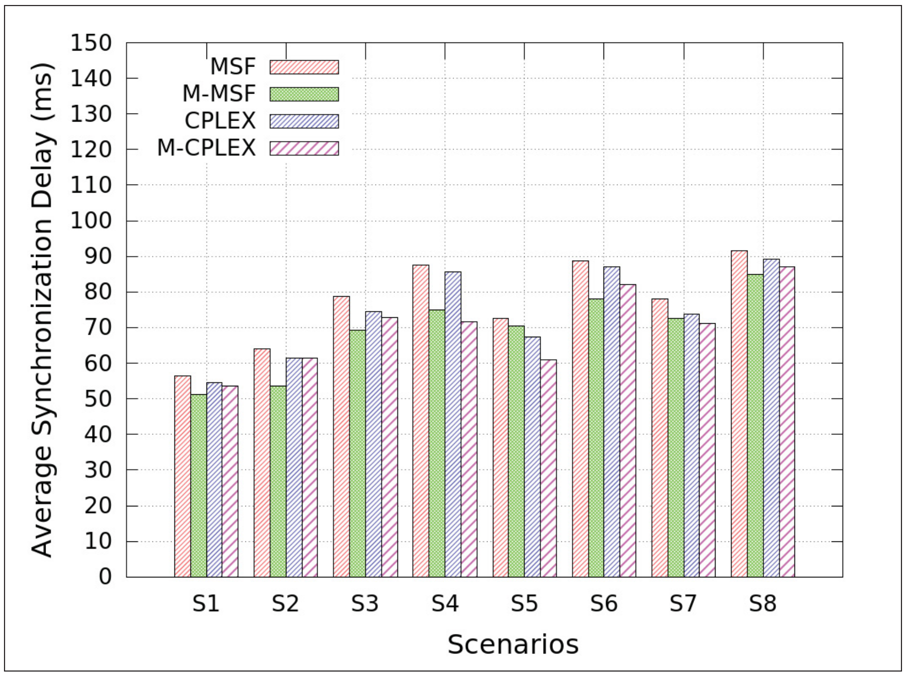


Figure 2.7 Average synchronization delay with and without migration

2.6.4 Execution Time

Figure 2.8 depicts the execution time for the proposed algorithms compared to that of CPLEX. The figure shows that the execution time for CPLEX is between 46 seconds and 29 minutes depending on the number of instances and the distance between them. The reason is that, when the number of variables in the ILP increases (e.g., number of network function instances), the problem becomes harder to solve and it takes more time to explore the solution space. The figure also shows that the execution time for SPT, SSF, and MSF does not significantly change for all scenarios and regardless of the number of instances and their placement.

2.6.5 Discussion

The performed simulations show that SSF and MSF are able to find near-optimal solutions minimizing synchronization cost and delay, while SPT is so far from optimal one. Indeed, the average synchronization cost and delay for SSF, MSF, CPLEX are up to 80% and 37% less than

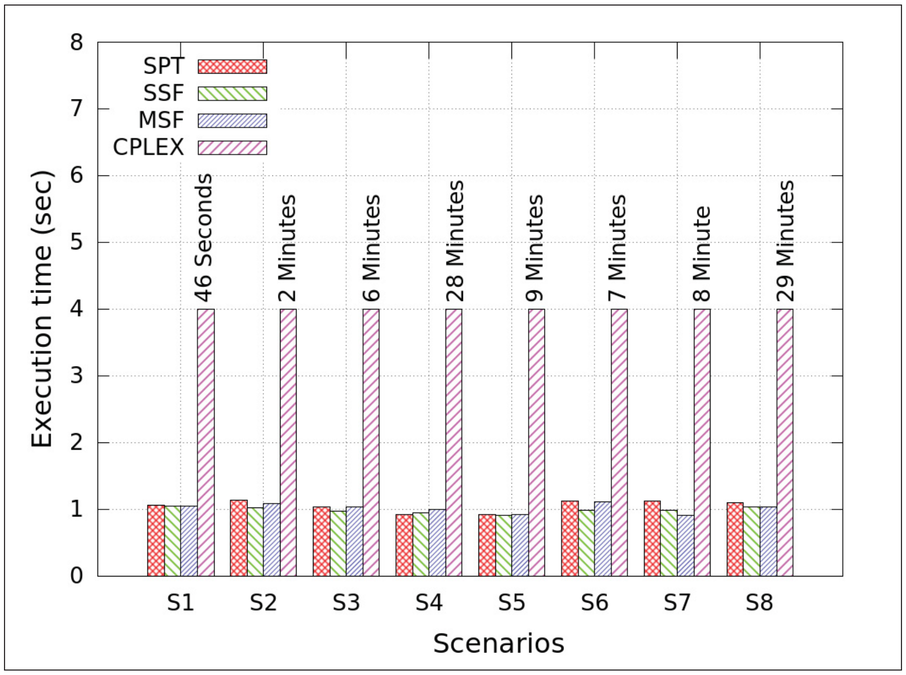


Figure 2.8 Execution time

the ones found with SPT, respectively. Furthermore, the execution time for the three proposed algorithms is stable and much lower than CPLEX. We can conclude that SSF and MSF find the best trade-off between complexity and performance with a slightly better performance for MSF.

Additionally, the performed simulations also show that instance migration further improves the results. Indeed, using migration reduces by up to 40% the average synchronization cost and by up to 12% the synchronization delay compared to SSF, MSF and CPLEX.

2.7 Conclusion

This chapter addresses the problem of finding the optimal communication pattern to synchronize data among VNF instances implementing the same VNF in a way that minimizes the synchronization costs and ensures that the synchronization delay does not exceed a certain bound. To this end, the problem was formulated as an Integer Linear Program implemented in CPLEX and three heuristic algorithms (i.e., SPT, SSF, and MSF) were proposed to deal with large-scale scenarios.

Though all the three algorithms show much lower computational time compared to CPLEX, MSF is the one that provides near optimal solution allowing to minimize synchronization costs and delay.

In addition to that, we proposed to use instance migration to further reduce the synchronization costs and delay by migrating the VNF instances closer to the synchronization functions. The conducted simulations showed that migration further improves the performance of MSF and CPLEX. Indeed, migration reduces by up to 40% the synchronization costs and by up to 12% synchronization delay.

In order to adopt the proposed solutions, we plan to study carefully the synchronization costs of different types of VNFs and to assess their synchronization requirements in terms of delay. Because a high synchronization delay among the instances of the same type of VNF could impact the performance of the function itself (e.g., late attack detection in the case of an IDS). This is why we assume there is a bound on the synchronization delay (i.e., the highest acceptable delay for the synchronization) that should be specified based on the function type and requirement.

CHAPTER 3

ON ENSURING COST-EFFICIENT SURVIVABILITY OF SERVICE FUNCTION CHAINS IN NFV ENVIRONMENTS

3.1 Introduction

Network Function Virtualization (NFV) is transforming traditional networks into a platform allowing to dynamically instantiate virtual network functions (VNFs) implemented in software regardless of the underlying hardware. Hence, NFV technology allows to provision services in the form Service Function Chains (SFCs) where an SFC is an end-to-end network services composed of an ordered collection of VNFs defined based on the service provider's need and connected through virtual links Halpern & Pignataro (2015).

In this context, a large body of work has recently addressed resource provisioning and management of such SFCs Bari *et al.* (2012), Herrera & Botero (2016), Mijumbi *et al.* (2015). The majority of current research make the assumption that the physical infrastructure is always available, which is unrealistic given the prevalence of failures in cloud network infrastructures Lee *et al.* (2017), ISG (2014), Zhang *et al.* (2014). Indeed, as the VNFs of a chain are deployed on different servers, probably in different locations, they are easily affected by physical failures which could be multiple and simultaneous. These failures might easily bring down a large number of VNFs, breaking a number of SFCs and making their associated services unavailable. A downtime, even for a very short period, could damage service provider's reputation, and, depending on the type of the supplied service, could result in significant revenue losses (e.g., downtime could cost $5,600 per minute and up to $300,000 per hour according to Cohen).

In the literature, there are two basic strategies for achieving service survivability: proactive and reactive Zhani & Boutaba (2015). In proactive strategies, we provision and reserve backup resources for each SFC early on during the mapping process. As a result, this strategy speeds up recovery; however, the backup resources remain idle until failure occurs Yu *et al.* (2011), Ayoubi *et al.* (2016b), Chowdhury *et al.* (2016), Fan *et al.* (2015). This means that resources are booked and paid. The second strategy in the literature is the reactive one, in which we provision backup resources in the event of a failure Bo *et al.* (2014), Xiao *et al.* (2014). As a result, there is an extra delay to provision resources and set up the new backups before they can take over the service. The first strategy has a larger backup footprint even if it promises speedy recovery from failures. Although the second strategy has the smallest backup footprint, it does not necessarily ensure thorough and speedy service recovery following failures. In light of this, proactive solutions are increasingly desirable, even if they incur additional operational costs to provision and maintain the backup VNFs.

In order to address this issue and reduce resource waste, various research efforts recommended the proactive strategy proactive using shared backup VNFs Ayoubi *et al.* (2016b),Yu *et al.* (2011), Aidi *et al.* (2018). In this case, several VNFs share the same backup, and hence, if one of them fails, the backup replaces it and takes over. Unfortunately, this works only when we assume only a single failure happens at a time, and hence, if multiple VNF failures occur at the same time, only one or a subset of VNFs could recover. The others will remain non-available and not replaced.

Unlike previous work, this research looks into potential ways to provision enough backups in a proactive manner to handle multiple and simultaneous failures. The proposed solutions analyze and predict the amount of traffic processed by each VNF in order to identify the traffic rate fluctuation and provision backups based on these fluctuations rather than the current number of VNF instances. Hence, if a low demand is detected, the resources for the backup is automatically decreased and vice-versa. As a result, we ensure that there is always enough VNF backups to handle all the demand of all VNFs to be able to overcome potential multiple failures. By that means, it is possible to minimize backup resources while at the same time being ready for all failures.

The solution presented in this work was integrated into a Survivability Management Framework that ensures the survivability of all embedded SFCs by adjusting the number of VNF backups for each type of VNF based on the predicted traffic rates over time. It dynamically provisions VNF backup instances for each type of VNF and adjusts their number and location (through VNF migration when it is needed). In summary, the contributions of this paper can be summarized as follows:

• We put forward a traffic prediction model based on the ARIMA model Zhani, Elbiaze & Kamoun (2009) to forecast demand traffic rates for each of SFC.

• We formulate the backup provisioning problem as an Integer Linear Program (ILP) aiming at minimizing the number and the location of shared backups for each type of VNF and minimizing the amount of backup resources and operational costs (i.e., costs of running the VNF backup instances, of management including instantiation and migration, and of data synchronization between a VNF and its backup).

• We have developed two algorithms more adapted to search the solution space for large-scale problems. The first one is a baseline algorithm, called *the 1-1B Algorithm*, where one backup instance is provisioned for each VNF instance. The second algorithm, called *the Adjusting & Sharing Backups Algorithm* (ASB), is more sophisticated and more efficiently manage backups such that their number and their operational costs are minimized.

• Furthermore, we evaluate the effectiveness of the ASB algorithm compared to the Baseline for different scenarios. We show that it could also provide near optimal solutions that are close to the ones obtained with the proposed ILP (computed using the CPLEX optimizer IBM (2020)).

• Finally, we show that using demand prediction could further reduce backup operational costs incurred by the ASB algorithm by reducing the number of backup VNFs that are needed to ensure survivability.

The remainder of the chapter is structured as follows. Section 3.2 presents a deeper description of the survivability problem of service function chains. Section 3.3 provides a general overview of the proposed survivability management framework. Section 3.4 introduces the traffic prediction model. Section 3.5 provides the mathematical description of the problem. We then present the details of the proposed heuristic solutions in Section 3.6. Section 3.7 presents the experimental setup and results. Finally, Section 3.8 concludes the paper and provides some avenues for future work.

3.2 Problem Description

A service function chain is a collection of chained virtual network functions that should process incoming traffic from and to a preset source and destination, respectively Mijumbi *et al.* (2015). A VNF could be any network function, and hence, we could have different types of VNFs including a Firewall, a load balancer, an IDS and a NAT. Usually, a VNF is implemented in software that could be run on a Virtual Machine (VM) or a container hosted in a standard or dedicated server. The VNFs belonging to the same chain are linked together by logical/virtual links with enough bandwidth. To deploy SFCs, service providers use an SFC provisioning schemes that embed the SFCs into the physical infrastructure, that is to allocate the resources needed to run and deploy the VMs running the VNFs and allocate paths to map the virtual links connecting the VNFs ISG (2014); Zhani & ElBakoury (2020).

