Towards Virtualized Network Functions as a Service

Windhya Rankothge

TESI DOCTORAL UPF / 2017

Director de la tesi

Dr. Jorge Lobo
Dept. of Information and Communication Technologies

The doctoral defense was held on ..... at the Universitat Pompeu Fabra and scored as .....

Dr. Jorge Lobo
(Thesis Supervisor)
Universitat Pompeu Fabra, Barcelona

Dr. Joan Serrat-Fernandez
(Thesis Committee Member)
Universitat Politcnica de Catalunya, Barcelona

Dr. Helena Ramalhinho
(Thesis Committee Member)
Universitat Pompeu Fabra, Barcelona

Dr. Alessandra Russo
(Thesis Committee Member)
Imperial College, London

To Amma, Sudu and Nangi: for giving me wings to fly

This thesis has been carried out at the Department of Information Communication Technologies (DTIC) of Universitat Pompeu Fabra in Barcelona, Spain, from Feb. 2013 to Feb. 2017. It was supervised by Dr. Jorge Lobo.

Work in several parts of this thesis have been carried out in collaborations with International Technology Alliances (ITA). ITA is a collaborative partnership between the US Army Research Laboratories, the UK Ministry of Defense and a consortium of industries and universities in US and UK. A list of collaborators include Dr. Alessandra Russo (Imperial College, London), Dr. Frank Le (IBM T. J. Watson labs), Dr. Jiefe Ma (Adbrain, UK), Dr. Christian Makaya (IBM T. J. Watson labs) and Dr. Helena Ramalhinho (UPF, Barcelona).

This work has been supported by the Department of Information and Communication Technologies (DTIC) PhD fellowship (2013-17), Universitat Pompeu Fabra. Also, it has been sponsored by the ITA, the U.S. Army Research Laboratory, U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing official policies, either expressed or implied, of U.S. Army Research Laboratory, U.S. Government, U.K. Ministry of Defence or U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

Acknowledgements

> "It is good to have an end to journey toward; but it is the journey that matters, in the end."
>
> *Ursula K. Le Guin*

It is been four years !!! For sure time has gone past in a blink and has left me wanting more !!! The main reason for that, I believe is the group of people I have been fortunate to be a part of.

I am extremely grateful to my advisor Dr. Jorge Lobo for the direction, guidance, support and freedom he has given me. His research vision, initiative, perspective and far-sightedness are some qualities that leave me in awe every time. I also thank him for the opportunity he gave me to be a part of the ITA project, which took me to a new level of research.

I wish to express a deep sense of gratitude to Dr. Alessandra Russo, Dr. Frank Le, Dr. Jiefe Ma, Dr. Christian Makaya and Dr. Helena Ramalhinho who have guided me throughout our continuous collaborations. I have greatly benefited from their years of experience in research and the vast area of knowledge. It was a great joy to work with them and learn from their meticulous approach to research spanning in diverse areas.

I highly cherish the friendships that were forged during my time in UPF. I thank my office mates: Arezu and Marcelus, for being there with me during all the stressful times, listening to my complaints and making me feel better. Thanks to my UPF South Asian clan: Praveen, Waqas, Kalpani, Ajay, Sankalp, Shefali, Vignesh, Ratheesh, Ashlesha, Geordie, Princy and Pallabi for making my days in Barcelona sunnier. A note of thanks to UPF-DTIC secretariat, for always being ready to help with a smile, to untangle the infinite web that bureaucracy could be.

I wholeheartedly thank my Barcelona family at Calle Cantabria, 55: Rafiq, Shaaz and cute little prince Waleed, for providing me a home away home for all these years.

I am always grateful to my lecturers in Sri Lanka, who led the foundation to my higher studies and research career, Prof. Gihan Dias and all the lecturers in University of Moratuwa, Dr. Malitha Wijesundara and all the lecturers in Sri Lanka Institute of Information Technology. A big hug to all my friends in Sri Lanka, for keep bringing joy to my life, even we are miles apart.

Last but not the least, I would like to thank my mother for always believing in me and being a constant source of support and strength. My sister, Wimadhya, for the assurance she brings to my life, and my better-half, Tharindu, who will be the co-author of all our life's publications in the time to come !!!

Without you all, nothing would have been possible....

Abstract

Network Function Virtualization (NFV) [1] is a promising technology that proposes to move packet processing from dedicated hardware middle-boxes to software running on commodity servers. As such, NFV brings the possibility of outsourcing enterprise Network Function (NFs) processing to the cloud.

However, for a Cloud Service Provider (CSP) to offer such services, several research problems still need to be addressed. When an enterprise outsources its NFs to a CSP, the CSP is responsible for deciding: (1) where initial Virtual NFs (VNFs) should be instantiated, and (2) what, when and where additional VNFs should be instantiated to satisfy the traffic changes (scaling), (3) how to update the network configurations with minimum impact on network performances, etc. This brings the requirement of a cloud management framework for VNFs and the cloud infrastructure related operations: provisioning, configuring, maintaining and scaling of the VNFs, as well as configuring and updating of the cloud network.

In this thesis we explore three aspects of a cloud management framework for VNF: (1) dynamic resource allocation, (2) VNFs scaling methods and (3) dynamic load balancing.

In the context of dynamic resource allocation for VNFs, we explore two resource allocation algorithms for: (1) the initial placement of VNFs, and (2) the scaling of VNFs to support traffic changes. We propose two approximation approaches (heuristic based): (1) Iterated Local Search (ILS) and (2) Genetic Programming (GP) to implement the resource allocation algorithms. We compare these heuristic based approaches with a traditional resource allocation approach: Integer Linear Programming (ILP). In the context of VNFs scaling methods, we explored three different scaling approaches: (1) vertical scaling, (2) migration and (3) horizontal. We analyse the three scaling methods in-terms of their practical implementation aspects as well as the optimization aspects with respect to the management. In the context of dynamic load balancing, we explore load balancing

approaches that maintain affinity and handle states and sessions of the traffic, so that the requirement of state migration is avoided. We propose a session-aware load balancing algorithm based on consistent hashing.

Resume

La virtualizacin de funciones de redes (NFV) [1] es una tecnologa prometedora que propone mover el procesamiento de paquetes de cajas intermedias de hardware dedicadas al procesamiento especializado de paquetes a mdulos de software que se ejecuta en servidores no especializados. Como tal, NFV crea la posibilidad de externalizar de las redes empresariales el procesamiento hecho por funciones de redes (NFs) a la nube.

Sin embargo, para que un Proveedor de Servicios en la Nube (CSP) ofrezca tales servicios, todava hay que resolver varios problemas. Cuando una empresa subcontrata sus NF a un CSP, el CSP es responsable de decidir: (1) dnde deben instanciarse las NF virtuales iniciales (VNF), y (2) qu tipo, cundo y dnde deben instanciarse VNF adicionales para satisfacer los cambios de trfico (Escalamiento), (3) cmo actualizar las configuraciones de la red con el mnimo impacto en los rendimientos de la misma, etc. Esto requiere de un marco de gestin de la nube para VNFs y las operaciones relacionadas con la infraestructura de nube: provisiona miento, mantenimiento y escalado de les VNS. As como la configuracin y actualizacin de la red en la nube.

En esta tesis exploramos tres aspectos de un marco de gestin de la nube para VNF: (1) asignacin dinmica de recursos, (2) mtodos de escalado para VNFs y (3) balanceo de carga dinmico.

En el contexto de la asignacin dinmica de recursos para VNFs, exploramos dos algoritmos de asignacin de recursos para: (1) la ubicacin inicial de VNFs, y (2) la escala de VNFs para apoyar los cambios de trfico. Proponemos dos mtodos de aproximacin (basadas en heursticas): (1) Bsqueda Local Iterada (ILS) y (2) Programacin Gentica (GP) para implementar los algoritmos de asignacin de recursos. Comparamos estos enfoques heursticos con en un enfoque tradicional de asignacin de recursos: Programacin Lineal Entera (ILP). En el contexto de los mtodos de escalde VNFs, hemos explorado tres enfoques de escala diferentes: (1) escalamiento vertical, (2) la migracin y (3) horizontal. Analizamos los tres mtodos de escalado en

trminos de sus aspectos de implementacin prctica, as como los aspectos de optimizacin con respecto a la gestin. En el contexto del balanceo de carga dinmico, exploramos enfoques de equilibrio de carga que mantienen la afinidad y manejan estados y sesiones del trfico, de manera que se evita el la necesidad de migracin del estado. Proponemos un algoritmo de equilibrio de carga que considera sesiones basado en funciones de hash consistente.

Translated from English by Dr. Jorge Lobo

Contents

List of Figures

List of Tables

Acronyms

Acronyms

CC	Cloud Computing
CSP	Cloud Service Provider
DFS	Depth First Search
GA	Genetic Algorithms
GP	Genetic Programming
ILP	Integer Linear Programming
ILS	Iterated Local Search
IT	Information Technology
MOO	Multi-Objective Optimization
NF	Network Function
NFC	Network Function Center
NFCRMP	Network Function Center Resource Management Problem
NFV	Network Function Virtualization
SDN	Software Defined Networks
SLA	Service Level Agreement
QoS	Quality of Service
VDC	Virtualized Data Centers
VNF	Virtualized Network Function

Chapter 1

Introduction

"A problem well put, is half solved."

John Dewey

Information Technology (IT) executives of enterprises have always faced challenges when it comes to delivering the IT services needed to support changing business goals and demands. But these days they are under more pressure than ever to keep up with an ever-shifting business environment that demands agility, reliability, availability, and security - all at the same time. A big part of the need for the change, focuses on the data center, or in a larger context, the platforms that run business applications. Enterprises have always been struggling with finding ways to optimize data center performance, keep costs down, enhance security, and ensure constant uptime and reliability.

However, the evolution of a sophisticated "as-a-service" mechanism: Cloud computing, has influence the way enterprises think about their data centers. Cloud-based services have become so pervasive that, for many enterprises, it is difficult to imagine using applications and technology infrastructure components without the cloud. Cloud computing, using a grid of servers to support virtualized infrastructure, platforms, or applications gives an enterprise the flexibility it never had before, to respond quickly to opportunities, deploy new applications, or scale up fast to meet growing customer demand. At the same time, it dramatically reduces expenses for hardware, maintenance, and IT staffing. The newest addition to the "as-a-service" of cloud computing is Virtualized Network Functions (VNFs), where the essential network services such as firewalls, intrusion detection systems etc. are offered as on-demand services to the enterprises.

The work presented in this dissertation is at the crossroads of cloud computing and offering VNFs as a service in the cloud computing environment. We now delve into the Introduction chapter of the thesis. First,

we present the motivation for the research work presented in this thesis, and then discuss the opportunities and challenges brought by the research context. Then we introduce the main objectives of the research and highlight the contributions of the thesis. The concluding section of the chapter describes the organization of the thesis in detail.

1.1 Motivation

Traditionally, data centers are physical infrastructures used by enterprises to house computer, server and networking systems and components for the enterprise's IT needs, which typically involve storing, processing and serving large amounts of mission critical data of the enterprise's clients. A data center, often requires extensive redundant or backup power supply systems, cooling systems, redundant networking connections and policy-based security systems for running the enterprise's core applications. Therefore, the data center management involves ensuring the reliability of both the connections to the data center and the mission-critical information contained within the data center's storage. It also entails efficiently placing application workloads on the most cost-effective computing resources available.

However, with the introduction of the cloud concept, cloud computing technologies have started to obtain mass appeal in enterprises's data centers as they allowed data center infrastructure to be leased profitably in an on-demand basis to third parties. They enable data centers to operate like the Internet, through the process of enabling computing resources to be accessed and shared as virtual resources in a secure and scalable manner. Cloud computing can be defined as the delivery of computing services: servers, storage, databases, networking, software etc. over the Internet on a pay-for-use basis [8]. From the perspective of an enterprise, cloud computing provides several benefits. First, cloud computing provides almost immediate access to hardware resources, with no upfront capital investments for the enterprise, leading to a faster time to market in many businesses. Second, cloud computing makes easier for enterprises to scale their services according to client demands. Since the computing resources are managed through software, they can be deployed very fast as new requirements arise or scale resources up or down dynamically through software APIs. Third, cloud computing is often offered with a pricing model that lets the enterprise pay as it goes and for just the services that the enterprise needs [8].

A cloud service is any service made available to users on demand via the Internet from a Cloud Service Provider's (CSP) servers as opposed to being provided from an enterprise's own on-premises servers [9]. Cloud services are designed to provide easy, scalable access to applications, resources and services, and are fully managed by the CSP. A cloud service can dynamically scale to meet the needs of its users, and because the CSP supplies the

hardware and software necessary for the service, and there is no need for the enterprise to provision or deploy its own resources or allocate IT staff to manage the service. Examples of cloud services include online data storage and backup solutions, web-based e-mail services, hosted office suites and document collaboration services, database processing, managed technical support services etc [9].

CSPs such as Amazon, Google, Salesforce, IBM, Microsoft, and Sun Microsystems have established cloud infrastructures to host cloud services in various locations around the world to provide redundancy and ensure reliability in case of site failures [9]. As the cloud computing is enabled by advances in virtualization, service oriented computing, and utility computing, from the perspective of the CSP, managing a cloud infrastructure is a challenging task. There are several requirements for cloud computing to be successful. These include low-cost, Service Level Agreements (SLA) compliance, security guarantees, high availability, energy efficiency, accurate accounting etc. The key to meeting these requirements is effective management of the cloud resources and services.

For the past decade, enterprises have been outsourcing their application services (i.e, web servers, mail servers etc.) to the CSP by requesting physical resources: Virtual Machines (VMs) to run these services. However, as well articulated in [10], moving applications from private data centers to cloud centers is complicated by the fact that many of these applications require network-based services such firewalls, load balancers, intrusion detection systems, proxies, etc. According to the study done in [6], a very large enterprise have average of 1946 total middle-boxes and a small enterprise have an average of 10.2 middle-boxes that are processing network-based services. The deployments of these middle-boxes are large scaled and costly, requiring high up-front investment in hardware: thousands to millions of dollars in physical equipments. Furthermore, managing these heterogeneous middle-boxes requires broad expertise and consequently a large management team. [6] shows that even small networks with only tens of middle-boxes typically required a management team of 6-25 personnel. Thus, middle-box deployments incur substantial operational expenses in addition to hardware costs.

[6] highlights several key challenges that enterprise administrators face with middle-boxes deployment and maintenance. First, the deployments of middle-boxes are complex as they involve large number and diverse types of middle-boxes: a very large enterprise can have average of 1946 total middle-boxes, with seven different types (i.e, firewalls, load balancers, intrusion detection systems, etc.). Second, the management of these middle-boxes also a complex task. To maintain a smooth operation level with the enterprise, administrators must monitor the middle-boxes 24 hours, to identify failures or overloads. Once a failure or overload is identified, administrators must act accordingly, so that the impact on the enterprise operations is minimum and damages are minimized. This might involve finding the cause of the failure, repairing the middle-box or even

deploying a new middle-box. Furthermore, network administrators often face problems when upgrading the network with middle-boxes. Deploying new features in the network entails deploying new hardware infrastructure and middle-boxes. Each time an enterprise decides for a new deployment, administrators must select between several offerings, weighing the capabilities of devices offered by numerous vendors. Administrators must evaluate, select, purchase, install, and train to maintain new appliances.