In contrast, service chain 2 is made up of two VNFs of type 1 and type 2 that are embedded in physical nodes 4 and 6, respectively, and it has incoming traffic from physical node 3 towards physical node 8. The last service chain is service chain 3, which is made up of two VNFs of type 1 and type 3 embedded in physical nodes 14 and 15, and which receives incoming traffic from physical node 13 towards physical node 10.

Once SFCs have been embedded into the physical infrastructure, the network operator face the challenge of ensuring that they are survivable to potential failures that could be multiple and could occur simultaneously. In other words, SFCs must be able to survive unexpected network failures. However, failures are common with physical nodes, and a single or multiple node failure could bring down a large number of VNFs, resulting in the loss of several service chains Zhani & Boutaba (2015).

In this chapter, we propose a solution allowing to have enough backups for each type of VNF to handle the traffic before the failure occurs. We therefore propose to provision backups based on the demand (incoming packet rate) to provision shared backups to ensure the survivability of all SFCs affected by potential single or multiple node failures.

To do so, for a VNF of type k, a backup VNF of the same type should be provisioned as the backup must contain the exact same software stack as the primary one. Furthermore, the provisioned backups of VNFs type k should have enough processing capacity to handle all the traffic processed by all primary VNFs of that type.

• **Provisioning shared backups:** Figure 3.1 depicts three service function chains, SFC1, SFC2 and SFC3, that were mapped into the physical infrastructure (colored in blue). The figure shows how VNFs of each chain are embedded from the chain's source to destination. For instance, service chain SFC1 is made out from two VNFs, the first one of type 1 whereas the second of type 2, that are embedded in physical nodes 1 and 12, respectively. The chain carries inbound traffic originating from the source, physical node 2, towards the destination, physical node 9.

Figure 3.1 also shows provisioned shared backups could be located. For instance, physical node 5 hosts a VNF type 1 backup that is shared between the VNF type 1 of chain SFC1 and that of SFC2. If physical node 1 fails, VNF1 of chain SFC1 goes down, the backup VNF1 in node 5 takes over and replaces the failing function. Similarly, if physical node 4 fails, the same shared backup could take over VNF1 of SFC2. If both VNFs of the two chains fail at the same time, it is possible for the backup VNF type 1 placed in node 5 to take over the two failures if it has enough resources to handle the demand received by the two VNFs. In the figure also, physical node 7 hosts a VNF type 2 backup that is shared between VNF types 2 of chains 1 and 2. In physical node 12 fails, VNF2 of chain SFC1 goes down, the backup VNF2 housed in node 7 takes over and replaces the failing function. If physical node 6 fails, the same shared backup could take over VNF2 of SFC2. It is obvious to see that the three service chains depicted in the figure are survivable to simultaneous node failures if enough backup VNFs are provisioned to handle the demand.

Additionally, to make sure that Backup VNFs are ready to take over the service in the case of failure, they should be updated regularly with the last state of the primary VNFs Alomari, Zhani, Aloqaily & Bouachir (2020); Boutaba *et al.* (2014). Figure 3.1 shows the state synchronization between each primary and backup VNFs (see green dashed arrows). As an illustration, the backup VNF1 housed in physical node 5 has the states of the primary VNF1 hosted in physical nodes 1, 4, and 14. When a failure occurs, the backup will utilise the most recent state of the failed function. A different level of state synchronization can be used. For example, at the virtual machine level performing the function Boutaba *et al.* (2014) or by employing customized synchronization scripts based on the type of network function Gray *et al.* (2017).

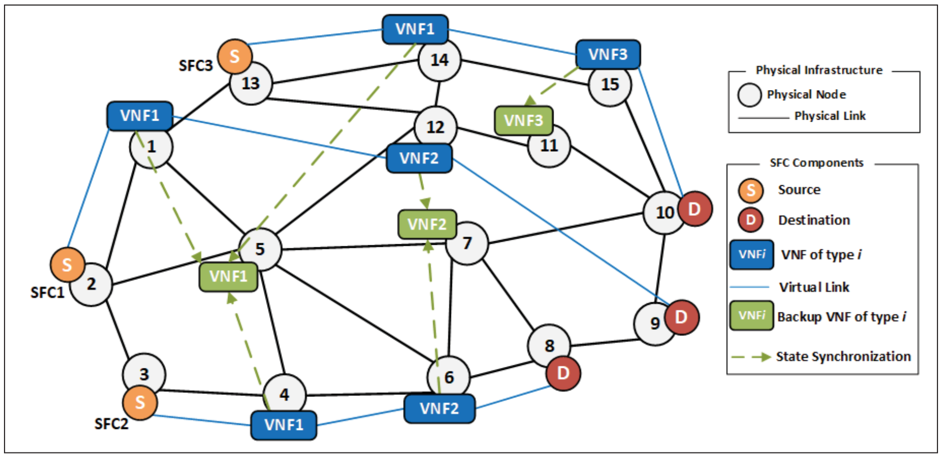


Figure 3.1 Deployment of service function chains and shared backups

In order to guarantee effective state synchronization, the delay between a VNF and its backup should not go over a specific threshold that should be defined based on the nature of the VNF and its requirements Alomari *et al.* (2020). Furthermore, state synchronization is costly as it consumes bandwidth in the path separating the primary and the backup VNFs Alomari *et al.* (2020), Moufakir, Zhani, Gherbi, Aloqaily & Ghrada (2022). In our work, we try to minimizing such synchronization costs by minimizing the number of hops between each VNF and its backup and by taking into consideration the monetary costs of using bandwidth of each link.

• **Adjusting shared backups:** as traffic demand within service function chains could fluctuate over time Aben (2014), Sandvine (2015), the number of shared backups could be adjusted accordingly. Assuming that traffic increases in chain 1 or 2 and the backup VNF type 1 hosted in node 5 is not able to handle the new demand in case of failure, we must increase the number of shared backups. For instance, Figure 3.2 shows how a second backup VNF type 1 has been provisioned in physical node 5 in order to be able to handle the total amount of traffic destined to the three VNFs type 1 belonging to three chains. If the demand decreases over time, we can go back to the first configuration (i.e., Figure 3.1) to reduce backups costs.

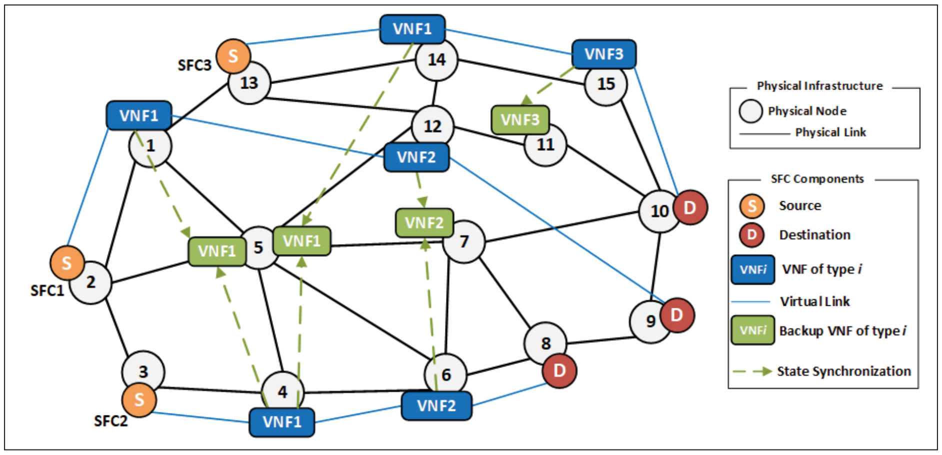


Figure 3.2 Adjusting shared backup

This adjustment process over time of the number of backups and their locations incurs management costs for the instantiation, migration and the deletion of the VNF backups. These costs have to be taken into account and minimized.

This chapter focuses on how to find the minimal number of shared VNF backups estimated based on the demand of the provisioned SFCs and how to determine their optimal placement in the infrastructure in a way that minimizes operating costs (i.e., running, management, and synchronization costs).

3.3 Survivability Management Framework

This part introduces a management framework with a survivability module. The framework is shown in Figure 3.3 and consists of the following modules:

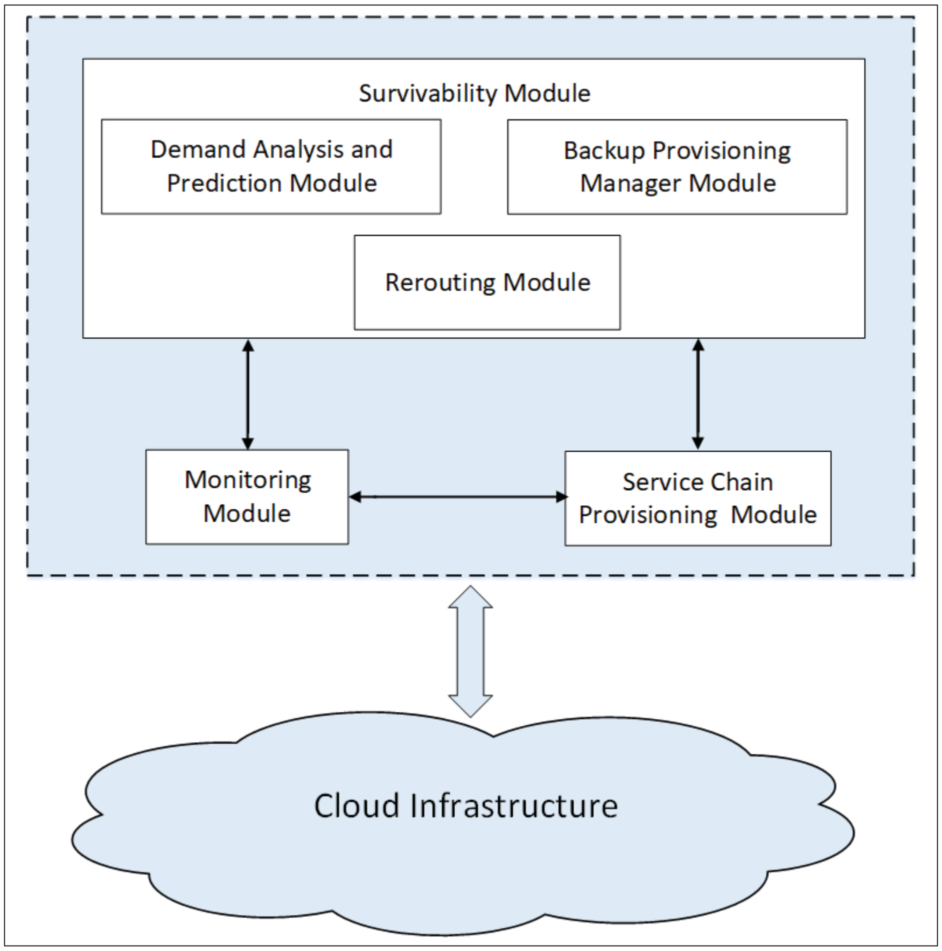

Survivability Module
Demand Analysis and Prediction Module
Backup Provisioning Manager Module
Rerouting Module
Monitoring Module
Service Chain Provisioning Module
Cloud Infrastructure

Figure 3.3 Survivability Management Framework.

• **Demand analysis and prediction module**: This module is in charge of keeping track of the packet arrival rate (demand) of each SFC in order to analyze the inbound and outbound traffic rates fluctuations for each VNF in the SFC. This module is also in charge of predicting future traffic rate of each SFC.

• **Backup provisioning manager module**: This module is in charge of determining the bare minimum number of shared backups required to cope with the predicted traffic rate in order to guarantee the survival of the embedded service chains, and determining their locations. This module also creates backup instances, shared backups, adjusts their number and placement, and creates the synchronization links needed to keep them up-to-date.

• **Rerouting module**: This module is responsible for redirecting the traffic intended to the failing VNFs towards the backup VNFs in the event of a failure.

• **Monitoring module**: This module is responsible for continuously monitoring the physical nodes and links of the infrastructure and giving real-time information about the state of the resources to other modules. When a failure is discovered, the monitoring module alerts the survivability module, and the latter responds by reducing the fault and maintaining service continuity.