So from the perspective of both the enterprise and network administrators, deploying and managing middle-boxes are very challenging: (1) they are large deployments with high capital and operating expenses, (2) they have complex management requirements inflating operation expenses, and (3) they often cause failures and overload of physical infrastructures. Thus, the researches argue these factors make middle-boxes good candidates for the cloud, where both the enterprises and CSPs can get benefits from a "middle-boxes as-a-service" business model.

There are several advantages that an enterprise can receive when it outsources middle-boxes processing to the cloud. First, outsourcing middle-box processing can reduce hardware costs. Out-sourcing eliminates most of the infrastructure at the enterprise, and a CSP can provide the same resources at lower cost due to economies of scale. Second, out-sourcing eliminates the frequent middle-boxes upgrade problem. Enterprises sign up for a middle-box service and how the CSP chooses to upgrade hardware is orthogonal to the service offered. Third, out-sourcing gets rid of the requirement of 24 hours monitoring and recovering from failures and overloads. The CSP monitors utilization and failures of specific devices, and reacts to the situation accordingly without the interaction of enterprise administrators. Fourth, a cloud-based capability to elastically provision resources avoids overload, by enabling on-demand scaling and resolves failure with standby devices without the need for expensive over provisioning. Therefore, from an enterprise perspective, outsourcing middle-boxes can cut deployment expenses, simplify management for enterprise administrators, and can provide elastic scaling to limit failures.

Network Function Virtualization (NFV) [1] is a promising technology that proposes to move network-based services from dedicated hardware middle-boxes to software running on commodity servers: Virtualized Network Functions (VNFs). NFV technologies decouples the Network Functions (NFs), such as firewalls, intrusion detection systems, proxies etc., from proprietary hardware appliances, so that they can be hosted on virtual machines (VMs). NFV utilizes standard virtualization technologies and brings several advantages. First it reduces capital investment, energy consumption and time to market a new service. Next, it enables new properties such as elastic scaling. NFV allows the flexible increase and decrease of resources (e.g., CPU, memory) allocated to each VNF instance, and the number of VNF instances to satisfy the dynamic traffic changes.

With all these advantages, NFV technologies and VNFs concept bring a

great opportunity to CSPs, by opening the path to a new "as-a-service" business model, where VNFs can be offered as a service to enterprises, on demand basis. This enhance the business scope of CSPs, as now CSPs can offer VNFs as a service, along with the application services. The enterprises also get the benefits from this "VNFs as-a-service" concept, because they get the opportunity to out-source their network-based services, which are currently running as hardware middle-boxes.

1.2 Opportunities and challenges

The concept and collaborative work on NFV was established in 2012 when a number of the world's leading Telecommunication Service Providers (TSPs) (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica and Verizon) started their joint work to develop the required standards for NFV as well as to share their experiences of its development and early implementations [1]. They selected the European Telecommunications Standards Institute (ETSI) to be the home of the Industry Specification Group for NFV (ETSI ISG NFV) [1]. Even with all the anticipated benefits, and despite the immense speed at which it is being accepted by both academia and industry, NFV is still in early stages. Therefore, for a CSP to offer VNFs as a service, there are many research questions that need to be addressed and standard practices which should be established.

For a CSP, to achieve the expected benefits from NFV, the physical resources of the cloud infrastructure should be used efficiently. Therefore, from the CSP's perspective, the efficient management and orchestration of VNFs is crucial for the success of the business. This brings the requirement of a cloud management framework for VNFs and the cloud infrastructure related operations: provisioning, configuring, maintaining and scaling of the VNFs, as well as configuring and updating of the cloud network.

Cloud resource allocation process is the most challenging responsibility of a cloud management platform. The cloud resources allocation includes allocation of computing resources (i.e, CPU, memory, etc.) and allocation of network resources (i.e, bandwidth). Existing work on cloud resource allocation for VMs are not suitable for cloud resource allocation for VNFs for several reasons [11].

First, traditionally enterprises outsource their application services to the CSP by requesting physical resources (VMs) to run these services. The client request comes as number of VMs required along with the required specifications of the VMs (i.e, CPU and memory needed). Therefore, the work on the cloud resources management were focused on provisioning of one or more VMs in the cloud, as requested by the client. However, for the VNFs, the client request for a VNF comes with the specification of the expected traffic rate. The CSP is responsible for deciding number of VNF instances needed to process the expected traffic rate and the specifications of the VMs that are going to host the VNF instances [9].

Second, optimizing the placement of VMs in the cloud infrastructure for the application services tend to be node-centric as VMs are end-points. But optimizing the placement of VNFs is network-centric as their provisioning are normally large middle points in a network [11]. Third, the requests of VNFs normally involves service chains of VNFs rather than individual VNFs. The placement of a service chain requires, initially the allocation of computing resources (cpu, memory, etc.) for the VNFs, which is basically selecting servers to place the VNFs. Then the allocation of network resources (bandwidth) which is basically selecting paths to route traffic flow from one VNF to next VNF in the chain, within the cloud [11]. More over, most of the existing work on cloud resources allocation, consider only a part of the problem, by optimizing either host or bandwidth resources, but do not provide an integrated view of multiple optimization objectives (i.e, computation, storage, networks, energy, etc.) [12].

All these reasons call for efficient resource allocation algorithms for cloud resources allocation for VNFs, that can optimize different goals of the CSP such as to maximize resource utilization, maximize number of requests accepted, minimize energy consumption etc [12]. A popular technique that has been used for cloud resources allocation is to model the resource allocation as an Integer Linear Programming (ILP) optimization problem. There is the intrinsic constraint that ILP optimization is a NP-complete problem, and even when solutions are obtained for special classes and smaller networks, it might be too slow for continued adjustments of the system configuration causing inappropriate hysteresis in the reaction [13, 14]. We believe it is more realistic to look for good feasible configuration and do not expect to find optimal solutions like the ones returned by ILP. Therefore, we need to explore approximations techniques: heuristics based resource allocation algorithms.

Following the initial instantiation, NFV enables new properties such as elastic scaling: NFV allows the flexible increase and decrease of resources (e.g., CPU, memory) allocated to each VNF instance, and the number of VNF instances, to satisfy the dynamic and fluctuating service demands. From the CSP's perspective, to take the benefits from the elastic property of VNFs and offer flexible services to the clients, the CSP should be able to **scale the VNFs and allocate/de-allocate resources dynamically**. This can ensure that the Service Level Agreements (SLA) are not violated over the unpredictable traffic changes. Despite some initial efforts [15, 16], scaling of VNFs still presents many open challenges.

One of the main challenge is to monitor the network and computing resources and determine when to scale the VNFs according to the dynamic traffic changes. Finding the exact VNF(s) or path(s) which are causing the bottleneck is essential, because of the costs involved in running new VNFs as well as the impact of reallocating VNFs and traffic flows that may cause traffic lost [10]. Also, it is important to decide the right type of the resources (computing and network) and amount of resources to increase/decrease to achieve the demand and avoid the potential for some

kind of thrashing phenomenon [10].

After deciding, when and what additional VNFs should be instantiated to satisfy the dynamic traffic changes, the next challenge is the **dynamic resource re-allocation**. As it is an on-line version of the resource allocation problem, we believe that it is more realistic to look for approximation techniques: heuristics based resource allocation algorithms. The resource re-allocation for the scaling requirements of existing VNFs happens during the middle of the operations, where the already deployed VNFs are processing traffic. Therefore, the resource re-allocation for the scaling requirements of existing VNFs is a time critical on-line problem. The solutions have to be given in the order of milliseconds, so that the disturbances and damages to current operations are minimal.

The second challenge is **deciding the scaling method**: i.e., whether to use vertical scaling (allocation/release of computing and bandwidth resources to/from a VNF instance) or migration (running VNFs are paused, serialized and transferred to different servers with more resources) or horizontal (installation/removal of VNF instances) [17]. In the case of horizontal scaling, it needs a gateway and a **dynamic load balancer** that distributes the traffic to multiple VNF instances, which makes it resource-consuming, as well as complex. The third challenge is how to resolve potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimize usage of computing and network resources [17].

Finally, for the dynamic resource allocation and dynamic load balancing, it is important that both computing resources and network configuration of the cloud infrastructure, should be able to be updated concurrently, easily and fast [18]. However, **dynamic update of network configuration** introduces a new challenge to the CSPs, because when updating the network configuration, the CSPs should try to avoid any inconsistencies in transient traffic and to minimize traffic lost, so that the SLAs are not violated. In this context, we believe that use of Software Defined Network (SDN) infrastructure is a very appropriate approach, as it allows reconfigure the physical network easily. SDN is a networking technology that decouples the control plane from the underlying data plane and consolidates the control functions into a logically centralized controller [19]. NFV and SDN are mutually beneficial, highly complementary to each other, and share the same feature of promoting innovation, creativity, openness, and competitiveness [20].

To summarize the challenges that CSPs face when offering VNFs as service, and managing the cloud infrastructure:

1. Allocating computing and network resources
2. Monitoring the computing and network resources
3. Deciding the scaling requirements of VNFs

4. Scaling VNFs
5. Dynamic load balancing for VNFs
6. Updating the network configurations

1.3 Research Objectives

As described in the previous section, there are many challenges that CSPs face when offering VNFs as service; specifically with cloud infrastructure related operations: provisioning, configuring, maintaining and scaling of the VNFs, as well as configuring and updating of the cloud network. Each of these challenges is a broad research topic that can be approached from a number of perspectives and different academic disciplines. Therefore, it is necessary to define and delimit the scope of the research presented in the thesis, while identifying the objectives of the thesis clearly. The broad objectives of the thesis are listed below:

1. The first objective of the research was to explore **resource allocation** approaches for VNFs. We looked into the problem of resource allocation in two aspects: (1) initial resource allocation for new VNFs requests and (2) resource re-allocation for scaling requirements of existing VNFs.
2. The second objective of the research was to explore **VNFs scaling methods**: (1) vertical scaling (allocation/release of computing and bandwidth resources to/from a VNF instance), (2) migration (running VNFs are paused, serialized and transferred to different servers with more resources) and (3) horizontal (installation/removal of VNF instances). We wanted to explore advantages as well as disadvantages of each of the method, in-terms of practicality of the implementation as well as their effect on the cloud resources optimization.
3. The third objective of the research was to explore on **dynamic load balancing** mechanisms that can be used with the horizontal scaling of VNFs. The concept of horizontal scaling as the is to create/delete VNFs instances to satisfy the scaling requests, therefore the total traffic has to be balanced between multiple instances. We wanted to explore on dynamic load balancing approaches that can be used in a cloud based infrastructure.

1.4 Research contributions

As explained in the previous section, there were three main objectives for our research work. Each of the objective is a broad research topic that can be approached from different perspectives and disciplines. Therefore, it is essential to highlight the contributions of this thesis clearly, briefly

describing the approaches we explored to achieve the objectives. The main contributions of the thesis are described below:

Resource allocation for VNFs

The first set of contributions of this thesis, are to the context of resource allocation for VNFs, which was the first objective of the thesis.

As mentioned earlier, existing work on cloud resource allocation for VMs are not suitable for cloud resource allocation for VNFs. Optimizing the placement of VMs in a cloud tend to be node-centric as VMs are end-points. Optimizing the placement of VNFs is, in contrast, network-centric – their provisioning normally involves service chains of VNFs rather than individual VNFs. The placement of a service chain requires allocation of server resources (for the VNFs), as well as allocation of network resources (paths) to route traffic flow from one VNF to next VNF in the chain, within the cloud. Most of the existing work on placement of VNFs, try to optimize either host or bandwidth resource, but do not provide an integrated view of computation, storage and networks optimization [21]. Furthermore, most of the research work on resource allocation focus only on a part of the problem: resource allocation for the initial placement of VNFs, but do not consider the resource re-allocation for the scaling requirements of existing VNFs. Therefore, despite some initial efforts [15, 16, 22] the resource re-allocation for scaling VNFs presents still many open challenges. For example, one of the challenges is how to achieve scaling: i.e., whether to use horizontal, vertical or migration scaling technologies. A second challenge is how to resolve potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimize usage of server and network resources [17].

In this thesis, we focused on resource allocation for two situations: the resource allocation for the initial placement of VNFs and (2) the resource re-allocation for the scaling requirements of existing VNFs. Once the client request for a set of VNFs has been accepted, the resource allocation for the initial placement of these VNFs can be done in the order of minutes, and then the new VNFs can be deployed accordingly. But the resource re-allocation for the scaling requirements of existing VNFs happens during the middle of the operations, where the already deployed VNFs are processing traffic. Therefore, it is a time critical on-line problem. The solutions have to be given in the order of milliseconds, so that the disturbances and damages to current operations are minimal. Therefore, we specifically explored the resource re-allocation for the scaling requirements of existing VNFs problem in depth, as it is an interesting on-line resource allocation problem. There are two aspects to look at the problem of resource re-allocation for the scaling requirements of existing VNFs. First, is to monitoring the network and computing resources and determining

when to scale the VNFs according to the dynamic traffic changes: scaling triggers. Second is to re-allocating resource: increase/decrease resources according to the changed traffic amount with minimal impact to the current operations of the cloud. Our work in this thesis is limited to specifically the resource re-allocation process, we did not explore the problem of scaling triggering.

We formulated the resource allocation problem as a set of Integer Linear Programming (ILP) equations; we called it as the Network Function Center Resource Management Problem (NFCRMP). It addresses both (1) the initial resource allocation for new VNFs provisioning, and (2) the resource re-allocation for the scaling of existing VNFs to support traffic changes. For resource allocation for new VNFs, the goal was to minimize the required resources (i.e., number of servers, number of links, and average link utilization). For resource allocation for scaling of existing VNFs, the aim was to adjust the resources to satisfy the traffic changes and, at the same time, minimize the number of configuration changes to reduce potential service disruptions, and performance degradation.

We explored approximation approaches: heuristic based approaches to implement the resource allocation algorithms that can give reasonable solutions fast. First we used the **Iterated Local Search (ILS)** approach, because ILS is one of the most popular single-solution based meta-heuristics due to its simplicity but at the same time powerful approach [23]. However, for the ILS based resource allocation algorithms to be more efficient, they have to be designed with conditions based on the network architecture. In other words, efficiency of ILS procedure is not agnostic to the network architecture. Therefore, we propose another heuristic based approach: **Genetic Programming (GP)** to implement the resource allocation algorithms. GP based algorithms are agnostic to the network architecture, and can be used for any general network architecture. Also, they are easy to implement and popular in general optimization domain. Both the ILS and GP processes rely on either (1) a Depth First Search (DFS) or (2) a random approach to find the initial solution. We compare the performances of both ILS and GP approaches, using DFS and random approaches as the baseline, for three types of network architectures: 4-fat tree [24], BCube [25] and VL2 [26]. We used a more realistic traffic pattern generated based on [27]. To achieve scaling, we adopted to a simple scaling procedure: first we tried to allocate resources for a vertical scaling, and if it failed, then we tried to allocate resources for a migration scaling. Our results showed that both ILS and GP algorithms can decide server and network allocations for hundreds of policies (around 400 VNFs) in a 128 server environment and find reasonable solutions in milliseconds. Moreover, ILS produced better results than GP for all three types of architectures. Furthermore, we implemented the ILP formulation of the NFCRMP in CPLEX [28]. We used the optimal solutions found by ILP implementation to have a comparison with the solutions provided by the approximation approaches: ILS and GP. Both the ILS and GP found the

optimal solutions in a small scaled network in the order of micro-seconds while ILP took hours.

VNFs scaling methods

The second set of contributions of this thesis, are to the context of VNFs scaling methods, which was the second objective of the thesis.

As mentioned earlier, the existing research work on scaling of VNFs is very limited, most of them focus only on the complexities of the scaling technologies. They do not consider the resource allocation or optimization aspects of these scaling technologies. Therefore, we explored different scaling approaches and the optimization perspectives: (1) **vertical scaling** (allocation/release of computing and bandwidth resources to/from a VNF instance), (2) **migration** (running VNFs are paused, serialized and transferred to different servers with more resources) and (3) **horizontal** (installation/removal of VNF instances).

We conducted experiments to check how the optimization is effected by the scaling approach and the optimization objectives. We expanded our ILS algorithms with the three scaling methods. We considered a single optimization goal: maximize the accepted bandwidth of scaling requests while ensuring a new constraint: that the delay experienced by each packet of an accepted scaling request (inside the NFC), does not exceed its relative deadline. We compared the different characteristics of the solutions provided by scaling approaches such as accepted bandwidth ratio, resource utilization etc.

The Vertical scaling approach had the highest percentage of bandwidth rejection: average of 49.6%, and therefore accepted lowest number of scaling requests. The next was migration scaling approach, with and average of 3.89% of bandwidth rejection. The horizontal scaling approach had the lowest percentage of bandwidth rejection: average of 0.12%, and accepted highest number of scaling requests. Therefore, it can be considered as the best scaling approach, in terms of the optimization goal of accepting maximum bandwidth requests as much as possible. Our observation is that vertical scaling is always limited by the spare computational resources of the VNF's current server, and therefore, vertical scaling is rarely able to satisfy the scaling requirements. The Migration scaling and horizontal scaling have more freedom in scaling, and able to satisfy most of the scaling requirements.

Dynamic load balancing for VNFs

The third set of contributions of this thesis, are to the context of dynamic load balancing mechanisms, which was the third objective of the thesis.

With VNFs, the load balancing approaches should look beyond the unidirectional flows and take care of the sessions: which is bi-directional traffic flows between two nodes. This is because, unlike layer 3 forwarding, many

VNFs such as firewall, proxy, and VPN perform stateful packet processing: session based packet processing [29]. Therefore, these VNFs require affinity, where traffic for a given flow must reach the instance that holds that flow's state [30]. In such cases, splitting traffic to balance the load, requires extra measures to preserve affinity. Existing solutions that maintain affinity mostly depend on state migration techniques: moving the relevant state from one instance to another. However, these systems require that NF vendors adopt a new programming model or add a non-trivial amount of code to existing NF implementations. Therefore, we argue that it is more appropriate to design the load balancing algorithm in a way that the algorithm itself maintain affinity and handle states and sessions, so that the requirement of state migration is avoided. We proposed a load balancing algorithm that controls sessions, using consistent hashing techniques.

We have conducted preliminary experiments for a small scaled network, to verify the accuracy and basic performances of the algorithm. We have considered three scenarios: (1) the system starts with more than one VNF instance, so from the beginning the traffic has to be balanced between multiple VNF instances, (2) the system starts with more than one VNF instance, but some VNF instances are removed dynamically, so the load has to be re-balanced and (3) the system starts with n VNF instances, but more VNF instances are added dynamically, so the load has to be re-balanced. Our preliminary results show that the proposed session-aware hashing algorithm balances load evenly (within 1.5% of ideal) within a reasonable time frame.

1.5 Organization and thesis outline

The dissertation has seven chapters. Each chapter is written on a major topic of the thesis and is aimed to be self contained with a short introduction, content, and a summary. The writing style followed in the thesis is a mixture of both active and passive voice. Most of the dissertation derives content from published research papers describing the work done by collaborative teams. Hence, the word "we" refers to the author and when applicable, additionally includes the co-authors and collaborators in research papers.

The rest of the dissertation is organized as follows:

Chapter 2 provides an overview of the background material necessary for the thesis. It introduces a concrete terminology for NFV concepts and technical concepts useful to understand the thesis work. It then provides an overview of the state of the art for cloud management and offering VNFs as a service. The content of the chapter is compiled from several external sources cited appropriately when necessary.

Before moving to the research aspects of the thesis, Chapter 3 introduces the experimental environment, data-sets and use cases we used to explore different management aspects of VNFs. We present our experimental NFV

platform: the Network Function Center (NFC) in Section 3.1 and its architecture in Section 3.2. Section 3.3 describes the five key modules of the NFC Management System: (1) Resource Manager, (2) Topology Manager, (3) Flow Manager, (4) Elasticity Manager and (5) Rules Generator. Each module is responsible for different tasks, such as: resource allocation, monitor cloud infrastructure, decide scaling requirements, scale VNFs, updates the network configurations, etc. Conceptually, the Resource Manager, Topology Manager, Elasticity Manager and Flow Manager can be seen as management applications. The Rules Generator as an extension to the network operating system. Section 3.4 presents the use cases and data set generation process, that was necessary for design and evaluation of the algorithms used by the management system.

After the introduction to our experimental environment, Chapter 4 to Chapter 6 focus on describing our research contributions; specifically on achieving each objective of the research.

Chapter 4 describes our work on exploring resource allocation approaches for VNFs. It introduces the Resource Manager module of NFC, which is responsible for resource allocation for: (1) provisioning new VNFs requests and (2) scaling out/in of existing VNFs. Section 4.1 formulates the Network Function Center Resource Management Problem (NFCRMP) as a set of ILP equations for new VNFs provisioning and dynamic scaling, and then present the evaluation of the implementation in CPLEX [28]. Section 4.2 and Section 4.3 describe the ILS based and GP based implementations of the Resource Manager respectively, with a comprehensive analysis for each approach based resource allocation algorithms. In Section 4.4, we present an overall analysis for the Resource Manager module, comparing different approaches: (1) DFS, (2) Random, (3) ILP, (4) ILS and (5) GP.

Chapter 5 discusses our work on exploring VNFs scaling approaches, interms of the practical aspects as well as optimization aspects. We focus on three different scaling approaches for VNFs including: (1) vertical, (2) migration and (3) horizontal. Section 5.2.1 presents the extended version of the ILS based resource allocation algorithm for scaling, with three scaling methods. Section 5.2.3 presents an analysis on how the optimization of the algorithm is effected by the scaling approach and the optimization objectives.

Chapter 6 discusses our work on exploring dynamic load balancing approaches for VNFs, where we propose a load balancing algorithm with session control, based on consistent hashing techniques. We introduce basic consistent hashing concepts in Section 6.1 and describe the proposed session-aware load balancing algorithm in Section 6.2. Section 6.3 and Section 6.4 present the experimental set-up that we used to evaluate the proposed session-aware load balancing algorithm and the preliminary results of the evaluation. Furthermore, Section 6.5 discusses our work on exploring dynamic network configurations update mechanisms, where we present two mechanisms found in existing works: (1) configuration rules

are updated in all switches simultaneously, and (2) versioning tags are used to maintain per-flow consistency [31]. It provides a simple comparison between the two methods, in terms of the packet lost.

Finally, in Chapter 7 we present the overall summary of the thesis, including the key results and conclusions, and then discuss possible future perspectives for offering VNFs as a service as well as managing cloud for VNFs.

This thesis also contains two appendix sections. In Appendix A, we list the relevant publications by the author. In Appendix B, we describe the data modelling process to generate the use cases and data sets, and present the links to resources of the thesis (data, code, examples).

Chapter 2

Background

"I came, I saw, I conquered."

Julius Caesar

With the introduction of cloud computing technologies, enterprises have been able to outsource their application services (i.e, web servers, mail servers etc.) to the CSP by requesting physical resources (VMs) to run these services. The client request for the VM comes with the specifications of the VM (i.e, CPU and memory needed). Therefore, the initial work on the cloud management were focused managing aspects relevant to VMs; such as provisioning of VMs, balancing the work load among the VMs etc.

However, with the recent developments in NFV technologies, enterprises are able to outsource their network services too. CSPs are able to offer VNFs as a service to clients. The cloud management for VNFs has gained a significant attention from both industry and academia in the recent years. The client request for the VNF comes with the specifications of the expected traffic load, and therefore, based on the traffic load, the CSP has the flexibility to decide the specifications of the VM that needs to implement the VNF. Furthermore, over the time, with dynamic traffic changes, CSP has to allocate/re-allocate resources for the existing VNFs, or instantiate new VNFs to satisfy the new traffic demand and guarantee that the SLAs are not broken. Therefore, the research work on cloud management for VNFs are focused on aspects such as provisioning of VNFs, scaling of VNFs, traffic load balancing among VNF instances etc.

In this chapter, we provide the necessary research background for understanding the work presented in the dissertation. Furthermore, we present our review of the existing literature related to the cloud management, specifically for offering applications and VNFs services. Also, we present

a summary of how the work in this thesis is different from the existing work. The main objectives of this chapter are to:

1. Establish a consistent terminology for VNFs concepts : Section 2.1
2. Describe the technical concepts necessary to understand the algorithms and methods presented in the dissertation: Section 2.1
3. Present an overview of the state of the art for the cloud management: Section 2.2
4. Discuss the difference between existing work and the work presented in the thesis: Section 2.3

2.1 Virtualized Network Functions (VNFs) concepts

The **Cloud Computing**, often referred to as simply "the cloud", is the delivery of on-demand computing resources, everything from applications to data centers, over the Internet on a pay-for-use basis [8]. Cloud based applications or **Software as a Service (SaaS)** run on distant computers in the cloud that are owned and operated by a **Cloud Service Provider (CSP)** and that connect to users' computers via the Internet and, usually, a web browser [8]. **Platform as a Service (PaaS)** provides a cloud based environment with everything required to support the complete life cycle of building and delivering web based applications without the cost and complexity of buying and managing the underlying hardware, software, provisioning, and hosting [8]. **Infrastructure as a Service (IaaS)** provides companies with computing resources including servers, networking, storage, and data center space on a pay-per-use basis [8]. Therefore, cloud computing enables many organizations to outsource the management of the physical infrastructure for their IT needs to CSPs.

Moving applications from private data centers to cloud centers and outsource them to the cloud, is complicated by the fact that many of these applications require network-based services such as firewalls, load balancers, intrusion detection systems, proxies, etc. **Network Functions Virtualization (NFV)**, by making the NF a first-class virtual citizen, holds strong promises for both enterprise data centers and sophisticated network services. NFV proposes to virtualize the network based services, traditionally running on proprietary hardware appliances, and instead have them operate as virtual software components on commoditized hardware: **Virtualized Network Functions (VNFs)** [1]. In addition to reducing costs and time to market, NFV enables new properties such as elastic scaling: NFV allows the flexible increase and decrease of resources (e.g., CPU, memory) allocated to each VM instance, and the number of VM instances, to satisfy the dynamic and fluctuating service demands. As such,

the enterprises are able to outsource their NFs processing to the cloud and CSPs are able to offer VNFs as a service to the clients.

To offer VNFs as a service in the cloud and guarantee that the **Service Level Agreements (SLA)** are not violated over the traffic changes, the CSP should be able to scale the allocated resources dynamically. Therefore, both computing resources and network configuration of the cloud, should be able to be updated concurrently [18]. In this context, use of **Software Defined Network (SDN)** infrastructure is a very appropriate approach, as it allows reconfigure the physical network easily. In SDN, the control plane is decoupled from hardware data plane and given to a software application called a controller. The controller is the core of an SDN network and it lies between network devices and applications. Any communications between applications and devices have to go through the controller. A controller uses SDN protocols for the communications between the controller platform and data plane devices [19, 20]. SDN paradigm brings several benefits to network management compared to static traditional methods. It is much easier to introduce new concepts to the network through the controller, as it is easier to change and manipulate than using a fixed set of commands in proprietary network devices. With the centralized approach of network configuration and network management, operators do not have to configure all network devices individually to make changes in network behaviour.

Scaling of computing resources for a VNF instance, can be done in difefrent ways: (1) vertical, (2) migration and (3) horizontal [7]. **Vertical scaling** is allocation/release of host and bandwidth resources to/from a VNF instance, whereas **horizontal scaling** is installation/removal of VNF instances or paths. Vertical scaling is a basic feature of VMs, which adjusts logical partitions of multiple metrics (i.e. CPU, Memory, Bandwidth). So vertical scaling of VNFs can be done adjusting the existing VNF instance with new metrics of capacities for CPU, Memory and Bandwidth. However, horizontal scaling changes the number of VM instances, which involves running VNF instances on two or more separate VMs hosted on the same or different servers. Further, live migration of VNF instances can be employed to scale both bandwidth and host resources. With **live migration** of VNFs [32], the running VNFs are paused, serialized and transferred to a different servers with more resources, where they are once again scheduled for execution with additional resources.

When considering the real implementation, existing work has shown that each of the aspects has advantages as well as disadvantages [7]. Vertical scaling is better than horizontally scaling because: (1) needs less time for reconfiguration as it needs only metrics adjustment, (2) does not need additional software licenses, (3) does not affect the quantity of VNF instances and (4) does not introduce a coordination or a traffic distribution among multiple VNF instances. However, vertical and migration scaling is limited by capacity of the server or link, because it cannot increase the resources more than the maximum capacity. Also, the consolidation

after vertical scaling is more complicated because the fragments caused by it are likely to be irregular. Even though the live migration of VNFs is a complex process, it offers many benefits from the resource management perspective, enabling global scheduling objectives and consolidating VNFs on a minimal number of servers. In the case of horizontal scaling, it needs a gateway and a dynamic load balancer that distributes the traffic to multiple VNF instances, which makes it resource-consuming, as well as complex.

2.2 State of the art

After the brief description of concepts and terminologies relevant to the work presented in this thesis, in this section, we will present our review of the existing literature related to the offering VNFs as a service and cloud management.

We have explored existing work on cloud management in two aspects: (1) resource management for placement and provisioning and (2) load balancing in the cloud infrastructure. With respect to cloud resource management for placement and provisioning, we specifically looked in to following two aspects: (1) early work on resource management for traditional VM placement and (2) recent work on resource management for VNFs placement. With respect to load balancing in the cloud infrastructure, we specifically looked in to following two aspects: (1) early work on cloud load balancing with SDN and (2) recent work on load balancing between VNF instances.

In this Section, we describe the existing literature related to the offering VNFs as a service and cloud management in following aspects:

1. Work on offering VNF as a service: Sub-section 2.2.1
2. Early work on resource management for traditional VM placement: Sub-section 2.2.2
3. Recent work on resource management for VNFs placement: Sub-section 2.2.3
4. Early work on cloud load balancing with SDN: Sub-section 2.2.4
5. Recent work on load balancing between VNF instances: Sub-section 2.2.5

2.2.1 VNFs as a service

NFV has gained a significant attention from both industry and academia along with SDN. [33] and [12] discusses the NFV concepts, state-of-the-art and research directions. Virtualized data centers have become a promising infrastructure for data storage for large volume of data, and provide

the platform for deployment of diversified network functions and applications. They are envisioned to provide better management flexibility, lower cost, scalability, better resource utilization and energy efficiency [34–36]. With the popularity of virtualized data centers, [6] discusses the possibility of outsourcing enterprise middle-box processing to the cloud. FeatureAPI [37] introduces a policy language to map policies onto the underlying cloud network, so that external feature providers can easily provide NFs to enterprises. [38] highlights the customer expectations when they outsource the NFs. Stratos [10] is the first work that presents a framework for external NF providers. Stratos assumes a cloud service exists and to handle traffic flows and redirections, it associates each middle-box with a virtual switch that provides network functionalities.