• **Service chain provisioning module**: This module is in charge of allocating resources to service function chains and creating the virtual machines or containers required to run network functions. It also uses the SDN controller to provision the required bandwidth and to program the appropriate forwarding rules into the switches in order to direct traffic across the VNFs of each service function chain. The design of this module should be noted as being outside the purview of this work. Numerous studies have been conducted to design and develop such module, and any of the existing solutions may be used Luizelli *et al.* (2015), Herrera & Botero (2016), Racheg *et al.* (2017).

In this work, we focus on the design and implementation of the demand analysis and prediction module and the backup provisioning manager module. We therefore present first the used prediction model followed by the formulation of the backup provisioning problem and then lay out proposed heuristic solutions to handle the problem.

3.4 Traffic Prediction

As previously stated, the demand analysis and prediction module is in charge of monitoring the arrival of demands as well as their traffic rate. Additionally, it is responsible for predicting the demand (traffic rates) in the next time slots. Currently, we have used the well-known ARIMA model to develop a time series-based predictor Box, Jenkins, Reinsel & Ljung (2015) Zhani *et al.* (2009).

Once the predictions are determined, the next step is to determine the number of shared backups of each type of VNF required in the next time interval. To solve this issue, we determine how many shared backups are necessary to support the traffic rate for each demand. In Survivability Management Framework, this is performed by the backup provisioning manager module. The backup provisioning manager module determines the minimum number of shared backups per demand such that these backup instances are sufficient to meet the traffic demands. In other words, based on the traffic rate prediction of the ARIMA model, the backup provisioning manager module adjust the number of shared backups to minimize the overall cost and to make sure that there is enough backup instances before the failure happens.

3.5 Problem Formulation

In this section, we formulate the backup provisioning problem as an Integer Linear Program (ILP) with the goal of minimizing the number of shared backups and placing them in the physical infrastructure in order to ensure minimal backup operational costs (running, management, and synchronization costs). All symbols used in the formulation are summarized in Table 3.1.

The physical infrastructure is represented as a graph defined by $G = (N, L)$, where N is the collection of physical nodes connected by a set of physical links L. The computing capacity of each physical node $n \in N$ in the network is c_n. The computing capacity c_n is the maximum capacity that the physical node $n \in N$ can supply for VNF instances. To make things simple, we assume that each VNF instance runs on a single standard-sized virtual machine (i.e., all virtual machines are assumed to have the same resource capacity). We can therefore define c_n as the maximum number of virtual machines that can be deployed in physical node n. The minimum number of hops between physical nodes i and n is denoted by d_{in}.

We define $K = \{1, ..., |K|\}$ as the set of different types of VNFs that could be deployed. The number of VNF instances of type $k \in K$ embedded in physical node i is denoted by q_i^k. We denote by α_i^k the rate of the traffic received by all VNFs type k hosted in physical node i (i.e., q_i^k).

We also denote by P_k the processing capacity of the considered standard virtual machine instance when it is running VNF type k. It is expressed in terms of number of packets per second.

In addition, we assume that each SFC I can be described by the triplet $\{o_I, t_I, \gamma_I\}$, where $o_I \in N$ is the source node of the chain, $t_I \in N$ is the destination node of the chain, and γ_I represents the traffic demand that needs to go through the VNF instances of the chain in the right order.

Furthermore, we define only one decision variable denoted by $y_{in}^k \in \{0, 1\}$, which indicates whether the physical node n is hosting the backup instance allocated for the VNF of type k embedded in the physical node i.

In the following, we go over the various costs involved when provisioning backup instances as well as the objective function and the set of constraints that must be met.

• **Running costs**: The cost of running a single backup instance of type k in physical node n is denoted as ϕ_n^k and expressed in \$ per hour. As a result, the total running costs of all backup instances can be expressed as the follows:

$$\mathbb{R} = \sum_{i,n\in N} \sum_{k\in K} \phi_n^k \, y_{in}^k \tag{3.1}$$

• **Management costs**: Management costs are costs incurred when instantiating new backups. These include the costs of uploading/migrating an image of a virtual machine, and accommodating it in the new physical host.

Let η_n^k be the number of new backup instances of type k deployed on the physical node n at the current timeslot. It can be computed as follows:

$$\eta_n^k = max\{0, \sum_{n\in N} y_{in}^k - \sum_{n\in N} \overline{y}_{in}^k\} \quad \forall i \in N \quad \forall k \in K \tag{3.2}$$

Table 3.1 Symbols and their definitions

Symbol	**Definition**
N	Set of physical nodes in the physical infrastructure
L	Set of physical links in the physical infrastructure
c_n	Computing capacity of physical node n
d_{in}	Minimum number of hops between physical nodes i and n
K	Set of different types of VNFs
q_i^k	Number of VNF instances of type k embedded in physical node i
P_k	Processing capacity of the instance (in packets per second)
o_I	Source node of SFC I
t_I	Destination node of SFC I
γ_I	Traffic demand for SFC I
α_i^k	Traffic rate destined to all VNFs type k hosted in physical node i
y_{in}^k	Decision variable to indicate whether the physical node n is hosting the backup instance allocated for the VNF of type k embedded in the physical node i
$\overline{y}_{in}^k$	The number of backup instances of type k hosted in physical node n to backup all or some primary VNFs placed in node i but at the former timeslot
$\mathbb{R}$	Total running costs
$\mathbb{M}$	Total management costs
$\mathbb{S}$	Total synchronization costs
$\mathbb{O}$	Total operational costs
ϕ_n^k	Cost of running a single backup instance of type k in physical node n
η_n^k	Number of new backup instances of type k deployed in the physical node n at a point of time interval
φ_n^k	Cost of managing one backup instance of one VNF type k deployed in physical node n
b_{in}^k	Bandwidth required to perform the synchronization between the primary VNF of type k hosted in physical node i and its backup in a physical node n
δ_{in}	Bandwidth unit cost in a physical link (i,n) expressed in dollars per Mbps

where y_{in}^k is a decision variable that represents the number of backup instances of type k that should be provisioned in the current timeslot hosted in physical node n to backup all or some primary VNFs hosted in physical node i. The variable $\overline{y}_{in}^k$ is an input to the ILP and it represents the number of backup instances of type k hosted in physical node n to backup all or some primary VNFs placed in node i but at the former timeslot.

The cost of creating/migrating one backup instance for VNF type k is denoted by φ_n^k and expressed in \$ per hour. Therefore, total costs of management and creation of all backup instances in the network at the current timeslot can be expressed as follows:

$$\mathbb{M} = \sum_{i,n \in N} \sum_{k \in K} \varphi_n^k \, \eta_n^k \tag{3.3}$$

• **Synchronization costs**: It is expressed by the rate at which data is exchanged between the primary VNF instance in physical node i and its backup in physical node n. This cost is stated as follows:

$$\mathbb{S} = \sum_{i,n \in N} \sum_{k \in K} y_{in}^k \, b_{in}^k \, \delta_{in} \, d_{in} \tag{3.4}$$

where y_{in}^k indicates whether or not the primary VNF of type k embedded in a physical node i has backup in a physical node n. The input b_{in}^k represents the bandwidth required to perform the synchronization between the primary VNF of type k hosted in physical node i and its backup in a physical node n. Furthermore, δ_{in} represents the bandwidth unit cost in a physical link (i, n) expressed in dollars per Mbps. The value d_{in} represents the number of hops between the primary VNF hosted in physical node i and its backup in physical node n.

• **Objective Function**: Our primary goal is to minimize the operational costs incurred by backup instances while meeting all of the constraints listed below. This is accomplished by reducing the overall number of shared backups in the physical infrastructure. The objective function can hence be written as follows:

$$\mathbb{O} = \min_{(y_{in}^k)_{k\in K, i\in N, n\in N}} (\mathbb{R} + \mathbb{M} + \mathbb{S}) \tag{3.5}$$

• **Constraints**: A variety of constraints must be considered in order to find feasible solutions. For instance, we must make sure that the primary VNF and its backup instance are not located on the same physical node i. This can be stated in the following way:

$$y_{ii}^k = 0 \quad \forall i \in N \quad \forall k \in K \tag{3.6}$$

We must also ensure that any VNF of type k embedded in physical node i have its backup instance in a different physical node. It can be stated as follows:

$$\sum_{n\in N} y_{in}^k \geq 1 \quad \forall i \in N \quad \forall k \in K \tag{3.7}$$

Furthermore, to provision backup instances to process the traffic α_i^k destined to VNFs type k placed in physical node i (regardless of the number of VNFs type k in node i and to which SFC they belong to), we must provision backup instances that have processing capacity higher than or equal to α_i^k. The following equation captures this constraint:

$$\alpha_i^k \leq P_k \sum_{n\in N} y_{in}^k \quad \forall i \in N \quad \forall k \in K \tag{3.8}$$

Finally, to ensure that each physical node has enough resources to host the backup instances, each physical node n must meet the following capacity constraint:

$$\sum_{k\in K} q_n^k + \sum_{k\in K} y_{in}^k \leq c_n \quad \forall i, n \in N \tag{3.9}$$

The first term denotes the number of VNFs housed in physical node n, the second term is the number of backup instances hosted in the same physical node n, and the third term represents the amount of capacity that is currently available at physical node n.

This ILP should be solved as each time slot to regularly update and adjust the number and location of the backups. Except the variable y_{in}^{k}, which is a decision variable, all the others are input to the ILP.

This problem is formally known as the Virtual Network Embedding problem (VNE), and it has been demonstrated to be NP-hard Andersen (2002). As a result, several efforts have been made to find effective heuristics for solving it Zhani, Zhang, Simona & Boutaba (2013) Chowdhury, Rahman & Boutaba (2011). Because the Survivable Virtual Network Embedding (SVNE) problem is a variant of the VNE problem, it is also NP-hard. To put it another way, SVNE problem generalizes a minimum knapsack problem Zhani *et al.* (2013). Thus, we adopt greedy algorithms to solve it in next Section.

3.6 Proposed Heuristic

In this section, we outline two heuristic algorithms to solve the survivability problem of service function chains. We refer to the first algorithm as *One-to-One Backup* (1-1B) as it provision backups by considering the number of VNF instances (rather than the demand). In other words, we provision as many backup instances as primary ones to handle all failures (single or multiple node failure). The second algorithm is called *Adjusting & Sharing Backups Algorithm* (ASB) as we are looking into the demand to provision less backups that are shared but have enough resources to handle all potential failures (single or multiple node failures). In other words, we use the prediction of the demand to estimate the number of backups and thereby the number of backups is reduced compared to 1-1B. Furthermore, the ASB Algorithm adjusts dynamically the number of shared backups for each type of VNF and their placement based on the predicted demand and takes into account backup management costs (e.g., instantiation, migration). The details of the two proposed algorithms are provided below.

3.6.1 Solution 1: Algorithm 1-1B

The 1-1B algorithm is provided in Algorithm 1. It allocates a backup instance for each VNF instance. This means that 1-1B provisions as many backups as the number of original VNFs for each type of VNF. All nodes in the physical infrastructure are assumed to be capable of hosting backups any type of VNF. The main challenge that remains is to identify the location of these backups such that the operational costs.

To do so, the Algorithm considers VNFs types one by one. For each of them, it calls the function $MinOpCost$ (Function $MinOpCost$ in Line 3) that returns physical nodes n_{min} and m_{min} such that the backup instantiation of backups for VNFs type k hosted in m_{min} in physical node n_{min} would generate the lowest operational costs among. In other words, no lower operational costs could be incurred from any backup instantiation in any node $n \in N$ to backup VNFs hosted in any node $m \in N$.

The Function $MinOpCost$ is called until all VNFs type k are provisioned (i.e., $ListBackedUpVNF(k)$, the set of VNFs that have already backups provisioned, becomes equal to $ListVNF(k)$, the set of all VNFs type k hosted in the infrastructure), or there is no enough resources to provision backups (i.e., a_n, the amount of available resources in physical node n, is equal to zero for all physical nodes) (Line 2).