Traditionally NFs have been implemented as hardware based middle-boxes. CoMB [39] proposes an architecture which implements these middle-boxes as software-based, virtualized entities. [40, 41] present platforms to manage software-based middle-boxes. Provisioning requests from cloud's users involves service chains of VNFs [29, 42–44]. The placement of these VNF chains in physical machines and use of the network bandwidth are therefore crucial for performance of a NFC [45].

Many mechanisms have been proposed to support elasticity of VNFs through horizontal, vertical scaling and migration techniques. Amazon Auto Scaling Group [46] offers tenant controlled horizontal scaling based on tenant-defined thresholds. Microsoft Azure [47] adapts the number of instances based on time, history, or size of workload. CloudScale [48] and PRESS [49] scale by vertical scale: releasing or allocating CPU resources while ignoring network resources. However, vertical mechanisms are limited by the capacities of individual physical machines [50]. Furthermore, changing memory or computing resources on-the-fly is not supported in most cases. Therefore, CSPs do not encourage vertical scaling mechanisms. Migration is a popular technique to achieve elastic VNF placement and better server and network consolidation. Many studies have assessed the overhead of live migration [32, 51–53]. More recently, Xen [54] and VMWare [55] have implemented live migration of VMs that involves extremely short down-times ranging from tens of milliseconds to a second. [56] pointed out that migrating an entire OS and all of its applications as one unit allows to avoid many of the difficulties faced by process level migration approaches, and analyzed the benefits of live migration.

2.2.2 Cloud resources management for VM placement

Initial work on the cloud resources management were focused on provisioning of one or more VMs in the cloud, as enterprises were only outsourcing their application service to the CSP, not the network services. From the CSP perspective, optimizing the placement of VMs in the cloud tended to be node-centric as VMs are end-points. In the very beginning, the work

on the VM placement problem typically assumed that VMs are assigned static shares of a server's managed resources (i.e, CPU and/or memory).

VM placement and provisioning

When a request for the provisioning of one or more VMs is made by a client, the CSP's resource management system schedules the VMs by placing them onto servers. Selection of the servers is decided according to the CSP's current management objectives. Given that the initial work on the problem assumed that VMs are assigned static shares resources, the placement of VMs onto servers is related to the vector bin packing problem, which can be used to model static resource allocation problems where the resources used by each item are additive. To achieve server consolidation, the optimal placement is one where the items: VMs are packed into a minimum number of bins: servers, such that the vector sum of the items received by any bin does not exceed the bins resource limit. The vector bin packing problem and its variants are NP-hard problems [57]; thus, heuristic algorithms were proposed [9].

Most of the proposed heuristics are based on greedy algorithms using simple rules, such as First Fit Decreasing (FFD) and Best Fit Decreasing (BFD). [58] explored variants of the FFD algorithm and proposed a new geometric heuristic algorithm which scales to large data centers without a significant decrease in performance. [59] proposed a new generic algorithm, Reordering Grouping Genetic Algorithm (RGGA), which they apply to VM placement. [60] propose a solution for VM placement that accounts for the ability of some hypervisors to dynamically allocate a pool of unreserved CPU and memory resources between a set of contending VMs. [61] provides a solution for private clouds in which VMs are placed on those servers that already host VMs with complementary workloads. [62] propose a VM placement solution in which multiple VMs are consolidated and provisioned together, in order to exploit the statistical multiplexing amongst the workload patterns of the individual VMs.

More recent works have recognized that, provisioning requests from clients involve sets of VMs, rather than single VMs. In such cases, placement of the set of VMs is not only constrained by the collated resource requirements, but also by a range of possible placement constraints indicated by the client. [63] consider a range of constraints relating to full deployment, anti-collocation and security; [64] consider demand constraints (i.e., lower bounds for VM resource allocations) and placement constraints (i.e., collocation and anti-collocation); [65] considers similar constraints, adding some that indicate whether a set of VMs can be deployed across more than one data center in a geo-distributed cloud environment. [66] assume elastic services realized by a fixed constellation of VMs, but where the number of VM instances and their size varies in line with demand. A different perspective is taken by [67], who outline a solution that focuses on the resiliency of VMs comprising a multi tiered application, by seeking

to spread the VMs across as many servers as possible. [68] addressed a similar problem: that of making VMs resilient to k server failures, so that if up to k such failures occur in the cluster there is a guarantee that this VM is relocated to another server without relocating other VMs. [69] uses an Integer Linear Programming approach (ILP), and takes in the order of minutes to decide the placement of 1024 VMs in the data center of 16 servers.

Network aware virtual machine placement

VMs access a server's network interface to communicate with other application and system components. Hypervisors treat a servers network interfaces as unmanaged resources. They do not provide a guaranteed allocation to individual VMs, relying on the fact that VMs are unlikely to simultaneously maximize their use of their nominally assigned bandwidth [70]. However, this means that there is potential for VMs to affect each others' performance due to contention for network interface resources [71]. Recent works have addressed how to mitigate this risk by taking this contention into account when placing VMs. [72] applies the Stochastic Bin Packing problem formulated by [73] for bandwidth allocation to bursty network traffic flows to VM placement. They treat a VMs demand as a random variable and propose an online VM placement algorithm where the number of PMs required to place a set of VMs on is within the optimum.

There is this intuition fact, it is beneficial to place VMs that interact with each other in closer proximity in the data center network topology. [69] formulates heuristics that minimize the aggregate traffic rates perceived by switches in the data center network by placing VMs with large mutual bandwidth closer to each other. Evaluation results using production traces show a significant performance improvement. Similar approaches to minimizing data center traffic have been proposed [64, 74, 75]. [76] addresses network-aware VM placement; their Min Cut Ratio-aware VM Placement (MCRVMP) formulation and heuristics solution incorporate constraints of complex network topologies and dynamic routing schemes. They also take into account the time varying nature of traffic demands, so that the placements found have sufficient spare capacity across the network to absorb unpredicted traffic bursts. [77] addresses data center traffic minimization via an online algorithm that exploits multi-path routing capabilities and live migration of VMs. They argue that it is important to not only optimize the placement of VMs but also jointly optimize the routing between servers hosting interacting VMs. [78] extends the network-aware VM placement problem to geo-distributed clouds, proposing algorithms that select which data center to place a VM in, and, within that data center, which server to place it on. [79] approaches network-aware placement from the perspective of a client, who has leased a group of VMs with set inter-VM bandwidth limits. Their vBundle system facilitates the exchange of bandwidth allocations between VMs in a group as traffic patterns evolve. [80] focuses on network interface of machines as the network resource to optimize with the

server resources. They assume that the network interconnecting the machines has full bisection bandwidth, so that considerations of bandwidth of the links are put aside.

Dynamic VM placement

Following initial placement, VMs can be re-scaled vertically (allocation of additional server and network resources), according to the dynamic changes in the demand for the applications that they host [81, 82]. However, vertical scaling is limited by capacity of the server, because it cannot increase the resources more than the maximum capacity. Therefore, a more powerful method of scaling is, live migration of VMs [32], whereby running VMs are paused, serialized and transferred to a different servers with more resources, where they are once again scheduled for execution. Hence, the dynamic VM placement works focus on finding new placements for the VMs: servers that satisfy the new demand, so that the existing VMs can be migrated to the new servers.

Many studies have proposed optimizations for VM migration process [83–86]. [87] propose a first-fit heuristic that dynamically remaps VMs to servers in a manner that minimizes the number of servers required to support a workload at a specified allowable rate of SLA violations. [82, 88] propose Kingfisher, a set of techniques for VM rescaling, replication and live migration. They formulate the problem as a set of ILP equations and propose a greedy heuristic solution. [89] presents a technique that minimizes the number of VM migrations and migrates VMs with strong utilization correlation to different servers in order to minimize the risk of future server overload. [90] proposes a novel formulation of the VM live migration problem: they cast it as a Stable Marriages problem [91] and argue that, it is more appropriate than optimization based formulations that dictate "an arbitrary way of resolving the conflicts of interest between different stakeholders in order to achieve a global notion of performance optimality". They propose a polynomial time algorithm that generates an egalitarian stable matching between VMs and servers.

From a more practical point of view, researchers have observed that in public clouds the same VM instance types can exhibit significantly different performance levels depending on characteristics (processor, I/O speed etc.) of the hosting server [92, 93]. Therefore, the heterogeneity should be taken into account by VM migration optimization approaches. [94] addresses the use of VM migration in the context of cloud bursting for enterprise applications, specifically on the decision regarding which VMs to migrate to an external cloud environment and when. [95] also addresses cloud bursting; they analyse a more generalised task scheduling problem formulated as a Markov Decision Process and show that a threshold based scheduling policy is optimal. [96] addresses VM migration and VM consolidation, but from the perspective of how VM migration and VM consolidation are used to satisfy dynamically changing management objectives that

are enforced via policies. Their study addresses how to change between the conflicting goals of minimizing SLA violations (by spreading VMs across more servers) and minimizing power consumption (by consolidating VMs on fewer servers).

2.2.3 Cloud resources management for VNF placement

As opposed to the traditional VM placement in the cloud, optimizing the placement of VNFs is network oriented as they are middle points in the network. Provisioning requests of VNFs involves service chains of VNFs rather than individual VNFs. The placement of a service chain requires, first, the allocation of computing resources (cpu, memory, etc.)for the VNFs, which is basically selecting servers to place the VNFs. Then the allocation of network resources (bandwidth) which is basically selecting paths to route traffic flow from one VNF to next VNF in the chain, within the cloud.

VNF placement and provisioning

The VNF Placement is a NP-hard optimization problem, as it can be seen as a generalization of the VM allocation problem which is NP-hard [69]. Execution time is crucial in VNF placement because VNFs deals with dynamic on-line environments where arrival time of service requests is not known in advance. Therefore, to avoid delay when solving the VNF Placement problem, the execution time of the proposed algorithms should be minimized. Hence, instead of optimal solutions which take long time to find, heuristic-based solutions are preferred because they try to find a good solution while keeping execution time low. Similar to the VM placement problem, the early work on VNF placement considered to be an off-line problem: a static resource allocation approach that focus only on the initial placement of VNFs in the cloud [11].

[97] provides an ILP formulation with implementation in CPLEX and a dynamic programming based heuristic to solve larger instances of the VNF placement problem. Their model can be used to determine the optimal number of VNFs and to place them at the optimal locations to optimize network operational cost and resource utilization. [98] proposes a set of heuristic algorithms: a greedy algorithm and a tabu search based local search to solve both the VNF placement VNF scheduling in a coordinated way. [99] proposes a heuristic based approach to minimize the number of deployed VNF instances. Their heuristic approach dynamically and efficiently guides the search for solutions performed by ILP solvers in order to quickly arrive at high quality, feasible ones. The authors of [100] solve the problem of VNF service chain placement using a mixed ILP and give insights into trade offs between legacy and NFV based Traffic Engineering (traffic engineering goal alone and traffic engineering goal combined with NFV infrastructure cost minimization goal). [101] presents an analytic

model for the VNFs forwarding graph, with the aim to optimize the execution time of the deployed network services. [102] presents and evaluates an energy aware Game Theory based solution for resource allocation of VNFs within NFV environments. Authors consider each VNF as a player of the problem that competes for the physical network node capacity pool, seeking the minimization of individual cost functions. [103] formulates the problem of composing, computing and networking VNFs to select those nodes along the path that minimizes the overall latency (i.e. network and processing latency). The optimization problem is formulated as a Resource Constrained Shortest Path problem on an auxiliary layered graph accordingly defined. [104] formulates the VNFs composition problem as a non-linear optimization model to accurately capture the congestion of physical resources. They also propose a dynamic pricing strategy of network resources, proving that the resulting system achieves a stable equilibrium in a completely distributed fashion, even when all virtual operators independently select their best network configuration. [105] provides a mixed ILP formulation to determine the placement of services and routing of the flows, and heuristics to provide the opportunity to perform the placement incrementally without imposing a significant penalty.

The main contribution of [106] is a deterministic competitive on-line algorithm called ACE (Admission control and Chain Embedding). This work attends to the problem of admitting and embedding a maximum number of service chains, i.e., a maximum number of source-destination pairs which are routed via a sequence of to-be-allocated, capacitate NFs. [107] presents a joint design which optimally deploys NFs and allocates physical resources satisfying end-to-end requests with generated routes. They propose a mixed ILP approach, which simultaneously identifies physical nodes to be deployed with NFs and generates routes sharing common physical resources realizing end-to-end requests. [108] proposes two algorithms to map service function chains to the network infrastructure while allowing possible decomposition of NFs. The first algorithm is based on ILP which minimizes the cost of the mapping based on the network service chains requirements and infrastructure capabilities. The second one is a heuristic algorithm to solve the scalability issue of the ILP formulation. It targets to minimize the mapping cost by making a reasonable selection of the NFs decompositions. [109] formulates the virtual deep packet inspection placement problem as a cost minimization problem. The cost captures the different objectives the operator is pursuing (cost for the traffic management or cyber security targets such as the number of inspected flows and operational cost constraints such as license fees, network efficiency or power consumption). [14] proposes rounding-based heuristics to solve VNFs forward graph embedding, that tries to minimize ISP's Operating Expenditures (OPEX). They provide bi-criteria results, approximating the objective function by a constant factor while violating the size constraint by a constant factor as well. [110] proposes a mixed ILP formulation to find its optimal solution in the VNF placement context, and then followed by a two-player pure-strategy game model which captures the competi-

tion on physical resources between network function instance allocation and routing.

Dynamic VNF placement

Similar to the VM placement problem, following initial placement, VNFs can be re-scaled vertically (allocation of additional computing and network resources), to offer infrastructure resources for client's traffic on an on-demand basis [111]. The work on scaling of VNFs is limited and same as for the initial placement, existing solutions for scaling also look for approximations.

[111] introduces the elastic virtual network function placement problem and presents a model for minimizing operational costs in providing VNF service. In the proposed model, the elasticity overhead and the trade-off between bandwidth and host resource consumption are considered together. They propose a heuristic based solution called Simple Lazy Facility Location (SLFL) that optimizes the placement of VNF instances in response to on-demand workload. [15] works with VNF chains and solves the initial placement problem by iterating over initial input chains. Scaling is achieved by duplicating full instances of VNF chains and the optimization is very dependent on the data center architecture. [16] pushes the locality to the limit and tries to deal with the optimization of one VNF at a time, independent of the location in the chain of VNFs. They take as input a set of individual VNFs demands and to achieve scaling, they might migrate, duplicate or remove VNF instances. [112] presents an on-line heuristic algorithm to allocate VNF instances effectively, achieving minimum overall bandwidth occupancy, VM usage and migration overhead while accepting as many requests as possible. Their algorithm provide two alternatives for VNF scaling: (1) an incremental deployment, where the extra part of traffic is treated as another service chain and (2) an unified deployment, where the service is treated entirety and reallocate all the traffic by invoking the heuristics again.