Algorithm 3.1 Algorithm 1-1B

```
Input: $N$: set of physical nodes
       $K$: set of VNF Types
       $a_n$: available resources in physical node $n$

Output: Provisioning backups

1 for each VNF type $k \in \{1, 2, 3, ...K\}$ do
2     while ($ListBackedUpVNF(k)! = ListVNF(k)$) or ($a_n = 0$) do
3         $n_{min}, m_{min}, ListVNF[m_{min}, n_{min}, k] \leftarrow MinOpCost(k, d_{max})$
4         AllocateBackups(ListVNF[$m_{min}$,$n_{min}$,$k$])
5     end while
6 end for
```

Specifically, the function $MinOpCost$ operates as follows (Algorithm 3.2). We identify first the set of neighbors of physical node n (i.e., $Neighbours(n, k, d_{max})$) as the collection of physical nodes that can be accessed from n within d_{max} hops and hosts at least one VNF type k. We then compute the operational cost for the list of VNFs type k hosted in m than can be backed up in n (if n has enough resources) (Line 4). Next, we select the node n_{min} and m_{min} such that the operational costs of backups instantiated in n for primary VNFs type k hosted in m is the lowest (Line 7). The backup for VNFs type k contained in the list of VNFs (i.e., $ListVNF[m, n, k]$) is then allocated and update all variables (e.g.,a_n, $ListBackedUpVNF(k)$) (Line 4).

Having no neighbors for all physical nodes indicates that there are either insufficient resources to host backup VNFs or that there are no VNFs of type k that are left without backups.

Finally, the same procedure is performed for each VNF type (Line 1).

Algorithm 3.2 Function $MinOpCost(k, d_{max})$

```
1 function $MinOpCost(k, d_{max})$
2 for $n \in N$ do
3     for $m \in Neighbours(n, k, d_{max})$ do
4         $Cost[m, n, k], ListVNF[m, n, k] = ComputeOpCost(m, n, k)$
5     end for
6 end for
7 $n_{min}, m_{min} = \min_{n \in N, m \in N}(Cost[m, n, k])$
8 Return $n_{min}, m_{min}, ListVNF[m_{min}, n_{min}, k]$
9 end function
```

3.6.2 Solution 2: Algorithm ASB

Unlike the 1-1B Algorithm where the number of backups is equal to the number of primary VNFs, this algorithm, called *the Adjusting & Sharing Backups Algorithm* (ASB), finds the optimal number of backups based on the current chain demand, determines their locations and takes also into account the costs of migrating VNF Backups.

Algorithm 3.3 describes a high level description of the ASB Algorithm that encompasses two phases. The first phase of the Algorithm aims at provisioning, with minimal operational costs, VNF backups for the new additional demand that has been added at time t to the demand of the service chains at time $(t-1)$ (Line 1-14). It starts by considering the VNF types one by one (Line 2) and then for each physical node (Line 3), it checks whether the demand α_n^k (packets per second) of the VNFs type k has changed. If the demand has not changed, no new backups should be provisioned (Line 4). If the demand has decreased, the VNF backups serving the decreased demand is freed (Line 7). Finally, if the demand has increased, new backups should be provisioned for the additional backup (Line 10). The second phase of the algorithm (Line 15-20) aims at further optimizing the location of the VNF backups by leveraging VNF migration. In the following, we present the main functions that are used in this Algorithm:

• Function $ProvisionBackups(k, n, d_{max}, \beta)$ (Algorithm 3.4): this function provisions enough backups for VNF type k hosted in n to handle β packets per second. The locations considered to host these backups are within a distance of d_{max} hops to node n. This distance limitation is used to limit the search space and running time; however, it could impact the solution quality. Hence, a trade-off should be realized and the value of d_{max} could be determined by the network operator.

• Function $ProvisionBackupsInNode(m, n, \beta_n^k)$: this function provisions backups in physical node m to handle β_n^k packets per second required by VNFs type k hosted in n. If the capacity of node m is not sufficient, backup instances are provisioned using all the remaining capacity and β_n^k is updated to the amount of processing left without backups.

• Function $OpCosts(m, n, \beta_n^k)$: this function provides the operational costs (including running, management and synchronization costs) of backups type k that should be provisioned in node m to back up VNFs type k hosted in n in order to handle processing of β_n^k packets per second.

• Function $FreeBackups(k, n, \alpha_n^k)$: this function tears down VNF backups type k that are (i) provisioned for primary VNFs type k hosted in node n, (ii) having a processing capacity of α_n^k packets per second, (iii) having the highest operational costs among backup VNFs type k provisioned to back up VNFs type k hosted in n.

Algorithm 3.3 Adjusting & Sharing Backups Algorithm - ASB

```
Input: $N$: set of physical nodes
       $\overline{\alpha}_n^k, \alpha_n^k$: demand (packets per second) of VNFs type $k$ hosted in physical node $n$
       at time $(t-1)$ and $t$, respectively

Output: Provisioning minimum number of backups

Provisioning Phase:
for each VNF type $k = \{1, 2, 3, ...K\}$ do
    for $n \in N$ do
        if $\alpha_n^k = \overline{\alpha}_n^k$ then
            Backup adjustment not required
        end if
        else if $\alpha_n^k < \overline{\alpha}_n^k$ then
            $FreeBackups(k, n, \alpha_n^k)$
        end if
        else if $\alpha_n^k > \overline{\alpha}_n^k$ then
            $ProvisionBackups(k, n, d_{max}, \alpha_n^k - \overline{\alpha}_n^k)$
        end if
    end for
end for
Optimization Phase:
for each VNF type $k = \{1, 2, 3, ...K\}$ do
    for $m \in N$ do
        $Optimization(k, m, d_{max})$
    end for
end for
```

• $Optimization(k, m, d_{max})$ (Algorithm 3.5): this function optimizes the placement of VNF backups type k hosted in physical node m. Indeed, for each VNF backup type k hosted in m (Line 5), it carefully explores free resources available at the distance of d_{max} hops from its associated primary VNFs (Line 6) and then migrates the considered VNF backup if operational costs are less and minimized in the new location (migration costs are considered within these costs) (Line 6-13).

Algorithm 3.4 Function Provision Backups

```
function $ProvisionBackups(k, n, d_{max}, \beta_n^k)$
$a_m$: available resources in physical node $m$
Repeat
    Find node $m \in Neighbours(n, d_{max})$
        such that $a_m \neq 0$ and $OpCosts(m, n, \beta_n^k)$ is minimal
    $ProvisionBackupsInNode(m, n, \beta_n^k)$
Until backups to handle $\beta_n^k$ are provisioned or no free resources are left
end function
```

Algorithm 3.5 Function Optimization through VNF migration

```
function $Optimization(k, m, d_{max})$
$Backups(k, m, n)$: set of VNF backups type $k$ hosted in $m$
                    and serving VNFs type $k$ in physical node $n$
for $n \in N$ do
    for each VNF backup $v \in Backups(k, m, n)$ do
        Find $o \in Neighbours(n, d_{max})$
            such that operational costs in $o$ is less than in $m$
        if $o$ exists then
            Migrate VNF $v$ from physical node $m$ to node $o$
        end if
        else
            No migration is requested
        end if
    end for
end for
end function
```

3.7 Simulation and Results

In this section, we report the results obtained through extensive simulations performed to assess the performance of the proposed algorithms (1-1B and ASB). We mainly evaluate their performance and compare it to the optimal one supplied by the ILP (implemented using CPLEX). We considered several metrics including backup running costs, management costs, synchronization costs, operational costs, the total number of backups, and the backup ratio (defined as the number of backup instances divided by the total number of instances). Furthermore, we predict the future data rate of all the SFC demands by using the ARIMA model and the obtained prediction are used as inputs to the proposed backup provisioning solutions (ILP, 1-1B and ASB).

3.7.1 Simulation Setup

In order to assess the proposed solutions, we have implemented a C/C++ simulator that mimics the whole environment, including the physical infrastructure, embedded service chains, and the two proposed algorithms. In the following, we provide more details about the simulated physical infrastructure, the different costs, and the considered configuration scenarios.

• **Physical infrastructure and service function chains:** we randomly generated a physical infrastructure made out from 24 physical nodes with 52 physically links connecting and ensure the connectivity (not through different multi-hop paths) among any pair of nodes. Computing capacities of physical nodes were generated randomly between 40 and 120 virtual machines. All virtual machines assumed to host the VNFs are assumed to have the same resource capacity (t2.micro instance Ghrada *et al.* (2018)). A single virtual machine could host only one VNF. The generated links have distinct bandwidth capacities that were randomly generated between 50MB and 1000MB and have propagation delays generated between 10ms and 20ms.

We assume that embedding of service chains has been already performed using an existing an SFC placement and chaining algorithm. In particular, we used the SFC provisioning scheme proposed by Racheg *et al.* (2017) to allocate resources for the service chains while satisfying their resource and end-to-end delay requirements.

• **Operational costs:** as electricity prices vary from a region to another Jiang (2011), the running and management costs for each backup instance were randomly varied, ranging from \$0.0116 per hour to \$0.0186 per hour. Similarly, running costs of a backup instance range from \$0.0577 per hour to \$0.4687 per hour as suggested by the VM pricing of Amazon EC2 AWS

(2022). The cost of synchronizing bandwidth between the primary VNF and its backup instance is set to \$0.01 per GB per hour.

• **Configuration scenarios:** in order to evaluate the proposed algorithms, we considered six different scenarios (S0–S5), where, in each, the number of SFCs as well as their mapping configuration are different from the others.

Table 3.2 summarizes the configuration of each of the considered scenarios. For instance, for Scenario S0, there are 4 SFCs. The number of VNFs per SFC is 3. We consider that there are three different VNF types. The packet processing capacity of the standard virtual machine considered to host the VNFs varies depending on the type of the instance Ghrada *et al.* (2018). Hence, the processing capacity used in scenario S0 is 11000 pps for Type 1 VNFs, 18000 pps for Type 2, and 9000 pps for Type 3. The synchronization bandwidth required to perform the synchronization between each VNF and its backup instance are as following 2MB for Type 1 VNFs, 4MB for Type 2 VNFs, and 6MB for Type 3 VNFs.

Table 3.2 Configurations of the different considered scenarios

Scenarios	**S0**	**S1**	**S2**	**S3**	**S4**	**S5**
Number of SFCs	4	20	40	60	80	100
Number of VNFs per SFC	3	Random number between 1 and 6				
Number of VNF Types	3	6				
Processing Capacity	Type 1: 11000pps Type 2: 18000pps Type 3: 9000pps	Type 1: 5000pps Type 2: 7000pps Type 3: 9000pps Type 4: 11000pps Type 5: 13000pps Type 6 18000pps				
Synchronization Bandwidth	Type 1: 2Mbps Type 2: 4Mbps Type 3: 6Mbps	Type 1: 3Mbps Type 2: 5Mbps Type 3: 2Mbps Type 4: 6Mbps Type 5: 7Mbps Type 6: 8Mbps				

Regarding the demand of the SFCs, we used packet traffic rates considered and analyzed in Zhani *et al.* (2009) (see Figure 3.4). The packet rate is between 10000 and 25000 packets per second (pps) and is measured each timeslot of 15 $minutes$.

We run simulations for each scenario for 24 $hours$. As it can be noticed, the Scenario S0 is a small-scale deployment with a small number of SFC (and hence VNFs). As a result, it is possible to obtain the optimal solution using CPLEX and compare it to the ones obtained 1-1B and ASB. However, Scenarios S1 to S5 are large-scale deployments, and, thus, the CPLEX Solver is unable to determine the optimal solution because of the high number of SFCs and VNFs that are embedded into the infrastructure. For these scenarios, we only report the results comparing the performance of ASB Algorithm compared to that of the 1-1B Algorithm.

3.7.2 Simulation Results

To further assess how well the two proposed algorithms perform in the various considered scenarios, we computed the following metrics:

– *Running costs*: they correspond to the costs of running the backup instances in physical nodes. As backups are spread over different locations, their running cost is different as it depends on its location.

– *Management costs*: these are the costs incurred by deployment and migration of a virtual machine image from one location inside the infrastructure to another.

– *Synchronization costs*: these are the costs of data exchange between primary VNFs and their associated backup instances. The synchronization is performed to ensure that backup instances keep the last state of the VNF instances. These costs depend on the used links to exchange data as well as the synchronization rate, which depends on the type of the VNF.

– *Operational costs*: these are the overall costs which are the sum of the running costs, management costs and synchronization costs.

– *Number of backups*: it corresponds to the total number of backups that have been provisioned in the infrastructure.