2.2.4 Cloud load balancing with SDN

The initial research on load balancing in cloud infrastructure, was about balancing the work load across available cloud resources. To alleviate heavy-traffic network flux and to reduce the risk that a single server (or VM) will be overloaded, CSPs adopted dynamic load balancing mechanisms and distributed the traffic across multiple servers (or VMs). Current dynamic traffic load balancing methods tend to be flow-level based, where the traffic distribution is decided according to the traffic flows [29]. A traffic flow is a uni-directional sequence of packets sent from a source to a destination, where one flow is distinguished from another by its header contents: 5 tuple (source and destination ip addresses, source and destination port addresses, and the protocol) [29].

SDN based load balancing has become popular among CSPs, where the SDN controller acts as the load balancer. There are two main reasons for the popularity of SDN controller based load balancers. First, with the SDN concepts, the whole network is managed with a centralized controller, which has the full view of the network, including up to date traffic statistics. Therefore, a SDN controller can use the traffic statistics and decide the traffic distribution efficiently [113]. Second, SDN allows easy re-configuration of the network, therefore the network can be easily configure to route the traffic according to the load balancing decisions [18].

[113] is the first work on SDN controller based load balancers. The authors present Plug-n-Serve, a module residing within an OpenFlow controller that is capable of performing load balancing over unstructured networks, aiming to minimize average response time of HTTP servers. Plug-n-Serve load balance HTTP requests by gathering metrics about CPU consumption and network congestion on the network links, which makes its load balancing algorithm to select the appropriate server to direct requests, while controlling the path taken by packets on the network. [114] proposed an OpenFlow based load balancing approach that aims to pro-actively load balance traffic from clients to servers by slicing the IP address space into trees that isolates a set of clients to a set of servers. The work uses the concept of server weighting, which defines a fixed portion of the clients to a server on the network. To do so, it is proposed the extensive use of wild-cards, which may reduce forwarding performance and create management issues, as shown in [115]. Furthermore,the proposed solution requires that in certain conditions (network topology changes or server weight updates); a part of the network traffic passes through the controller, which could cause serious scalability problems that may lead the network controller to collapse. [116] proposed an OpenFlow based load balancer for Fat-Tree networks that supports multi-path forwarding.Their proposal aims to recursively find the current best path from a source to a destination, load balancing the network by enabling the use of alternate paths at runtime, minimizing network congestion.Their algorithm works only on networks that operate on the Fat-Tree topology and use network metrics for choosing the optimal path. The work from [117] proposes an architecture to enable in-network load balancing of multiple services using OpenFlow. Their proposal relies on a set of SDN controllers on top of a Flow Visor instance [118], where each controller is responsible for load balancing the traffic of a specific service. The authors have focused on the architecture, so there is no information about particular service implementation, while the experiment does not fit real-world scenarios. The idea of using a set of controllers to handle exact services might be interested in some specific cases, but has the drawback of not permitting multiple services to be handled by a single controller, which is the most common situation of SDN deployment.

2.2.5 Load balancing between VNF instances

Traditionally CSPs have been adopting dynamic load balancing approaches to reduce the risk of a single server (or VM) being overloaded. They try to distributed the traffic load across multiple servers (or VMs). In the similar manner, there are situations where CSPs should adopt dynamic load balancing mechanisms for VNFs, to balance the load among multiple VNF instances. For an example, when using horizontal scaling as the scaling approach, VNF instances are added/removed to satisfy the scaling requirement. However, when new VNF instances are added/removed, the traffic has to be balanced among the remaining VNF instances dynamically.

As mentioned in previous sub-section, most of the current load balancing algorithms for cloud are flow-level based, where a traffic flow is a uni-directional sequence of packets sent from a source to a destination. However, with VNFs, the load balancing approaches should look beyond the uni-directional flows and take care of the sessions: which is bi-directional traffic flows between two nodes. This is because, unlike layer 3 forwarding, many VNFs such as firewall, proxy, and VPN perform stateful packet processing: session based packet processing [29]. Therefore, these VNFs require affinity, where traffic for a given flow must reach the instance that holds that flow's state [30]. In such cases, splitting traffic to balance the load, requires extra measures to preserve affinity.

Most of the existing solutions that maintain affinity depend on state migration techniques: moving the relevant state from one instance to another. Frameworks like Split/Merge [119] and OpenNF [30] facilitate fine-grained transfers of internal NF state to support fast and safe reallocation of flows across NF instances. In these frameworks, a scenario-specific control application decides: (1) when internal NF state should be moved; i.e., after a new NF instance is launched; (2) what subset of state should be moved; this is usually defined in terms of a flow space *fspace*, i.e., all state pertaining to flows originating from a particular subnet; and (3) between which pair of NF instances the transfer should occur. A central controller then asks the source NF instance to export the state pertaining to flows in *fspace*. This state is provided to and imported by the target NF instance. In Split/Merge, the state is transferred directly from source NF instance to target NF instance, while in OpenNF the state passes through the controller. Finally, the controller updates the forwarding state in a SDN switch, such that traffic in *fspace* is now forwarded to target NF instance. However, the drawback of both of these approaches: Split/Merge and OpenNF, is NFs must be modified prior to run time to support such export/import operations. These systems require that NFs vendors adopt a new programming model or add a non-trivial amount of code to existing NF implementations. More ever, they require large number of rule sets in hardware switches.

E2 [120] presents a VNFs scheduling framework that supports affinity based NF placement while trying to minimize the traffic across switches

as well as deploying dynamic scaling of NF instances. The authors introduce a high-performance and flexible data plane where Click [41] modules (i.e., classifier and TCP re-constructor, etc.) are embedded to accelerate the packet processing. The VNFs in E2 need to explicitly call the API exported by the data plane to achieve the benefit from such data paths. E2 uses a novel migration avoidance strategy in which the hardware and software switch act in concert to maintain affinity. Their strategy does not require state migration, and it is designed to minimize the number of flow table entries used on the hardware switch to pass traffic to NF instances. They consider a NF with a corresponding range filter $[X, Y) -> A$. When A is split into A and A', the range can be partitioned into $[X, M) -> A$ and $[M, Y) -> A'$. Additionally, to maintain affinity, any existing flows in [M,Y) must also be sent to A. E2 uses a three phase strategy. First, upon splitting, the range filter $[X, Y)$ in the hardware switch is left unchanged, and the new filters (two new ranges plus exceptions) are installed in the E2D of the server that hosts A (i.e., as software defined filters). Second, as existing flows gradually terminate, the corresponding exception rules are removed. Finally, once the number of remaining exception rules falls below some threshold, the new ranges and remaining exceptions are pushed to the switch. However, E2 migration scheme works only for NFs that operate on state that is easily partitioned or replicated across NF instances; e.g., per-flow counters, per-flow state machines and forwarding tables. They do not address the consistency issues that arise when global or aggregate state is spread across multiple NF instances.

2.3 Remarks

So far, in the Background chapter we have discussed the existing work related to the research work in this thesis in depth and how the work in this thesis is motivated by the existing work. In this section we will summarize how the work in this thesis is different from the existing work.

As stated in the Introduction section, the first set of contributions of this thesis, are to the context of resource allocation for VNFs. As explained in early sections, existing work on cloud resource allocation for VMs are not suitable for cloud resource allocation for VNFs.

First, the client request for VMs comes with the specifications of the VM (i.e, CPU and memory needed), therefore, the resource allocation on the cloud was focused on provisioning of VMs. On the other hand, the client request for VNFs comes with the specifications of the expected traffic load, and therefore, based on the traffic load, the CSP has to decide the specifications of the VM that needs to implement the VNF, and after that deploy the VNFs according to the placement decisions.

Second, Optimizing the placement of VMs in a cloud tend to be node-centric as VMs are end-points. Optimizing the placement of VNFs is, in contrast, network-centric – their provisioning normally involves service

chains of VNFs rather than individual VNFs. The placement of a service chain requires allocation of server resources (for the VNFs), as well as allocation of network resources (paths) to route traffic flow from one VNF to next VNF in the chain, within the cloud. Most of the existing work on placement of VNFs, try to optimize either host or bandwidth resource, but do not provide an integrated view of computation, storage and networks optimization [21].

Third, most of the research work on resource allocation focus only on a part of the problem: resource allocation for the initial placement of VNFs, but do not consider the resource re-allocation for the scaling requirements of existing VNFs. Therefore, despite some initial efforts [15, 16, 22] the resource re-allocation for scaling VNFs presents still many open challenges. For example, one of the challenges is how to achieve scaling: i.e., whether to use horizontal, vertical or migration scaling technologies. A second challenge is how to resolve potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimize usage of server and network resources [17].

In this thesis, we focused on resource allocation for two situations: the resource allocation for the initial placement of VNFs and (2) the resource re-allocation for the scaling requirements of existing VNFs. Once the client request for a set of VNFs has been accepted, the resource allocation for the initial placement of these VNFs can be done in the order of minutes, and then the new VNFs can be deployed accordingly. But the resource re-allocation for the scaling requirements of existing VNFs happens during the middle of the operations, where the already deployed VNFs are processing traffic. Therefore, the resource re-allocation for the scaling requirements of existing VNFs is a time critical on-line problem. The solutions have to be given in the order of milliseconds, so that the disturbances and damages to current operations are minimal. Therefore, we specifically explored the resource re-allocation for the scaling requirements of existing VNFs problem in depth, as it is an interesting on-line resource allocation problem.

The second set of contributions of this thesis, are to the context of VNFs scaling methods. Again, as explained in early sections, the existing research work on scaling of VNFs is very limited, most of them focus only on the complexities of the scaling technologies. They do not consider the resource allocation or optimization aspects of these scaling technologies. Therefore, we explored different scaling approaches and the optimization perspectives: vertical, migration and horizontal.

The third set of contributions of this thesis, are to the context of dynamic load balancing mechanisms. With VNFs, the load balancing approaches should look beyond the uni-directional flows and take care of the sessions: which is bi-directional traffic flows between two nodes. This is because,

unlike layer 3 forwarding, many VNFs such as firewall, proxy, and VPN perform stateful packet processing: session based packet processing [29]. Therefore, these VNFs require affinity, where traffic for a given flow must reach the instance that holds that flow's state [30]. In such cases, splitting traffic to balance the load, requires extra measures to preserve affinity. Existing solutions that maintain affinity mostly depend on state migration techniques: moving the relevant state from one instance to another. However, these systems require that NF vendors adopt a new programming model or add a non-trivial amount of code to existing NF implementations. Therefore, we argue that it is more appropriate to design the load balancing algorithm in a way that the algorithm itself maintain affinity and handle states and sessions, so that the requirement of state migration is avoided. We proposed a load balancing algorithm that controls sessions, using consistent hashing techniques.

Chapter 3

Network Function Center (NFC)

> "A man may die, nations may rise and fall, but an idea lives on."
>
> *John F. Kennedy*

With the emerging technologies of NFV, CSPs are able to offer VNFs as a service to enterprises. However, for a CSP to offer such services, many research questions still need to be addressed. When an enterprise outsources its NFs to a CSP, the CSP is responsible for deciding:

- Where the initial VNFs should be instantiated
- What, when and where additional VNFs should be instantiated to satisfy the dynamic traffic demands (scaling)
- How to update the network configurations with minimum latency, packet loss, and the impact on network performances

We have built an experimental platform, called Network Function Center (NFC), to study management issues related to the combined management of VNFs. The architecture and the complexity of our experimental NFC platform are assumed to be simpler than those defined by existing standardization bodies (e.g., ETSI, IETF) [1]. This simplification allowed us to focus on specific research aspects and conduct experiments.

In this chapter, we introduce our experimental platform: the NFC in following aspects:

1. Description of the overall functionality of the NFC: Section 3.1

2. Description of the NFC architecture, and an overview of the NFC Management System: Section 3.2 and 3.3
3. Description of the data sets and use cases used to explore different aspects of VNFs Management: Section 3.4

3.1 NFC Functionality

As shown in Figure 3.1, the main concept of NFC is a cloud infrastructure and a service provider that offerers VNFs as a service to enterprises. Therefore, the owner of the cloud infrastructure can be considered as the **cloud service provider of the NFC**. The enterprises who are outsourcing their network based functions to the NFC can be considered as the **clients of the NFC**.

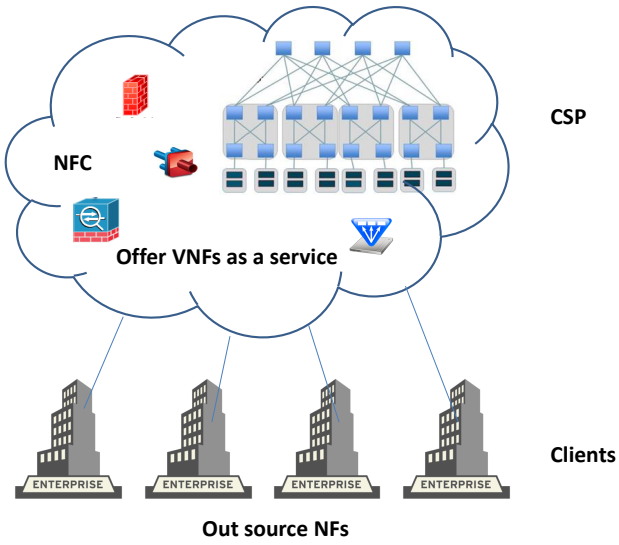

Figure 3.1: NFC business model

Contrary to traditional NFs that are hardware based middle-boxes, deployed at specific locations in the network, NFC assumes a NF to be implemented by a VM that can be deployed in any server in the CSP network. As shown in Figure 3.2, to receive services from NFC, a client needs to provide three specifications to NFC:

1. Types of NFs required and interconnectivity between these NFs (i.e, redundant eliminator - firewall - proxy)
2. Expected traffic load to be processed by these NFs (i.e, 200 GB)
3. Ingress and egress locations (i.e, internet to web server)

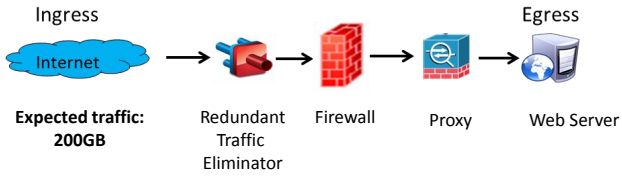

Figure 3.2: Example of a client request

The abstractions used to solicit this information from a client will depend on the NFs. The specification could be complex as a high-level description of a virtualized network where each network's endpoint is connected to predefined Virtual Networks [121]; or a simple specification as a set of NFs chains through which different classes of traffic must go through [29] (e.g., all traffic from 10.0.0.0/24 must traverse IDS-Firewall-Proxy). The specifications can be (automatically or semi-automatically) translated into a collection of Directed Acyclic Graphs (DAG) connecting sources to destinations of data flows in which the intermediate nodes in a graph path represent NFs that must be applied to the traffic flow going through the path. The first and the last node in a path are the source and the destination of the flow and may or may not be hosted inside the NFC. Each DAG must be accompanied with capacity information for each node (i.e., function capacity requirements) and information about traffic characteristics that will go through the different paths (e.g., traffic class, expected throughput). As an illustration, we assume a NFC that provides VNFs represented as chains of services.