– *Backup ratio*: it is the ratio of the total number of backup instances to the total number of VNFs. The lower is this ratio, the better the backup provisioning scheme is. In other words, if this ratio is low, it indicates that the system is using a number of backup VNFs that is much lower than the primary VNFs.

In the following, we present first aim at comparing the performance of the 1-1B Algorithm and the ASB Algorithm compared to the ILP. Hence, in the first sets of experiments, we do not use prediction. We first report simulation results for the small-scale deployment Scenario (S0) using the ILP, the 1-1B Algorithm and the ASB Algorithm. Afterwards, we report results for large-scale deployments (Scenario S1 to S5) but only for the 1-1B Algorithm and the ASB Algorithm. Finally, we show the benefits of demand prediction by conducting several simulations on the ASB Algorithm but using the predicted demand rather than the current one.

• **Small-Scale deployment (Scenario S0) - ILP vs. 1-1B vs. ASB**:

We first report the detailed results obtained for Scenario S0 showing all measured metrics for the three proposed solutions including the ILP (using the CPLEX Solver), the 1-1B Algorithm and the ASB algorithm. As previously mentioned, this scenario is a small-scale one, and hence, CPLEX is able to compute the optimal solution.

Figure 3.4 shows the demand over time of each service function chain in scenario S0. As there are many fluctuations, it is natural that the number of backups needed to handle the demand in case of failure should be dynamically adjusted over time.

Figure 3.5 shows a comparison between the number of backups over time obtained using the proposed algorithms and the optimal result produced by CPLEX. We can see that ASB gives a slightly higher number of backups compared to CPLEX's optimal solution, indicating that its solution is not far from optimal. Furthermore, we note that ASB provides much less backups than 1-1B. Furthermore, Figure 3.6 shows a comparison between the backup ratio obtained using the three solutions. It shows a significant difference between ASB and 1-1B. The backup ratio for CPLEX is slightly lower than ASB, showing an average ratio of 40% and 60% for both algorithms, respectively, compared to 100% for the 1-1B Algorithm.

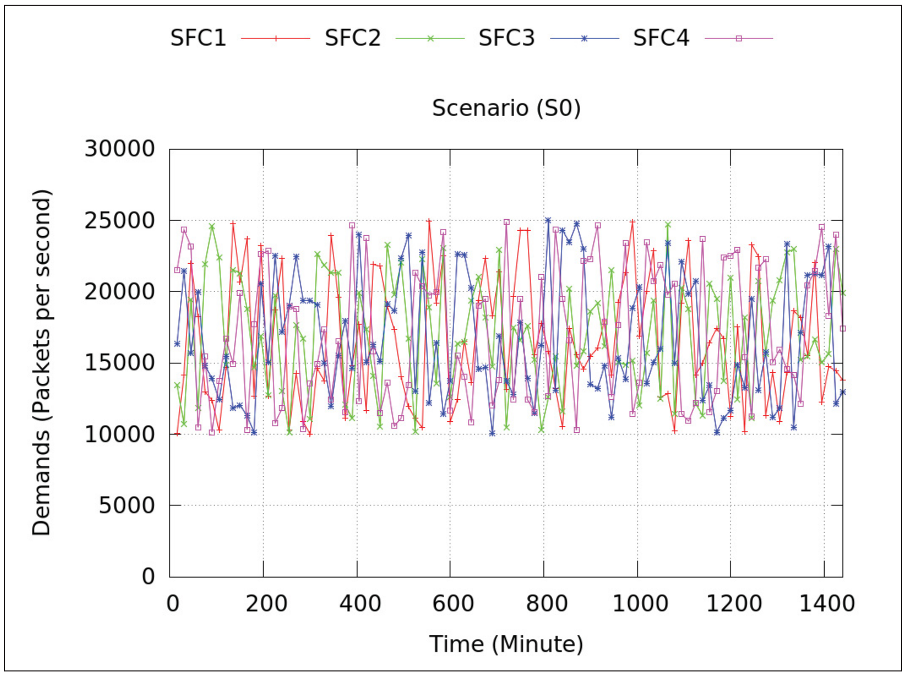


Figure 3.4 Service chain demand over time - Scenario S0.

To explain these results, 1-1B and ASB provide service chain survivability against single or multiple physical node failures by guaranteeing that enough backups are available before the failure occurs. However, 1-1B provisions backups based on the number of original VNFs. Therefore, the cost of management will be high compared to the ASB as the total number of provisioned backups will be high as shown in Figure 3.5. Unlike 1-1B, ASB looks into the demand to provision backup instances rather than the original VNFs. Therefore, ASB evaluates the demand and reduce the number of backups based on the demand (provision less backups). To do so, ASB adjusts the number of backups as necessary by creating new backups, sharing existing backups, and migrating existed backups if needed. In this case, the cost of management will be low compared to the 1-1B as the total number of provisioned backups will be reduce in the infrastructure as shown in Figure 3.5.

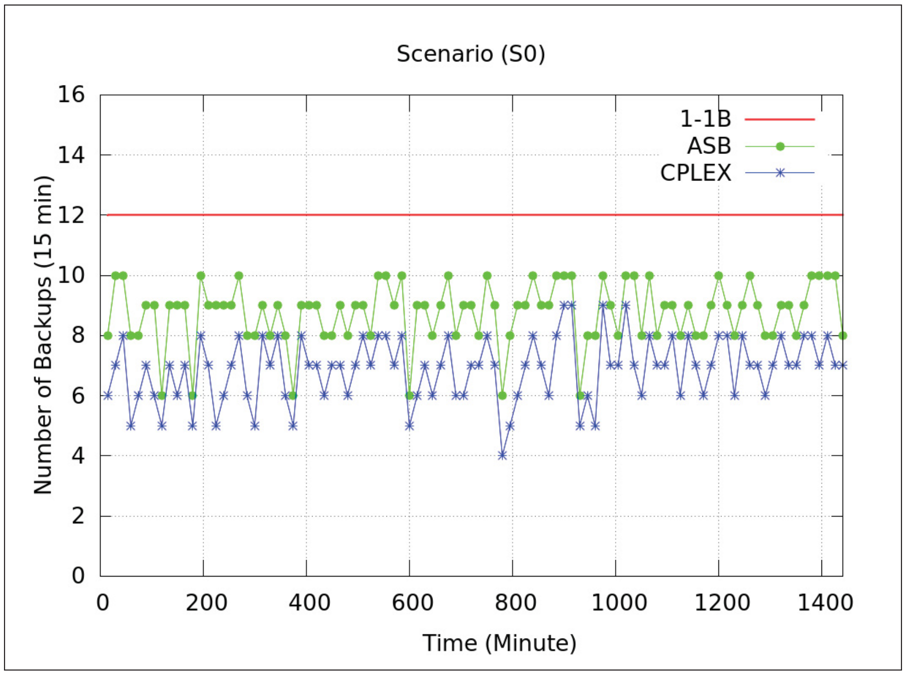


Figure 3.5 Number of backups over time - Scenario S0.

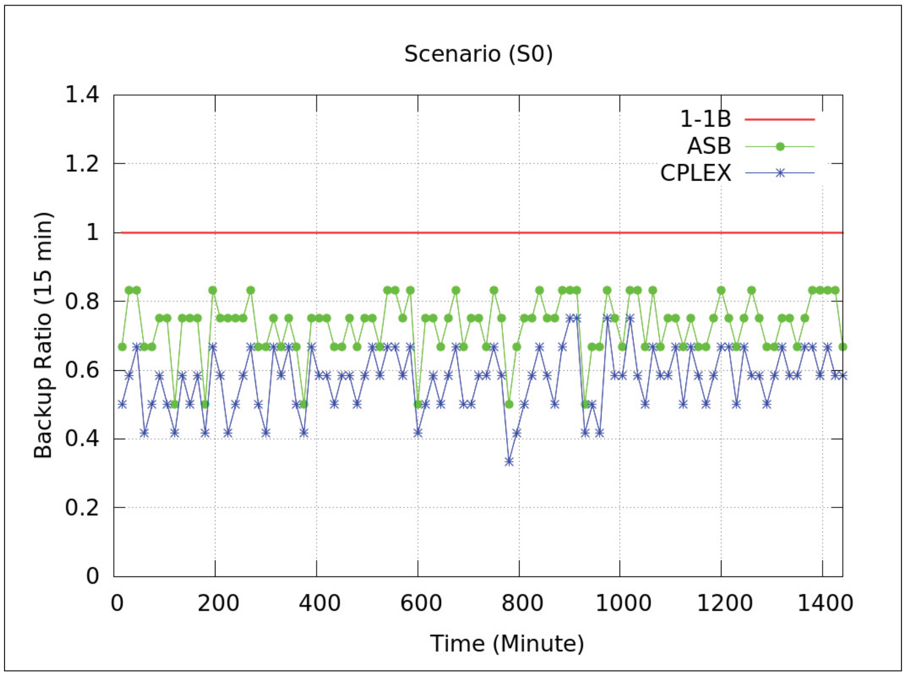


Figure 3.6 Backup ratio over time - Scenario S0.

Furthermore, Figure 3.7 shows the different types of costs throughout the simulation for scenario S0. It shows that the ASB Algorithm succeeds to reduce by up to 25% operational costs compared to 1-1B Algorithms as it provisions much less VNF backups as it can be deduced from the difference in running costs of the backup instances. It also incurs less backup instantiation and migration over time as it results in less managements costs. The synchronization costs are also much lower, demonstrating a better choice of the locations of the VNF backups.

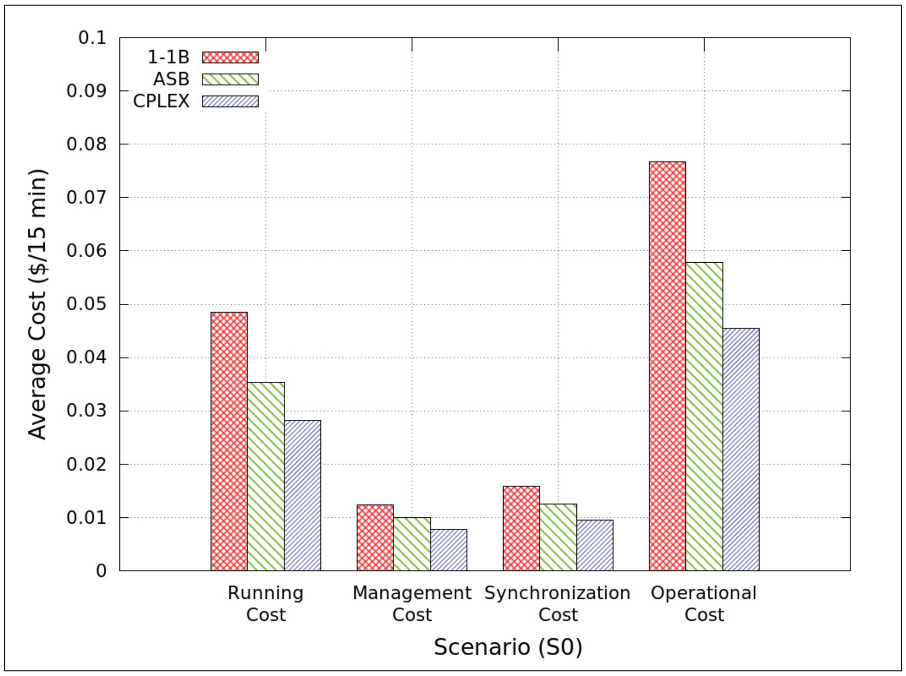


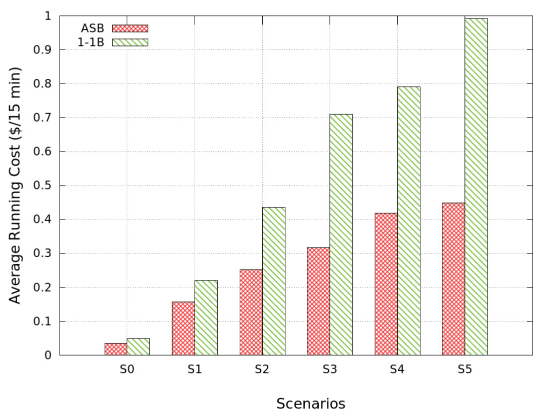
Figure 3.7 Cost breakdown - Scenario S0.