As shown in Table 1 of Figure 3.3, the client request comes to the NFC in the form of:

- Policy (chain of required NFs)
- Ingress and egress locations of client's traffic flow
- Expected volume of the traffic flow

Table 1

Client	Traffic Flow	Policy	Expected Traffic
Client1	192.168.0.0/24 - *, HTTP	Load Balancer – Firewall - Proxy	200 GB
Client2	172.0.0.0/24-178.0.0.0/24, *	IDS – Proxy	300 GB

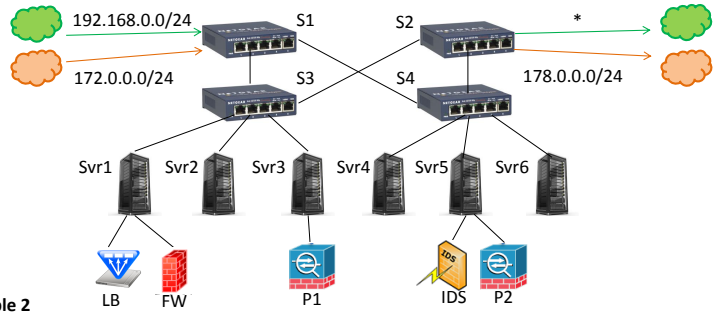

Table 2

Policy	Physical Sequence
Load Balancer – Firewall - Proxy	S1-S3-Svr1-LB-Svr1-FW-Svr1-S3-Svr3-P1-Svr3-S3-S2
IDS - Proxy	S1-S4-Svr5-IDS-Svr5-P2-Svr5-S4-S2

Figure 3.3: Network Function Center

Once the client request is accepted by the NFC, the client's traffic is redirected to the NFC to traverse the NFs. The NFC must guarantee that the client's traffic traverse all the NFs in the correct order. In addition, the NFC is expected to increase/decrease the number of VNFs instances and paths for the traffic flow according to the application's dynamic needs and agreements with the client.

3.2 NFC Architecture

The overall architecture of a NFC consists of two main components: a physical infrastructure, and a management system for the infrastructure.

The physical infrastructure comprises a network and a server infrastructure. The network infrastructure provides connectivity for all communications occurring in the NFC and between the NFC and its users. The server infrastructure hosts all NFs. Servers in the NFC are used to deploy the virtual machines (VMs) where NFs run. A NF is implemented as a software on a VM. Similar to recent work in data centers [18], to make more efficient utilisation of the NFC resources, we allow both computing resources and network configurations to be managed concurrently and assumes a Software-Defined/OpenFlow infrastructure [122] to easily reconfigure the physical network.

Figure 3.3 represents a snapshot of a NFC. It depicts the placement of VNFs to implement the two policy chains in Table 1 of the Figure. Table 2 of the Figure shows the physical sequences of switches and VNFs

the client's traffic will go through. **Client 1** wants his HTTP traffic flow coming from 192.168.0.0/24 to any destination to go through the policy chain of Load Balancer-Firewall-Proxy VNFs. To grant his request a virtual Load Balancer service and a virtual firewall service are implemented on two VMs at **Server 1** and a virtual proxy service is implemented on a VM at **Server 3**. **Client 2** wants any type of his traffic flow coming from 170.0.0.0/24 to destination 178.0.0.0/24 to go through the policy chain of IDS-Proxy VNFs. To grant his request a virtual IDS service and a proxy service are implemented on two VMs at **Server 5**.

The network architecture depicted in the figure represents a tree-like architecture typical of datacenters, but our NFC architecture does not assume any particular network structure. As network element reconfiguration becomes a more active part of the management, these standard architectures may change for the benefit of the management. The management system for the infrastructure is described in more detail in the next section.

3.3 NFC Management System

The goal of the NFC Management system is to automate arrangement, coordination and management of NFC components to maximize on-demand client requests whilst guaranteeing QoS. Figure 3.4 gives an overview of the proposed NFC Management System. The NFC Management System requires three high-level inputs:

1. Client Requirements given as a set of policies,
2. Topology data and Traffic
3. Resource constraints

Each client's requirement is annotated with the traffic flow's ingress and egress locations, flow properties (source destination IP addresses, source destination ports), expected volume of traffic traversing each NF in the policy. Topology data and traffic describes the placement of current VNFs, paths between servers, links between switches, available capacities of links, switches and servers. Resource constraints specify (1) server and switch resources (CPU, memory, TCAM sizes, etc.) and (2) bandwidth capacity for each link in the topology.

Process of NFC Management System

The process NFC Management System is built around five key modules: (1) Resource Manager, (2) Topology Manager, (3) Flow Manager, (4) Elasticity Manager and (5) Rules Generator.

Once a new client request is submitted, Resource Manager takes decisions on the placement of VNFs and paths for the client's traffic to follow inside the NFC, based on the resource optimization goals of the NFC. The

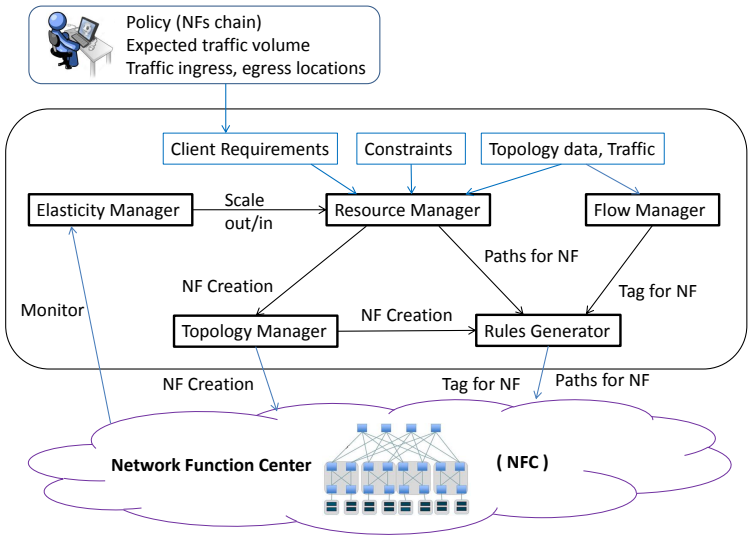

Figure 3.4: NFC Management System

Resource Manager is also called by the Elasticity Manager. The Elasticity Manager monitors the resources utilization. According to parameters such as network traffic, applications' requirements and agreements with clients, the Elasticity Manager takes decisions on when to increase/decrease the instances of VNFs and paths for the traffic flows. Once the Elasticity Manager makes its decision, the information is passed to the Resource Manager which then determines the possible changes to the placement of the VNFs instances and paths, based on the resource optimization goals of the NFC. The Topology Manager, Flow Manager, and Rules Generator configure the network according to decisions taken by the Resource Manager and Elasticity Manager. The five key modules are described in the following sub-sections.

3.3.1 Resource Manager

For each new policy request, the Resource Manager decides the acceptance or rejection, based on the current network traffic, topology data, constraints, client requirements and the resource optimization goals of the NFC. The server resources requirement for each VNF to serve the expected traffic rate, can be derived using the expected traffic rate and the resource profile of the VNF type. A resource profile of a VNF includes:

- Processing capacity of the VNF
- Required minimum number of CPU cores

- Bandwidth demand transformation by the VNF (compress or amplify the traffic)

The bandwidth demand transformations are associated with traffic-scaling NFs. For an example, certain NFs, such as Redundant eliminators and Caches compress traffic, while other NFs such as packet multiplication and encryptions amplify the traffic.

For each VNF in the requested policy, considering the optimization goals of the NFC, the Resource Manager identifies:

1. Either a non-used VM that can be re-used to run the required NF or a server where a new VM can be created to run the required NF.
2. A path for traffic flow between every two NFs directly connected in the policy, based on expected volume of traffic specified in client requirement and the available bandwidth of links in the NFC.

If the Resource Manager is able to find server and network resources for all the VNFs in the policy request, then the policy request is accepted.

With respect to the resource allocation decisions, there are many linear and non-linear factors that might affect the NFC: number of servers used, links used, links congestion, traffic lost, delay, cost of VNFs software license and number of policies accepted etc. Therefore, for the new policy requests provisioning, the resource allocation decisions can be taken with different optimizing goals, for instance: (1) minimize the server and network resources usage, (2) minimize the overall operational cost and (3) minimize the number of VNF instances used and (4) maximize the number of policies accepted etc.

In addition to the new policy requests provisioning, responding to requests from the Elasticity Manager to scale resources, the Resource Manager decides a new set of VNFs assignments and paths for new traffic demands of existing policy requests, based on the resource optimization goals of the NFC. In NFV context, scaling of a VNF instance can be done as: (1) vertical, (2) migration and (3) horizontal scaling. In a situation of dynamic resource allocation for scaling, the transition from one configuration of VNFs and flow assignment to another could create transient congestion which could degrade applications performances. Therefore, it introduces potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimize usage of server and network resources [17].

The combination of all these factors has to be considered by the Resource Manager to decide resource allocation. A popular technique that has been used for VM allocation [69] and network management [29] is to model resource allocation as a mixed Integer Linear Programming (ILP) optimization problem. There is the intrinsic constraint that ILP optimization

is a NP-complete problem, and even when solutions are obtained for special classes, it might be too slow for continued adjustments of the system configuration causing inappropriate hysteresis in the reaction. Therefore, we explored the more realistic approach of looking for a good feasible configuration fast. We have explored approximations techniques, specifically we have modelled the problem as finding the *best fitted* solution according to an Iterated Local Search (ILS) model and a Genetic Algorithmic (GA) model of the problem. Details of the implementations can be found in Chapter 4.2 and Chapter 4.3.

3.3.2 Topology Manager

The Topology Manager module is responsible for maintaining up to date state of the physical infrastructure of the NFC. It keeps an inventory of all functions running or dormant in the system. It knows about all physical paths between servers and paths used by all the traffic flows. It also maintains information about current traffic and service demands. It will be the source of data for any analytic needed to be done about the NFC. It is also in charge of the instantiation or re-use of the necessary VNFs as well as the provisioning needed according to the instructions given by the Resource Manager. Creation of VNFs may include deploying a VM in a server, installing the necessary software and starting the necessary processes required by the VNFs.

3.3.3 Flow Manager

The aim of the NFC is to provide flexibility in regard to VNFs placement which can be placed anywhere in the network. However, this complicates the network configuration, because the sequence of switches in the physical network that the traffic from a client has to follow, may have loops (the same traffic flow may visit the same server for two different functions more than once). If so, the combination of original source and destination information of a packet, is not sufficient to identify the VNFs this packet has gone through so far. Making the situation more complicated, many VNFs modify packet headers. Therefore, the original source and destination information of a packet can be changed during the flow [29]. Hence, it is essential for the Flow Manager module to find mechanisms to come up with unique identification for the state of a packet in a flow path. This identification can be done through tunnelling mechanism [121] or can be added as a tag into headers of packets going through the VNF.

In the most general case, this traffic identification must be agnostic to the physical topology of the NFC, so that changes in the topology do not require changes to the Flow Manager. However, special Flow Managers could be designed to take advantage of particular classes of topologies. If the topology has the tree structure similar to current data centers architectures, identification of flows can be specialized knowing that traffic will

go up and down the hierarchy. This can reduce the number of identifiers needed to differentiate the different paths.

3.3.4 Elasticity Manager

One of the main objectives of NFC is to support scaling VNFs according to the network traffic and dynamic needs of applications. Therefore, the Elasticity Manager monitors the network and servers to determine when to scale the resource allocation to meet the traffic demands according to the SLAs and QoS agreements. Finding the exact VNF(s) or path(s) which are causing the bottleneck is essential because of the costs involved in running new VNFs as well as the impact reallocating functions and flow paths may cause in service quality and traffic lost due to switching delays. Also, it is important to decide the right type an amount of resources to increase/decrease to achieve the demand and avoid the potential for some kind of thrashing phenomenon.

The most basic method to scale is to monitor system-level metrics (server and links utilization) and determining whether to scale based on a threshold. However, threshold-based algorithms do not capture the complex interaction among multiple resource parameters (server and links) and the potential diversity of traffic types. Determining the right set of thresholds for them to simultaneously achieve the right SLA and QoS for each type of traffic would be difficult. Often the thresholds are set based on ad-hoc measurements and experiences. This is an obvious situation where machine learning techniques, in particular reinforcement learning, must be explored to learn the behaviour of the applications and automatically adapt to changes. The learning algorithm can be augmented with heuristics to improve the responsiveness and guide the algorithm itself.

3.3.5 Rules Generator

The Rules Generator module generates data plane configuration for the switches to route traffic through the appropriate sequence of VNFs from their source to destination according to client requirements. The Rules Generator is the module that directly takes advantage of SDN and OpenFlow, therefore, it is as an extension to the network operating system. Because the NFC controls the switches, routing inside the NFC network is done entirely using OpenFlow VLAN and MPLS tags adapting the ideas from [29]. This is done to avoid all the problems that modifications of packet headers by middle boxes can cause [123]. Rules Generator uses the identification tag issued by the Flow Manager to infers mappings between the incoming and outgoing traffic flows of a network service and to identify the traffic flow that a packet belongs to. Tags are added at ingress point before the traffic has been gone through any VNF, thus letting the traffic be identified by the Source and Destination headers and they are removed at the egress point.

Furthermore, when updating the network configuration as a result of scaling needs, the Rules Generator follows the per-flow consistency introduced in [31] to avoid any inconsistencies in transient traffic and reduce traffic lost. The update mechanism works by stamping every incoming packet with a version number and modifying every configuration so that it only processes packets with a set version number. To change from one configuration to next, it first populates the switches in the middle of the network with new configurations guarded by the next version number. Once that is completed, it enables the new configurations by installing rules at the perimeter of the network that stamp packets with the next version number. This method makes network updates faster and cheaper, by limiting the number of rules or switches affected.

3.4 Datasets and use-cases

While exploring management aspects of VNFs, it was important to have more realistic use cases and datasets that can be used for the design and evaluation of the algorithms, with respect to the cloud management. Therefore, we needed data on:

1. Potential NFs chains (policies)
2. Traffic flows passing through these NFs chains
3. Different data center architectures for NFC

However, there are no publicly available real data sets on VNF chains and traffic that pass through VNF chains. Therefore, we combined a data set from a study about physical middle-boxes in enterprises [6] to generate our policies and a data set from a study about HTTP traffic [27] in the Internet to generate our traffic. Since in this thesis we did not explore the problem of monitoring the resources and determining when to scale the VNFs according to the dynamic traffic changes, we used the HTTP traffic in [27] and replicated the traffic changes over a day with relevant scaling triggers. We made some assumptions to derive the required data. We developed four programs to model the gathered data and generated the required data. All gathered data and data modelling programs are publicly available at [124].

In this Section we describe the data and use-cases generating process.