We can also see in the figure that the ASB Algorithm leads to total operational costs that are closer to the one of the optimal solution supplied by CPLEX. This indicates that the ASB approach succeeds to parse the solution space and provide a solution close to the optimal.

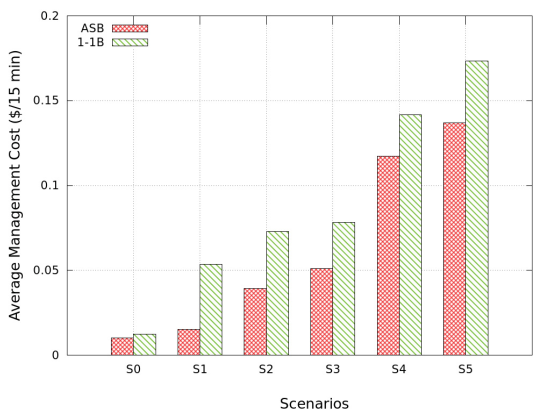
• **Large-Scale deployment (Scenario S0 to S5) - ASB vs. 1-1B**:

We now report the results obtained for all scenarios but only for the 1-1B Algorithm and the ASB algorithm as these scenarios are large-scale involving a large number of SFCs and VNFs, and hence, it is not possible to compute the optimal solution with the CPLEX Solver.

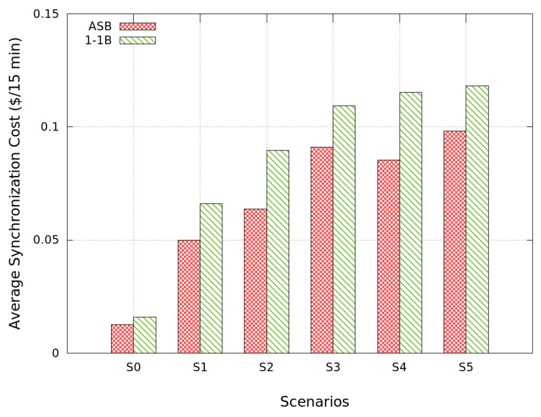
Figure 3.8 compares the average costs (all types of costs) found with the ASB algorithm with the 1-1B algorithm for each of the six scenarios. We can see that for all scenarios, ASB provides lower costs compared to 1-1B for all types of costs.



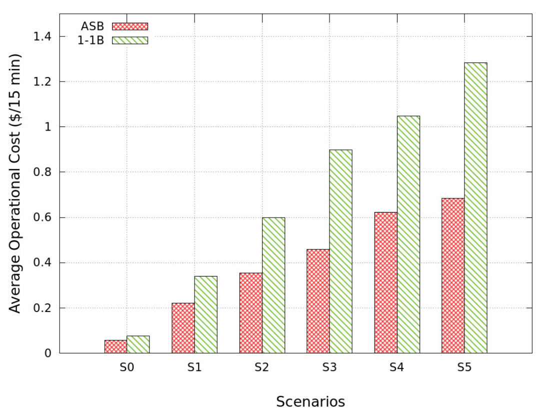
a) Average running costs.



b) Average management costs.



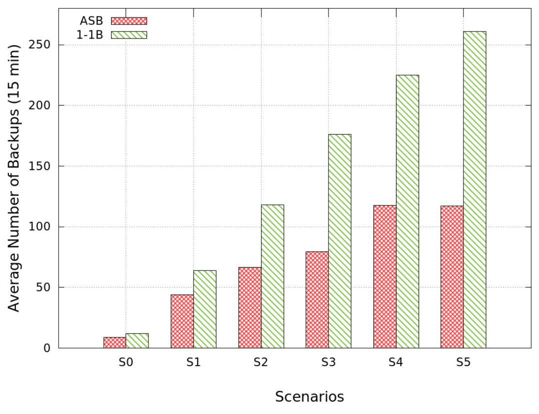
c) Average synchronization costs.



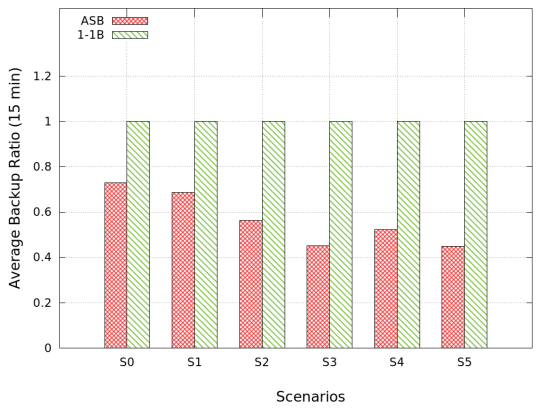
d) Average operational costs.

Figure 3.8   Average costs per time interval (Scenarios S0 to S5).

Figure 3.8a depicts the average running costs incurred by the two algorithms for all scenarios. The costs of the ASB algorithm are lower than the costs incurred by 1-1B in all different scenarios. We can see that the difference in running costs between the two algorithms grows as the number of SFCs and VNFs expands and can reach up to 50% lower (see scenario S5). This can be explained by the fact that, when there is a large number of VNFs, there are more possibilities for many VNFs to share the same backups.



a) Average number of backups.



b) Average backup ratio.

Figure 3.9 Average number of backups and backup ratio (Scenarios (S0-S5)).

As shown in Figure 3.8b, we can notice that the management costs (instantiation and migration) with ASB are much lower than those of 1-1B. This means that the number of backup instances with ASB is less than the one found with 1-1B. We also notice that the costs incurred by ASB grow much slower than 1-1B. Finally, Figure 3.8c shows that synchronization costs obtained with ASB is lower than 1-1B as the ASB allows to place VNF backups closer to primary VNFs. Unlike ASB, 1-1B uses a high number of backup instances and places them far from their associated primary VNFs. As a result, the incurred operational costs (i.e., the sum of all the previously mentioned costs) incurred by the ASB Algorithms is up to 40% lower compared to the 1-1B Algorithm (Figure 3.8d).

Figure 3.9a shows a comparison between the average number of backups obtained when using ASB and 1-1B. We can see that the number of backups grows as the number of SFCs and VNFs grows. However, compared to the ASB, 1-1B offers more backups for the small-scale scenario (S0). For larger-scale scenarios (S1-S5), the ASB Algorithm more than 50% less backups than the 1-1B Algorithm.

Figure 3.9b shows a comparison between the average backup ratio obtained using ASB and 1-1B. The Algorithm 1-1B has a backup ratio of 100% as it provision one backup per VNF. However, for the ASB Algorithm, in the small-scale scenario S0, the backup ratio for ASB is around 70% and it could reach 45% for larger-scale scenarios where there are higher opportunities to share VNF backups and minimize their number.

• **ASB Algorithm with Demand Prediction**:

In this set of simulations, we evaluate the benefits of demand prediction in minimizing the management costs by avoiding useless backup creation and migration.

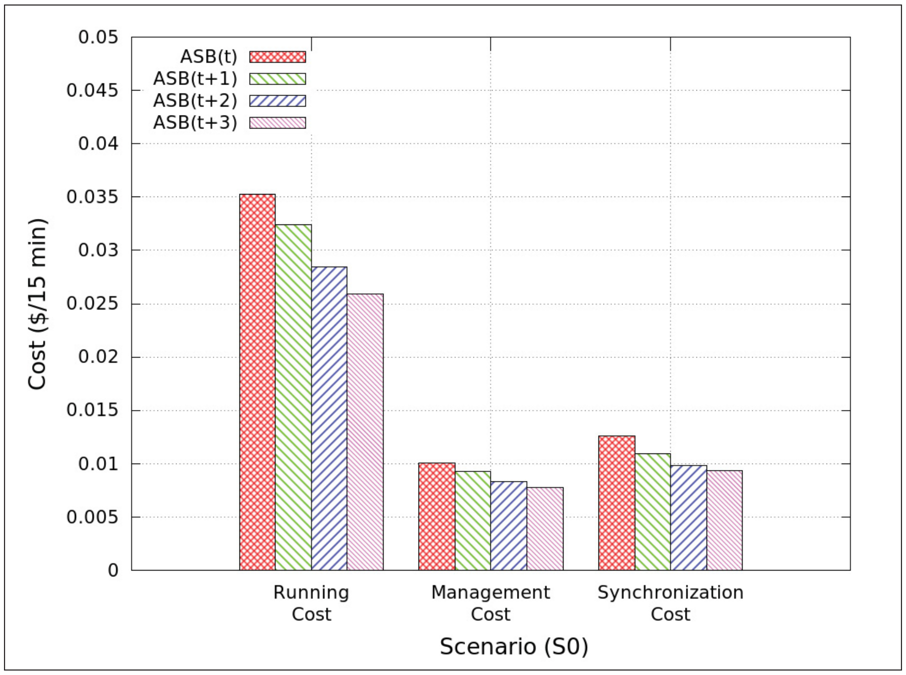


Figure 3.10 Cost breakdown when prediction is used - Scenario S0.

To this end, we start by predicting the future demand rate for each SFC using the ARIMA(p = 10, q = 1, d = 10) model where p is the number of lag observations used as input to the model, d is the number of times data is differentiated, and q is the size of the moving average window Zhani *et al.* (2009). We used the ARIMA model to predict the future demand at time $t + 1$, $t + 2$, $t + 3$ (assuming current timeslot is t) thanks to the multi-step prediction procedure described in Zhani *et al.* (2009).

Afterwards, we run the ASB algorithm for all the studied scenarios. We denote by ASB(t), the ASB algorithm using as input the demand at the current timeslot t (i.e., no prediction), and by ASB($t+i$), the ASB algorithm using as input the average predicted demand at timeslots $t+1$, $t+2$, ..., $t+i$.

Figure 3.10 reports the detailed breakdown of the operational costs, i.e., running, management, and synchronization costs found using ASB(t), ASB($t+1$), ASB($t+2$), ASB($t+3$) for scenario S0. We can see in the figure that when the backup provisioning scheme uses prediction, all costs are reduced. This is because, thanks to prediction, ASB takes into consideration the predicted fluctuation of the traffic rate in each time slot. For instance, to take a decision at time slot t, ASB considers the prediction at $t+i$.

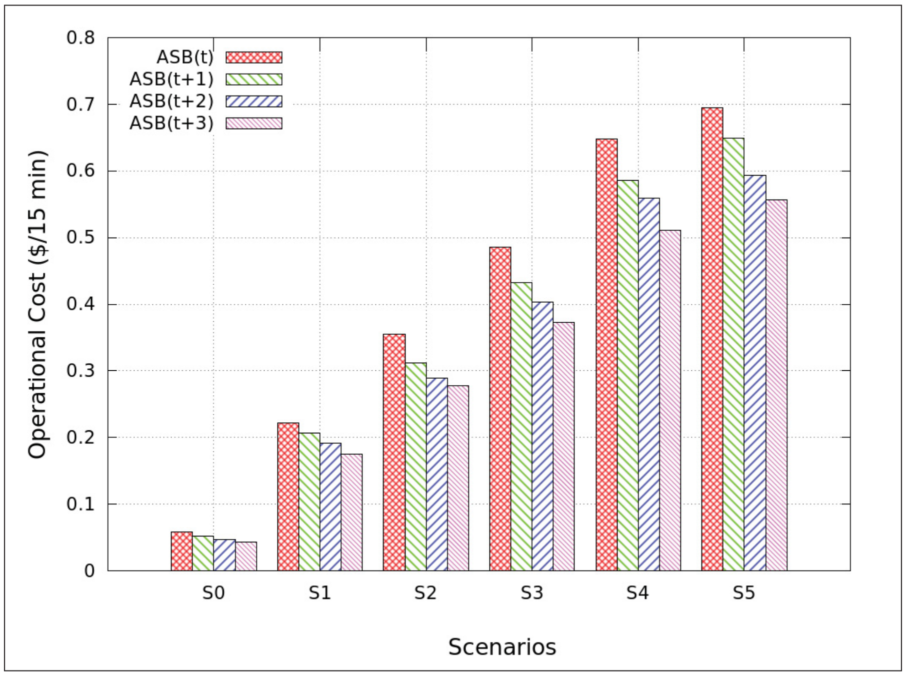


Figure 3.11 Operational costs when prediction is used - Scenarios S0 to S5.