Policy requests

When generating policy requests for the NFC, the main factor to be considered is the type (e.g., small, medium, large size network) of the enterprise/user, that is requesting the policies. Depending on the type of the enterprise/user, the total number of VNFs required, the number of VNFs in a policy and types of the VNFs in the policy can vary. Also, with this information, we can calculate the total number of policies requested by the enterprise/user. The policies used in our experiments are generated based on a study about physical middle-boxes used in enterprise networks [6], which includes figures about types of enterprise networks, number and types of middle-boxes used in them. For our experiments, following statistics given in [6], we have assumed that we are going to provide services for 4 large enterprise networks, each enterprise network having 100 VNFs. According to [6], large scaled enterprises, with 10k-100k hosts can have average: 46 IP Firewalls, 9 Application firewalls, 0 WAN optimizers, 6 Proxies, 3 Gateways, 6 VPNs, 7 Load Balancers, and 23 IDS/IPS with the total of 100 VNFs. The number of VNFs in a policy follows a truncated power-low distribution with exponent 2, minimum 2 and maximum 7. Therefore, for our scenarios, we derive a set of policies for each enterprise, where each set of policies have 100 VNFs and altogether all the policies of four enterprises have 400 VNFs.

Traffic flows

When simulating traffic, we need traffic data where owners (enterprises/users) of the flows can be identified, so that we can differentiate the traffic passing through each policy. The traffic load that each enterprise/user is expecting can vary according to their target applications [125]. We consider web-based applications and for the traffic, we rely on empirical data from previous studies [27]. The data set includes an HTTP traffic breakdown of 30,000 users for a day which is measured at three different vantage points of an Italian ISP over a period of 24 hours. The traffic breakdown reports traffic for every 2 hours. We focused on the traffic statistics of 4 enterprises: Megaupload, LeaseWeb, Level3 and Limelight.

In a data center, traffic changes happen throughout the day and according to the amount of these changes, the VNFs should be scaled to satisfy the current traffic demand. A limitation of the HTTP traffic data we are using is that, information was collected at every two hours. Therefore, the first challenge is interpreting the pattern of traffic change over two hours. According to [2], as shown in Figure 3.5, traffic changes on usual days happen gradually over time. Even at events where traffic will be increased in a huge amount (elephant flows), as shown in Figure 3.6, the change happens gradually over a 15 minutes time period [3]. As such, although sudden traffic changes may occur within few minutes, we have assumed a uniform traffic increase/decrease over the 2 hours time intervals and generated the traffic graph in Figure 3.7. It shows traffic flow for 24 hours in 10MBps units for each enterprise. To reflect scaling requirements of all situations, we spread the increase/decrease of number of VNFs (needed for the full 2 hour traffic change) over 2 hours and increase/decrease the capacity of one VNF at a time.

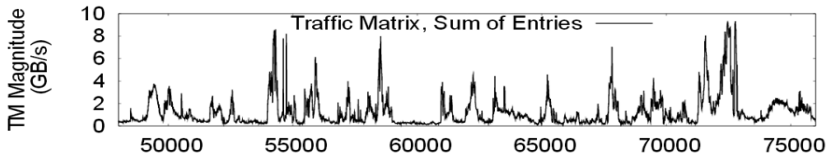

Figure 3.5: Traffic changes of the data center in the magnitude (Time in seconds) [2]

The second challenge is identifying the policies affected by each enterprise traffic change. For each enterprise we have x number of policies generated and each policy has a unique traffic passing through its VNFs. When there is a change in total traffic for that enterprise, it is very unlikely that traffic passing through all policies of that enterprise contributed to the change.

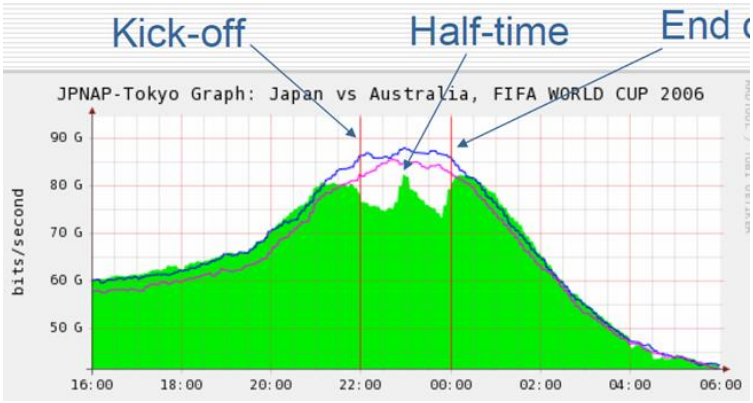

Figure 3.6: Traffic statistics from World Cup 2006 [3]

Therefore, we randomly select a subset of policies from that enterprise, as the policies affected by the traffic change.

Scaling patterns

After selecting the policies affected by each enterprise traffic change, the first challenge is deciding which VNF from each policy, needs to be scaled to satisfy the new traffic demands. An earlier study [10] shows that in general no two VNFs will be simultaneously and equally bottlenecked and scaling only the bottlenecked VNFs in the policy at a time is the best strategy. Hence, assuming the conditions in [10], we randomly select a VNF from each policy as the bottlenecked VNF for which the resource allocation needs to be increase/decrease.

The second challenge is, from the identified VNF instance to scale, how many instances we should add/remove to satisfy the new traffic demand. The server resources requirement for each VNF to serve a specific amount of traffic, can be derived using the inbound traffic rate and the resource profile of each VNF type. A resource profile for each VNF includes: (1) processing capacity of the VNF, (2) required minimum number of CPU cores and (3) bandwidth demand transformation by the VNF. Here, we are making an assumption: the traffic flowing through the VNF instance is proportional to the capacity of the VNF instance and it is the same for all types of VNFs. Therefore, the initial capacity unit requirement of all types of VNFs is assumed to be the same. Also, we assume that the

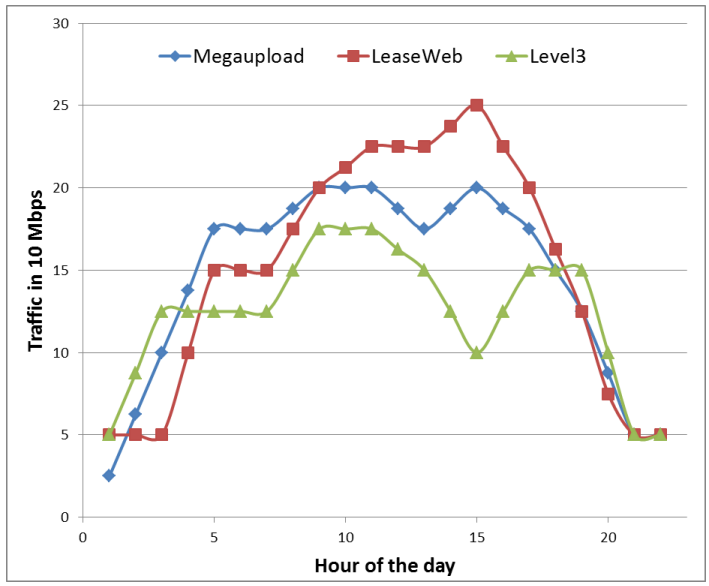

Figure 3.7: Traffic pattern over a full day

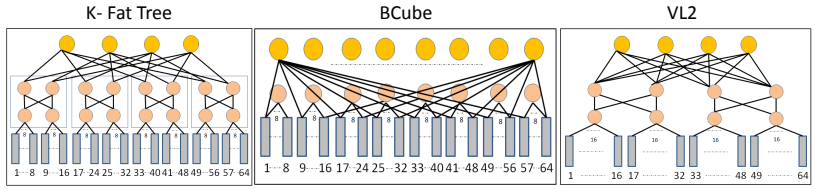

Figure 3.8: Data center architectures used for NFC

VNFs do not compress or amplify the traffic, hence, there is no bandwidth demand transformation by VNFs.

Another study [7] shows that if we add more than one instance at a time, we are usually adding more than what is needed and wasting resources. Therefore, we calculated a traffic change threshold to find how many instances we should add/remove to accommodate traffic change, and as explained earlier, we add/remove one instance at a time. This resulted in 42 significant events over the 24 hours of traffic data. There are two types of events: (1) when the traffic change has reached the threshold, resources have to be reallocated to increase/decrease at least one VNF instance or (2) when the traffic change has not reached the threshold, modify the bandwidth usages of the links of the paths that were effected by the traffic change, to reflect the new traffic amount passing by.

Data center architectures for NFC

We evaluated the performance of the resource allocation algorithm assuming three different data center network architectures for NFC: (1) k fat tree, (2) VL2 and (3) BCube shown in Figure 3.8.

A k-ary fat-tree network [24] has three layers: a core layer, an aggregation layer and a Top-of-Rack (ToR) layer. It consists of $(k/2)^2$ core layer switches and k pods of k switches, half of them aggregation switches and the other half ToR. Each switch in a pod has k ports. The ToR switches are at the bottom of the pod, and the aggregation switches in the middle. In one pod, each ToR switch is connected to every aggregation switch. Each aggregation switch connects to $(k/2)$ switches on the core layer. We have used a 4 fat-tree architecture, which has 20 switches: 4 pods of 4 switches and 4 switches in the core layer. For a NFC with 64 servers, 8 servers are connected to each ToR switch. The network consists of 96 links and 13770 paths connecting all source destination server pairs with maximum number of hops for a path of 6.

The VL2 architecture [26] shares many features with a k-ary fat-tree architecture, but the main difference is the core tier and aggregation tier form a Clos topology: the aggregation switches are connected with core one's by forming a complete bipartite graph. We have used a VL2 architecture with 12 switches. For a NFC with 64 servers, the network consists of 88 links and 33760 paths connecting all source destination server pairs with maximum number of hops for a path of 6. In the BCube architecture [25], servers are considered part of the network infrastructure, i.e., they forward packets on behalf of other servers.

A BCube is a recursively defined structure. At the level 0, a $BCube_0$ consists of n servers that connect together with a n-port switch. A $BCube_k$ consists of n $BCube_{(k-1)}$ connected with n^k n-port switches. We have used a $BCube_1$ architecture where there are 8 $BCube_0$s, each connected to 8 switches in the next level switches and form the $BCube_1$. Each s server of $BCube_0$s is connected to switch s of $BCube_1$. For a NFC with 64 servers, the network consists of 128 links and 7168 paths connecting all source destination server pairs with maximum number of hops for a path of 4.

Chapter 4

Resource allocation for VNFs

From the perspective of a CSP, offering VNFs as a service includes the process of taking decisions on resource allocations, for the new policy requests provisioning as well as scaling of existing policy requests. Optimizing the placement of VNFs is network-centric: their provisioning are middle-points in a network. Provisioning requests of VNFs normally involves service chains of VNFs and the placement of a service chain requires, first, the allocation of computing resources (cpu, memory, etc.) for the VNFs, which is basically selecting servers to place the VNFs. Then the allocation of network resources (bandwidth) which is basically selecting paths to route traffic flow from one VNF to next VNF in the chain, within the cloud [11]. Most of the existing work on placement of VNFs, try to optimize either host or bandwidth resource, but do not provide an integrated view of computation, storage and networks optimization [21].

Furthermore, most of the research work on resource allocation focus only on a part of the problem: resource allocation for the initial placement of VNFs, but do not consider the dynamic resource re-allocation for the scaling requirements of VNFs. The resource re-allocation for the scaling requirements of existing VNFs happens during the middle of the operations, where the already deployed VNFs are processing traffic. Therefore, the resource re-allocation for the scaling requirements of existing VNFs is a time critical on-line problem. The solutions have to be given in the order of milliseconds, so that the disturbances and damages to current operations are minimal. Despite some initial efforts [15, 16] the dynamic resource re-allocation for scaling VNFs presents still many open challenges. For example, one of the challenges is deciding the scaling method: whether to use vertical scaling (allocation/release of host and bandwidth resources to/from a VNF instance) or migration (running VNFs are paused, serialized and transferred to different servers with more resources) or horizon-

tal scaling (installation/removal of VNF instances). A second challenge is how to resolve potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimize usage of server and network resources [17].

In this thesis, we argue that Integer Linear Programming (ILP), traditionally used to optimize VM allocation and network management in cloud data centers [69], is not feasable for online scaling of VNFs in response to traffc changes. This is because solving ILP problems can take hours [13, 14]. Instead, one can find suitable approximation algorithms for the optimization. We have looked at two different approximation techniques: heuristic based approaches that can be used to find a good feasible configuration fast.

First, we investigate our hypothesis, by proposing an Iterated Local Search (ILS) approach to implement the resource allocation algorithms. ILS is one of the most popular single-solution based meta-heuristics due to its simplicity but at the same time powerful approach [23]. However, for the ILS based resource allocation algorithms to be more efficient, they have to be designed with conditions based on the network architecture. In other words, efficiency of ILS procedure is not agnostic to the network architecture. Therefore, we propose another heuristic based approach: Genetic Programming (GP) to implement the resource allocation algorithms. GP based algorithms are agnostic to the network architecture, and can be used for any general network architecture. Also, they are easy to implement and popular in general optimization domain. Both the ILS and GP processes rely on either (1) a Depth First Search (DFS), or (2) a random approach to find the initial solution. We compare the performances of both ILS and GP approaches, using DFS and random approaches as the baseline. To achieve scaling, we adopted to a simple scaling procedure: first we tried to allocate resources for a vertical scaling, and if it failed, then we tried to allocate resources for migration scaling.

In the proposed NFC Management System, the Resource Manager is responsible for the resource allocations (server and network resources), for VNFs in the cloud. In this chapter, we discuss our view of the resource allocation for VNFs in following aspects:

1. Formulation of the resource allocation problem as a set of ILP equations and evaluation of the implementation in CPLEX: Section 4.1

2. Description of the ILS based implementation of the Resource Manager with a comprehensive analysis for resource allocation algorithms: Section 4.2

3. Description of the GP based implementation of the Resource Manager with a comprehensive analysis for resource allocation algorithms: Section 4.3

4. A comparison between (1) ILP, (2) ILS, (3) GP, (4) DFS and (4) Random based resource allocation approaches: Section 4.4

4.1 Network Function Center Resource Management Problem (NFCRMP)

From the cloud resource allocation perspective, the NFC Resource Manager module has two main responsibilities:

1. **New policy requests provisioning:** upon receiving a new set of policies, the Resource Manager takes into account the physical network, servers constraints, and already allocated resources, to identify the resources where to instantiate the VNFs of the new policy.
2. **Scaling of existing policy requests**: upon receiving scaling requests from the Elasticity Manager, the Resource Manager decides the re-allocation of resources in order to satisfy traffic changes.

We formalize our resource allocation problem: the Network Function Center Resource Management Problem (NFCRMP), as a set of ILP equations for (1) resource allocation for new policy requests provisioning, and (2) dynamic resource re-allocation for Scaling of existing policy requests to support traffic changes. For the new policy requests provisioning, the NFCRMP aims at minimizing the required server and network resources (e.g., average link utilization.) For the scaling of existing policies, in addition to minimizing the required server and network resources, the NFCRMP also aims at minimizing the number of changes in server and links configurations. Table 4.1 and Table 4.2 provide a description of the key notations used in NFCRMP.

4.1.1 New policy requests provisioning

Once a new policy request is submitted by the client, the Resource Manager takes decisions on the placement and paths for the VNFs in the new policy. We call this as the problem of resource allocation for new policy requests provisioning.