Hence, if the predicted traffic rate at $t+i$ is less than the one at timeslot t, there is no need to create backups at timeslot t even if the traffic rate at timeslot t is high. As a result, the running and synchronization costs are significantly reduced. On the other hand, if the traffic rate at $t+1$ is larger than the rate at timeslot t, this indicates that the $t+1$ requires more backups. As a result, we must adjust the number of backups as necessary by creating or migrating in advance backups. We also notice that the more prediction horizon increases, the all costs are reduced.

Figure 3.11 reports the operational costs found using ASB(t), ASB($t+1$), ASB($t+2$), ASB($t+3$) for all scenarios. We can see in the figure that when the backup provisioning scheme uses prediction, the operational costs are reduced as there less backup instantiations and migrations over time. It also shows that the more visibility we have in the future, the less operational costs are incurred. This is because, thanks to prediction, the creation or the migration of many backups are avoided, and thereby the operation costs are reduced.

3.8 Conclusion

In this paper, we tackled one of the key challenges confronting infrastructure providers to deploy and manage service function chains. In particular, we looked at the survivability of service chains aiming at ensure service continuity in face of single or multiple simultaneous node failures. We hence proposed a new survivability management module able to dynamically adjust the number of shared backups based on the predicted traffic rates. The module is in charge of creating backup instances, migrating them if needed in order to share backup instances while minimizing backup operational costs (including the costs of running and managing the instances and the costs of synchronization among primary and backup instances).

The problem was formulated as an Integer Linear Program that was implemented in the CPLEX Solver, and two heuristic algorithms (ASB Algorithm and 1-1B Algorithm) were proposed to deal with large-scale scenarios of the problem. We show through extensive simulations how effectively the ASB algorithm compared to 1-1B and how it is able to find near-optimal solutions close to the one provided by the ILP solved using CPLEX.

As a future work, we plan to better optimize the two algorithms in order to further reduce their complexity and better parse the solution space to further approach the optimal solution. Another interesting avenue for this work is to design solutions addressing survivability but taking into account the availability of physical nodes and failure probabilities computed based on the history of the failures occurred in the infrastructure. In this case, it would be interesting to avoid backup deployment for instances hosted in physical location with low failure probability, which may significantly reduce backup operational costs.

CONCLUSION AND RECOMMENDATIONS

Emerging technologies like Network Function Virtualization is currently changing network management and operation. Indeed, network services are provided in the form of service function chains composed of an order series of virtual network functions. These service function chains process incoming traffic and route it towards its destination. Despite all benefits brought by this new concept, the deployment and management of service function chains is still a challenging tasks as it bring several challenges including the allocation of the requested resources, the implementation of distributed virtual network functions and the management of failures and survivability assurance.

In this thesis, we addressed two challenges pertaining to the deployment of SFCs. Particularly, we aimed at 1) designing solutions to perform synchronization among multiple instances of the same network function while minimizing synchronization costs between them and ensuring that the synchronization delay does not exceed a certain bound to ensure normal operation of the function and, 2) designing solutions able to ensure the survivability of service chains embedded into the physical infrastructure to guarantee their resilience even in case of multiple and simultaneous failures.

We hence proposed a technique that identifies the optimal communication pattern to perform synchronization between the VNF instances and able to that minimizes the synchronization cost and ensures a bounded synchronization delay. We also introduced the use a new function called Synchronization Function that ensures consistency among a set of instances and allows to further reduce the synchronization cost. The problem was formulated as an integer linear program implemented in CPLEX and three heuristic algorithms (SPT, SSF, and MSF) were proposed to deal with large-scale scenarios. Last, we proposed another algorithm that benefit from the migration of network function instances to further reduce costs. Extensive simulations show that the proposed algorithms efficiently find near-optimal solutions with minimal computation time and provide better results compared to existing solutions.

Furthermore, we proposed different solutions to ensure service chain survivability by provisioning backup with minimal operational costs. We then formulated the backup provisioning problem as an integer linear program and two heuristic algorithms (ASB and 1-1B) were proposed to solve the problem in large-scale scenarios. Finally, we demonstrated through a set of experiments with various scenarios that our algorithms offer solutions that are close to the optimal solution supplied by CPLEX. We also leveraged a traffic prediction model to forecast demand traffic rates in order to further improve the proposed backup provisioning solutions and allow them to avoid unnecessary backup provisioning, and thereby further reduce operational costs.

Future Research Directions

In order to stir the adoption of the aforementioned solutions, there are many considerable and interesting objectives that we plan to track in the future. For instance, we plan to carefully study the synchronization costs of different types of VNFs as well as the synchronization requirements in terms of delay as these parameters could impact the performance of the function itself (e.g., late attack detection in the case of an IDS). We also plan to develop and further extend the idea of Synchronization Function that could be integrated within the network to be customized based on the type of the function. Indeed, the nature of the network function could significantly impact the design the synchronization function. Furthermore, we plan to further optimize the proposed heuristic algorithms in order to reduce their complexity and improve their performance to parse the solution space and better approach optimal solutions.

BIBLIOGRAPHY

Aben, E. [https://labs.ripe.net/author/emileaben/internet-traffic-during-the-world-cup-2014/. Accessed 28 Aug. 2022]. (2014). Internet Traffic During the World Cup 2014.

Aidi, S., Zhani, M. F. & Elkhatib, Y. (2018). On improving service chains survivability through efficient backup provisioning. *International Conference on Network and Service Management (CNSM)*, pp. 108–115.

Al-Roomi, M., Al-Ebrahim, S., Buqrais, S. & Ahmad, I. (2013). Cloud computing pricing models: a survey. *International Journal of Grid and Distributed Computing*, 6(5), 93–106.

Alomari, Z., Zhani, M. F., Aloqaily, M. & Bouachir, O. (2020). On minimizing synchronization cost in NFV-based environments. *IEEE/ACM International Conference on Network and Service Management (CNSM)*, pp. 1–9.

Aloqaily, M., Kantarci, B. & Mouftah, H. T. (2014). Provisioning delay effect of partaking a trusted third party in a vehicular cloud. *IEEE Global Information Infrastructure and Networking Symposium (GIIS)*, pp. 1–3.

Andersen, D. G. (2002). Theoretical approaches to node assignment. https://kilthub.cmu.edu/ndownloader/files/12103001. Accessed 10 Jan. 2022.

Aqil, A., Khalil, K., Atya, A. O., Papalexakis, E. E., Krishnamurthy, S. V., Jaeger, T., Ramakrishnan, K., Yu, P. & Swami, A. (2017). Jaal: Towards network intrusion detection at isp scale. *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pp. 134–146.

AWS. (2022). Amazon Web Services Inc., Amazon EC2 Instance Types. https://aws.amazon.com/ec2/instance-types/?nc1=h_ls. Accessed 15 Feb. 2022.

Ayoubi, S., Assi, C., Narayanan, L. & Shaban, K. (2016a). Optimal polynomial time algorithm for restoring multicast cloud services. *IEEE Communications Letters*, 20(8), 1543–1546.

Ayoubi, S., Chen, Y. & Assi, C. (2016b). Towards promoting backup-sharing in survivable virtual network design. *IEEE/ACM Transactions on Networking*, 24(5), 3218–3231.

Barford, P., Donoho, D., Flesia, A. & Yegneswaran, V. (2002). Characteristics of network delays in wide area file transfers. https://minds.wisconsin.edu/bitstream/handle/1793/60288/TR1443.pdf?sequence=1. Accessed 15 Apr. 2022.

Bari, M. F., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., Zhang, Q. & Zhani, M. F. (2012). Data center network virtualization: A survey. *IEEE communications surveys & tutorials*, 15(2), 909–928.

Bays, L. R., Oliveira, R. R., Barcellos, M. P., Gaspary, L. P. & Madeira, E. R. M. (2015). Virtual network security: threats, countermeasures, and challenges. *Journal of Internet Services and Applications*, 6(1), 1–19.

Beck, M. T. & Botero, J. F. (2015). Coordinated allocation of service function chains. *2015 IEEE global communications conference (GLOBECOM)*, pp. 1–6.

Beck, M. T., Botero, J. F. & Samelin, K. (2016). Resilient allocation of service function chains. *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 128–133.

Bijwe, S., Machida, F., Ishida, S. & Koizumi, S. (2017). End-to-end reliability assurance of service chain embedding for network function virtualization. *IEEE conference on network function virtualization and software defined networks (NFV-SDN)*, pp. 1–4.

Binh, H. T. T., Thanh, P. D. & Thang, T. B. (2019). New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm. *Knowledge-Based Systems*, 180, 12–25.

Bo, L., Huang, T., SUN, X.-c., Chen, J.-y. & Liu, Y.-j. (2014). Dynamic recovery for survivable virtual network embedding. *Journal of China Universities of Posts and Telecommunications*, 21(3), 77–84.

Bökler, F. (2017). The Multiobjective Shortest Path Problem is NP-hard, or is it? *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 77–87.

Boutaba, R., Zhang, Q. & Zhani, M. F. (2014). Virtual machine migration in cloud computing environments: Benefits, challenges, and approaches. In *Communication Infrastructures for Cloud Computing* (pp. 383–408). IGI Global.

Box, G. E., Jenkins, G. M., Reinsel, G. C. & Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Carapinha, J. & Jiménez, J. (2009). Network virtualization: a view from the bottom. *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 73–80.

Casazza, M., Bouet, M. & Secci, S. (2019). Availability-driven NFV orchestration. *Computer Networks*, 155, 47–61.

Cho, D., Taheri, J., Zomaya, A. Y. & Bouvry, P. (2017). Real-time virtual network function (VNF) migration toward low network latency in cloud environments. *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pp. 798–801.

Chowdhury, M., Rahman, M. R. & Boutaba, R. (2011). Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on networking*, 20(1), 206–219.

Chowdhury, S. R., Ahmed, R., Khan, M. M. A., Shahriar, N., Boutaba, R., Mitra, J. & Zeng, F. (2016). Dedicated protection for survivable virtual network embedding. *IEEE Transactions on Network and Service Management*, 13(4), 913–926.

Chua, F. C., Ward, J., Zhang, Y., Sharma, P. & Huberman, B. A. (2016). Stringer: Balancing latency and resource usage in service function chain provisioning. *IEEE Internet Computing*, 20(6), 22–31.

Clemm, A., Zhani, M. F. & Boutaba, R. (2020). Network management 2030: Operations and control of network 2030 services. *Journal of Network and Systems Management*, 1–30.

Cohen, G. Downtime, Outages and Failures-Understanding Their True Costs. https://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html. Accessed 10 May 2022.

Cosma, O., Pop, P. C. & Zelina, I. (2020). A novel genetic algorithm for solving the clustered shortest-path tree problem. *Carpathian Journal of Mathematics*, 36(3), 401–414.

Cosma, O., Pop, P. C. & Zelina, I. (2021). An effective genetic algorithm for solving the clustered shortest-path tree problem. *IEEE Access*, 9, 15570–15591.

Cotroneo, D., De Simone, L., Iannillo, A. K., Lanzaro, A., Natella, R., Fan, J. & Ping, W. (2014). Network function virtualization: Challenges and directions for reliability assurance. *IEEE international symposium on software reliability engineering workshops*, pp. 37–42.

COUTINHO, N., SCHAAPMAN, P. & TEMIN, T. [https://edtechmagazine.com/higher/sites/default/files/rg_virtualization_011811.pdf. Accessed 11 July. 2021]. (2014). Virtualization and Infrastructure Optimization reference guide.

Ding, W., Yu, H. & Luo, S. (2017). Enhancing the reliability of services in NFV with the cost-efficient redundancy scheme. *IEEE international conference on communications (ICC)*, pp. 1–6.

Dinh, N.-T. & Kim, Y. (2019). An efficient reliability guaranteed deployment scheme for service function chains. *IEEE Access*, 7, 46491–46505.

Endo, S., Miyamoto, T., Kumagai, S. & Fujii, T. (2004). A data synchronization method for peer-to-peer collaboration systems. *IEEE International Symposium on Communications and Information Technology (ISCIT)*, 1, 368–373.

ETSI. (2015). ETSI GS NFV-REL 001 V1.1.1 (2015-01), Network Functions Virtualisation (NFV) Resiliency Requirements. https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_nfv-rel001v010101p.pdf. Accessed 28 Jan. 2022.

ETSI. (2020). Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV. https://www.etsi.org/deliver/etsi_gr/NFV/001_099/003/01.05.01_60/gr_NFV003v010501p.pdf. Accessed 20 Feb. 2022.

ETSI, N. F. V. (2012). Introductory white paper. https://portal.etsi.org/nfv/nfv_white_paper.pdf. Accessed 23 Apr. 2019.

Faiz, M. & Shanker, U. (2016). Data synchronization in distributed client-server applications. *IEEE International Conference on Engineering and Technology (ICETECH)*, pp. 611–616.

Fan, J., Ye, Z., Guan, C., Gao, X., Ren, K. & Qiao, C. (2015). GREP: Guaranteeing reliability with enhanced protection in NFV. *ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pp. 13–18.

Fan, J., Jiang, M., Rottenstreich, O., Zhao, Y., Guan, T., Ramesh, R., Das, S. & Qiao, C. (2018). A framework for provisioning availability of NFV in data center networks. *IEEE Journal on Selected Areas in Communications*, 36(10), 2246–2259.

Gember, A., Krishnamurthy, A., John, S. S., Grandl, R., Gao, X., Anand, A., Benson, T., Akella, A. & Sekar, V. (2013). Stratos: A network-aware orchestration layer for middleboxes in the cloud. http://robotics.duke.edu/courses/compsci514/spring14/Papers/Stratos.pdf. Accessed 12 Sept. 2021.

Gember-Jacobson, A. & Akella, A. (2015). Improving the safety, scalability, and efficiency of network function state transfers. *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pp. 43–48.

Gember-Jacobson, A., Viswanathan, R., Prakash, C., Grandl, R., Khalid, J., Das, S. & Akella, A. (2014). OpenNF: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 44(4), 163–174.

Ghaleb, A. M., Khalifa, T., Ayoubi, S., Shaban, K. B. & Assi, C. (2016). Surviving multiple failures in multicast virtual networks with virtual machines migration. *IEEE Transactions on Network and Service Management*, 13(4), 899–912.

Ghrada, N., Zhani, M. F. & Elkhatib, Y. (2018). Price and performance of cloud-hosted virtual network functions: Analysis and future challenges. *4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pp. 482–487.

Gray, N., Lorenz, C., Müssig, A., Gebert, S., Zinner, T. & Tran-Gia, P. (2017). A priori state synchronization for fast failover of stateful firewall VNFs. *International Conference on Networked Systems (NetSys)*, pp. 1–6.

Halpern, J. & Pignataro, C. (2015). Service function chaining (SFC) architecture. https://www.rfc-editor.org/rfc/rfc7665.txt. Accessed 26 Apr. 2018.

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

Hassani, A., Haghighi, P. D., Burstein, F. & Davey, S. (2017). Context aware data synchronisation during emergencies. *International and Interdisciplinary Conference on Modeling and Using Context*, pp. 406–417.

Herrera, J. G. & Botero, J. F. (2016). Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3), 518–532.

Hmaity, A., Savi, M., Musumeci, F., Tornatore, M. & Pattavina, A. (2017). Protection strategies for virtual network functions placement and service chains provisioning. *Networks*, 70(4), 373–387.

Huang, M., Liang, W., Shen, X., Ma, Y. & Kan, H. (2019). Reliability-aware virtualized network function services provisioning in mobile edge computing. *IEEE Transactions on Mobile Computing*, 19(11), 2699–2713.

IBM. (2007). IBM Global Education White Paper. Virtualization in Education. https://docplayer.net/235160-Ibm-global-education-white-paper-virtualization-in-education.html. Accessed 19 July. 2020.

IBM. (2020). IBM ILOG CPLEX Optimizer. https://www.ibm.com/analytics/cplex-optimizer. Accessed 19 July. 2022.

ISG, N. (2014). Network Function Virtualisation NFV-Resiliency Requirements. *ETSI GS NFV-REL*, 1, v1.

Jiang, J. [https://www.npr.org/sections/money/2011/10/27/141766341/the-price-of-electricity-in-your-state. Accessed 5 Mar. 2022]. (2011). The Price Of Electricity In Your State.

Jurski, J. & Wozniak, J. (2009). Routing decisions independent of queuing delays in broadband LEO networks. *IEEE Global Telecommunications Conference(GLOBECOM)*, pp. 1–6.

Kablan, M., Alsudais, A., Keller, E. & Le, F. (2017). Stateless network functions: Breaking the tight coupling of state and processing. *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 97–112.

Khalid, J. & Akella, A. (2019). Correctness and performance for stateful chained network functions. *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pp. 501–516.

Khalid, J., Gember-Jacobson, A., Michael, R., Abhashkumar, A. & Akella, A. (2016). Paving the way for {NFV}: Simplifying middlebox modifications using statealyzr. *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pp. 239–253.

Kubler, S., Främling, K. & Derigent, W. (2015). P2P Data synchronization for product lifecycle management. *Computers in Industry*, 66, 82–98.

Landström, S., Bergström, J., Westerberg, E. & Hammarwall, D. (2016). NB-IoT: A sustainable technology for connecting billions of devices. *Ericsson Technology Review*, 4, 2–11.

Lee, J., Ko, H., Suh, D., Jang, S. & Pack, S. (2017). Overload and failure management in service function chaining. *IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5.

Li, J., Liang, W., Huang, M. & Jia, X. (2019). Providing reliability-aware virtualized network function services for mobile edge computing. *International Conference on Distributed Computing Systems (ICDCS)*, pp. 732–741.

Lowe, S. (2012). Hyper-V™ vs. vSphere™: Understanding the differences. *Solarwinds. Retrieved February*, 2, 2014.

Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P. & Gaspary, L. P. (2015). Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106.

Luong, N. C., Wang, P., Niyato, D., Wen, Y. & Han, Z. (2017). Resource management in cloud networking using economic analysis and pricing models: A survey. *IEEE Communications Surveys & Tutorials*, 19(2), 954–1001.

Ma, J., Le, F., Russo, A. & Lobo, J. (2015). Detecting distributed signature-based intrusion: The case of multi-path routing attacks. *IEEE Conference on Computer Communications (INFOCOM)*, pp. 558–566.

Malhotra, N. & Chaudhary, A. (2014). Implementation of Database Synchronization Technique between Client and Server. *International Journal of Engineering Science and Innovative Technology (IJESIT)*, 3(4), 460–465.

McAfee, R. P. & McMillan, J. (1987). Auctions and bidding. *Journal of economic literature*, 25(2), 699–738.

Mell, P. & Grance, T. (2011). The NIST Definition of Cloud Computing (Draft). *NIST Special Publication*, 800, 145.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. & Boutaba, R. (2015). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1), 236–262.

Milgrom, P. R. & Weber, R. J. (1982). A theory of auctions and competitive bidding. *Econometrica: Journal of the Econometric Society*, 1089–1122.

Moufakir, T., Zhani, M. F., Gherbi, A., Aloqaily, M. & Ghrada, N. (2022). SFCaaS: Service Function Chains as a Service in NFV Environments. *arXiv preprint arXiv:2203.01098*.

Mounika, M. & Chinnaswamy, C. (2016). A comprehensive review on embedded hypervisors. *computing*, 5(5).

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.

Nobach, L., Rimac, I., Hilt, V. & Hausheer, D. (2017). Statelet-based efficient and seamless NFV state transfer. *IEEE Transactions on Network and Service Management*, 14(4), 964–977.

Peng, T., Leckie, C. & Ramamohanarao, K. (2007). Information sharing for distributed intrusion detection systems. *Journal of Network and Computer Applications*, 30(3), 877–899.

Peuster, M. & Karl, H. (2016). E-state: Distributed state management in elastic network function deployments. *IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 6–10.

Qing, H., Weifei, Z. & Julong, L. (2017). Virtual network protection strategy to ensure the reliability of SFC in NFV. *Proceedings of the 6th International Conference on Information Engineering*, pp. 1–5.

Qu, L., Assi, C. & Shaban, K. (2016). Network function virtualization scheduling with transmission delay optimization. *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 638–644.

Racheg, W., Ghrada, N. & Zhani, M. F. (2017). Profit-driven resource provisioning in NFV-based environments. *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7.

Rajagopalan, S., Williams, D., Jamjoom, H. & Warfield, A. (2013). Split/merge: System support for elastic execution in virtual middleboxes. *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 227–240.

Salameh, H. B., Musa, A., Outoom, R., Halloush, R., Aloqaily, M. & Jararweh, Y. (2019). Batch-based Power-controlled Channel Assignment for Improved Throughput in Software-defined Networks. *IEEE International Multi-Conference on Systems, Signals & Devices (SSD)*, pp. 398–403.

Sandvine. [Publisher at Sandvine Waterloo, Ontario]. (2015). Global Internet Phenomena: Latin America & North America.

Satapathy, P., Dave, J., Naik, P. & Vutukuru, M. (2017). Performance comparison of state synchronization techniques in a distributed LTE EPC. *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–7.

Shodiq, M., Wongso, R., Pratama, R. S., Rhenardo, E. et al. (2015). Implementation Of Data Synchronization With Data Marker Using Web Service Data. *Procedia Computer Science*, 59, 366–372.

Sridharan, S. (2012). A performance comparison of hypervisors for cloud computing. https://digitalcommons.unf.edu/cgi/viewcontent.cgi?article=1275&context=etd. Accessed 25 Dec. 2018.

Truong-Huu, T. & Tham, C.-K. (2014). A novel model for competition and cooperation among cloud providers. *IEEE Transactions on Cloud Computing*, 2(3), 251–265.

Tulloch, M. (2010). *Understanding Microsoft Virtualization Solutions from the desktop to the Datacenter* [2]. Microsoft Press.

Vaquero, L. M., Cuadrado, F., Elkhatib, Y., Bernal-Bernabe, J., Srirama, S. N. & Zhani, M. F. (2019). Research challenges in nextgen service orchestration. *Future Generation Computer Systems*, 90, 20–38.

Varghese, B., Leitner, P., Ray, S., Chard, K., Barker, A., Elkhatib, Y., Herry, H., Hong, C.-H., Singer, J., Tso, F. P. et al. (2019). Cloud futurology. *Computer*, 52(9), 68–77.

Wang, L., Park, K., Pang, R., Pai, V. S. & Peterson, L. L. (2004). Reliability and Security in the CoDeeN Content Distribution Network. *USENIX Annual Technical Conference, General Track*, pp. 171–184.

Woo, S., Sherry, J., Han, S., Moon, S., Ratnasamy, S. & Shenker, S. (2018). Elastic scaling of stateful network functions. *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pp. 299–312.

Xiao, A., Wang, Y., Meng, L., Qiu, X. & Li, W. (2014). Topology-aware virtual network embedding to survive multiple node failures. *IEEE Global Communications Conference (GLOBECOM)*, pp. 1823–1828.

Xie, A., Huang, H., Wang, X., Guo, S., Qian, Z. & Lu, S. (2019). Dual: Deploy stateful virtual network function chains by jointly allocating data-control traffic. *Computer Networks*, 162, 106868.

Yamada, D. & Shinomiya, N. (2019). Computing and Network Resource Minimization Problem for Service Function Chaining against Multiple VNF Failures. *IEEE Region 10 Conference (TENCON)*, pp. 1478–1482.

Yin, S., Huang, S., Liu, H., Guo, B., Gao, T. & Li, W. (2018). Survivable multipath virtual network embedding against multiple failures for SDN/NFV. *IEEE Access*, 6, 76909–76923.

Yu, H., Anand, V., Qiao, C. & Sun, G. (2011). Cost-efficient design of survivable virtual infrastructure to recover from facility node failures. *IEEE international conference on communications (ICC)*, pp. 1–6.

Zhang, L., Wang, Y., Qiu, X. & Guo, H. (2019). Redundancy mechanism of service function chain with node-ranking algorithm. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 586–589.

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.

Zhang, Q., Zhani, M. F., Jabri, M. & Boutaba, R. (2014). Venice: Reliable virtual data center embedding in clouds. *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 289–297.

Zhani, M. F. & Boutaba, R. (2015). Survivability and fault tolerance in the cloud. *Cloud Services, Networking, and Management*, 295–308.

Zhani, M. F. & ElBakoury, H. (2020). FlexNGIA: A flexible Internet architecture for the next-generation tactile Internet. *Journal of Network and Systems Management*, 28(4), 751–795.

Zhani, M. F., Elbiaze, H. & Kamoun, F. (2009). Analysis and Prediction of Real Network Traffic. *Journal of Networks*, 4(9), 855-865.

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