We consider a NFC with M servers and L links. A link l connects a server to a switch, or a switch to another switch. The amount of resource capacity of server m is denoted H_m, and the network capacity of link l is denoted K_l. A path p between two servers (a source and a destination server pair), is composed by two or more links. P denotes the set of the shortest paths between all source and destination server pairs in the NFC. Given a path p in P, that connects server $m1$ and $m2$, Q_p represents the source server ($m1$), and R_p represents the destination server ($m2$). The variable E_l^p indicates whether link l is used on path p. The definition of the shortest path can vary based on the network architecture type, as each

Constants	
N	No. of VNFs, indexed by $n = 1, ..., N$
S_n	Server capacity required for VNF n
B_n	Bandwidth required for VNF n
M	No. of servers, indexed by $m = 1, ..., M$
H_m	Capacity of server m
L	No. of links, indexed by $l = 1, ..., L$
K_l	Bandwidth of link l
P	No. of paths, indexed by $p = 1, ..., P$
O_n	No. of paths required by VNF n
Q_p	Source server of the path p
R_p	Destination server of the path p
E_l^p	Indicates whether the link l is used on path p, $(0, 1)$
w_1, w_2, w_3	Weighting factors

Dynamic variables	
Z_n^m	A binary decision for placing VNF n on server m
A_n^p	A binary decision for routing traffic of VNF n on path p
U	Average of link capacity used percentages
G_m	A binary decision Server m is used/not
X	Total servers used
F_l	A binary decision Link l is used/not
Y	Total links used

Table 4.1: Summary of key notations: new policy requests provisioning

of them have different default maximum hop count for a path between two servers.

The number of VNFs running in the NFC is denoted by N. Each VNF n is characterized by its resource requirements: (1) the required server capacity (S_n), (2) the required bandwidth: the expected amount of the traffic flow (B_n), and (3) the required number of paths to route traffic to its successor(s) (O_n). The successor is the next VNF(s) according to the sequence in the policy chain. The number of paths to route traffic to its successor, is calculated by considering the required bandwidth and the current maximum link capacities of NFC. The required server capacity for each VNF to serve the expected amount of the traffic flow, can be derived using the inbound traffic rate and the resource profile of each VNF type. The resource profile of a VNF includes: (1) processing capacity of the VNF,

(2) required minimum number of CPU cores and (3) bandwidth demand transformation by the VNF. The bandwidth demand transformations are associated with traffic-scaling VNFs. For an example, certain VNFs, such as Redundant eliminators and Caches compress traffic, while other VNFs such as packet multiplication and encryptions amplify the traffic.

For each VNF n in the new policies, we find a server to place the VNF. The server must support the physical resource requirements of the VNF. Also, if the VNF is not the last VNF of the policy chain, we find the required number of paths to route the traffic of the VNF to its successor(s) in the policy chain. Links in the selected paths should support the bandwidth requirements of the VNF. We define Z_n^m to be a binary variable for placing VNF n on server m, such that,

$$Z_n^m = \begin{cases} 1 & \text{VNF } n \text{ is placed on server } m \\ 0 & \text{otherwise} \end{cases}$$

Let $G_m \in \{0, 1\}$ be a binary variable indicating whether server m is used in a configuration solution. Therefore, the total number of servers used in a configuration solution is:

$$X = \sum_{m=1}^{M} G_m$$

The traffic flow of VNF n to its successor is represented by the vector B_n. To configure the routing between VNF n and its successor, we need to find a path in P, joining the servers where VNF n and its successor reside. Therefore, we define A_n^p to be a binary variable that indicates if path p is used to route traffic between VNF n and its successor, that is,

$$A_n^p = \begin{cases} 1 & \text{traffic of } n \text{ to its sucessor is routed on path } p \\ 0 & \text{otherwise} \end{cases}$$

Let $F_l \in \{0, 1\}$ be a binary variable indicating whether link l is used in a configuration solution, and Y be the total number of links used in a configuration solution:

$$Y = \sum_{l=1}^{L} F_l$$

For link l, the percentage of link utilization is defined as:

$$(\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p . B_n)/K_l$$

Therefore, considering all the links in the configuration solution, the average percentage of link utilization U, is:

$$U = (\sum_{l=1}^{L}((\sum_{p=1}^{P}\sum_{n=1}^{N} A_n^p.E_l^p.B_n)/K_l))/L$$

The NFC Management System takes decisions on new policy requests provisioning, with the goals of minimizing the average link utilization and the number of servers used. As the NFC network has more than one path between most of the (source, destination) server pairs, and these paths mostly use different links, the NFC Management System tries to maximize the number of used links, so that the system is encouraged to use different paths to route traffic between each (source, destination) server pair. This results in distributing traffic over different paths, and reduces the average link utilization.

We derive the optimization function as given in the equation 4.1, for NFC Management System to take decisions on new policy requests provisioning. Since U is an average, we normalize the optimization function by considering the number of servers M, and links L. The objective function tries to minimize some objectives (average link utilization and the number of servers used), also to maximize some objectives (the number of links used), and therefore, the objective function requires a trade-off between the objectives. We consider the optimization function given in the equation 4.1 as a **Multi-Objective optimization (MOO)** and follow a traditional decomposition-based method, specifically weighted sum method to solve the problem [126, 127]. The weighted sum method for MOO is used extensively, not only to provide multiple solution points by varying the weights consistently, but also to provide a single solution point that reflects preferences presumably incorporated in the selection of a single set of weights. Therefore, we introduce weighting factors w_1, w_2, w_3 for optimization function given in the equation 4.1 to allow operators to tune the trade-offs between the optimization factors.

The NFCMP can be explained as the following constrained optimization:

Minimize:

$$w_1 \frac{1}{M}.X + w_2.U + w_3(1 - \frac{1}{L}.Y) \tag{4.1}$$

Variables:

Z_n^m	A binary decision for placing VNF n on server m
A_n^p	A binary decision for routing traffic of VNF n on path p
U	Average of link capacity used percentages
G_m	A binary decision Server m is used/not
X	Total servers used
F_l	A binary decision Link l is used/not
Y	Total links used

Subject to:

$$\sum_{m=1}^{M} Z_n^m = 1, \forall n \tag{4.2a}$$

$$\sum_{n=1}^{N} Z_n^m . S_n \leq H_m, \forall m \tag{4.2b}$$

$$\sum_{p=1}^{P} A_n^p = O_n, \forall n \tag{4.2c}$$

$$\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p . B_n \leq K_l, \forall l \tag{4.2d}$$

$$\sum_{p=1}^{P} A_n^p . Q_p - \sum_{m=1}^{M} Z_n^m . m = 0, \forall n \tag{4.2e}$$

$$\sum_{p=1}^{P} A_n^p . R_p - \sum_{m=1}^{M} Z_{n2}^m . m = 0, \forall (n, n2) \tag{4.2f}$$

Constraints (4.2a) and (4.2b) model the server resource constraints of the NFCMP. Constraint (4.2a) guarantees that each VNF in a policy is placed on one and only one server. Constraint (4.2b) guarantees that the total capacity consumed by all VNFs placed on a server does not exceed total capacity of that server. Constraints (4.2c) to (4.2f) model the network resource constraints of the NFCMP. If a VNF is not the last VNF of a policy, constraint (4.2c) guarantees that the VNF has the required number of paths to its successors. Constraint (4.2d) guarantees that for each link, the total bandwidth consumed by the VNFs does not exceed the total bandwidth of that link. Constraints (4.2e) and (4.2f) guarantee that the path(s) selected for a VNF to send traffic to its successor, starts from the server where the VNF resides (source server), and ends in the server where the VNF's successor resides (destination server).

4.1.2 Scaling of existing policy requests

After the initial resource allocation for initial placement of VNFs, over the time, the Resource Manager has to re-allocate the resources to satisfy dynamic traffic changes. Therefore, the Resource Manager has to find new placement and paths for the VNFs in the policies that has to be scaled. We call this as the problem of resource allocation for scaling of existing policy requests.

We consider a NFC with M servers and L links. A link l connects a server to a switch, or a switch to another switch. The amount of resource capacity of server m is denoted H_m, and the network capacity of link l is denoted K_l. A path p between two servers (a source and a destination

Constants	
N	No. of VNFs, indexed by $n = 1, ..., N$
S_n	Server capacity required for VNF n
B_n	Bandwidth required for VNF n
M	No. of servers, indexed by $m = 1, ..., M$
H_m	Capacity of server m
L	No. of links, indexed by $l = 1, ..., L$
K_l	Bandwidth of link l
P	No. of paths, indexed by $p = 1, ..., P$
O_n	No. of paths required by VNF n
Q_p	Source server of the path p
R_p	Destination server of the path p
E_l^p	Indicates whether the link l is used on path p, $(0, 1)$
N'	Previous state's no. of VNFs
M'	Previous state's no. of servers
L'	Previous state's no. of links
P'	Previous state's no. of paths
$(E_l^p)'$	Previous state's link l is used on path p
$(Z_n^m)'$	Previous state's binary decision for placing n on server m
$(A_n^p)'$	Previous state's binary decision for routing traffic of n on path p
w_1 to w_5	Weighting factors
Dynamic variables	
Z_n^m	A binary decision for placing VNF n on server m
A_n^p	A binary decision for routing traffic of VNF n on path p
U	Average of link capacity used percentages
G_m	A binary decision Server m is used/not
X	Total servers used
F_l	A binary decision Link l is used/not
Y	Total links used
C	Total servers changed from previous state to current state
D	Total links changed from previous state to current state

Table 4.2: Summary of key notations: scaling of existing policy requests

server pair), is composed by two or more links. P denotes the set of the shortest paths between all source and destination server pairs in the NFC. Given a path p in P, that connects server $m1$ and $m2$, Q_p represents the source server ($m1$), and R_p represents the destination server ($m2$). The variable E_l^p indicates whether link l is used on path p. The definition of the shortest path can vary based on the network architecture type, as each of them have different default maximum hop count for a path between two servers.

The number of VNFs running in the NFC is denoted by N. Each VNF n is characterized by its resource requirements: (1) the required server capacity (S_n), (2) the required bandwidth: the current amount of the traffic flow (B_n), and (3) the required number of paths to route traffic to its successor(s) (O_n). Here, the current amount of traffic flow can be different from the expected traffic flow that was defined at the initial placement of the VNF, because now the resources are going to be re-allocated to satisfy the dynamic traffic change. The successor is the next VNF(s) according to the sequence in the policy chain. The number of paths to route traffic to its successor, is calculated by considering the required bandwidth and the current maximum link capacities of NFC. The required server capacity for each VNF to serve the expected amount of the traffic flow, can be derived using the inbound traffic rate and the resource profile of each VNF type. The resource profile of a VNF includes: (1) processing capacity of the VNF, (2) required minimum number of CPU cores and (3) bandwidth demand transformation by the VNF. The bandwidth demand transformations are associated with traffic-scaling VNFs. For an example, certain VNFs, such as Redundant eliminators and Caches compress traffic, while other VNFs such as packet multiplication and encryptions amplify the traffic.

For each VNF n in the scaling policies, we find a server to place the VNF. The server must support the current physical resource requirements of the VNF. Also, if the VNF is not the last VNF of the policy chain, we find the required number of paths to route the traffic of the VNF to its successor(s) in the policy chain. Links in the selected paths should support the bandwidth requirements of the VNF. We define Z_n^m to be a binary variable for placing VNF n on server m, such that,

$$Z_n^m = \begin{cases} 1 & \text{VNF } n \text{ is placed on server } m \\ 0 & \text{otherwise} \end{cases}$$

Let $G_m \in \{0, 1\}$ be a binary variable indicating whether server m is used in a configuration solution. Therefore, the total number of servers used in a configuration solution is:

$$X = \sum_{m=1}^{M} G_m$$

The traffic flow of VNF n to its successor is represented by the vector B_n. To configure the routing between VNF n and its successor, we need to

find a path in P, joining the servers where VNF n and its successor reside. Therefore, we define A_n^p to be a binary variable that indicates if path p is used to route traffic between VNF n and its successor, that is,

$$A_n^p = \begin{cases} 1 & \text{traffic of } n \text{ to its sucessor is routed on path } p \\ 0 & \text{otherwise} \end{cases}$$

Let $F_l \in \{0, 1\}$ be a binary variable indicating whether link l is used in a configuration solution, and Y be the total number of links used in a configuration solution:

$$Y = \sum_{l=1}^{L} F_l$$

For link l, the percentage of link utilization is defined as:

$$(\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p . B_n)/K_l$$

Therefore, considering all the links in the configuration solution, the average percentage of link utilization U, is:

$$U = (\sum_{l=1}^{L} ((\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p . B_n)/K_l))/L$$

In the scaling situations, the optimization needs to consider the previous configurations of the system, so that we can minimize the disturbances to the current traffic flow when implementing the solutions provided by the optimization to satisfy the new traffic changes. Therefore, while trying to minimize the link utilization, and required servers in the optimization process, we also try to minimize the changes to the previous system.

We use additional variables to represent the previous state of the NFC, i.e., the state of the NFC before scaling. N', M', L' and P' represent the previous state's number of VNFs, number of servers, number of links and number of paths. $(E_l^p)'$, $(Z_n^m)'$ and $(A_n^p)'$ represent whether the previous state's link l is used on path p, the binary decision for placing n on server m, and the binary decision for routing traffic of n on path p.

Hence, the total number of servers changed, C, and total number of links changed, D, from the previous state to the current state, is captured by the following equations:

$$C = [\sum_{n=1}^{N} \sum_{m=1}^{M} \mid Z_n^m - (Z_n^m)' \mid] + [\mid N' - N \mid]$$

$$D = \sum_{n=1}^{N}\sum_{l=1}^{L}\sum_{p=1}^{P} \mid A_n^p.E_l^p - (A_n^p)'.(E_l^p)' \mid + \sum_{n=N+1}^{N'}\sum_{l=1}^{L}\sum_{p=1}^{P'}(A_n^p)'.(E_l^p)'$$

The NFC Management System takes decisions on scaling of existing policy requests, with the goals of minimizing the server and link changes, in addition to minimizing the average link utilization and number of servers used and maximizing the number of links used.

We derive the optimization function as given in the equation 4.3, for NFC Management System to take decisions on scaling of existing policy requests. Since U is an average, we normalize the optimization function by considering the number of servers M, and links L. The objective function tries to minimize some objectives (server and link changes, average link utilization and the number of servers used), also to maximize some objectives (the number of links used), and therefore, the objective function requires a trade-off between the objectives. We consider the optimization function given in the equation 4.3 as a MOO and follow a traditional decomposition-based method, specifically weighted sum method to solve the problem [126, 127]. We introduce weighting factors w_1 to w_5 for optimization function given in the equation 4.3 to allow operators to tune the trade-offs between the optimization factors.

The scaling component of the NFCRMP, can therefore be formalised as the following constrained optimization:

Minimize:

$$w_1\frac{1}{M}.X + w_2.U + w_3(1 - \frac{1}{L}.Y) + w_4\frac{1}{M}.C + w_5\frac{1}{L}.D \tag{4.3}$$

Variables:

Z_n^m	A binary decision for placing VNF n on server m
A_n^p	A binary decision for routing traffic of VNF n on path p
U	Average of link capacity used percentages
G_m	A binary decision Server m is used/not
X	Total servers used
F_l	A binary decision Link l is used/not
Y	Total links used
C	Total servers changed from previous state to current state
D	Total links changed from previous state to current state

Subject to:

$$\sum_{m=1}^{M} Z_n^m = 1, \forall n \tag{4.4a}$$

$$\sum_{n=1}^{N} Z_n^m . S_n \leq H_m, \forall m \tag{4.4b}$$

$$\sum_{p=1}^{P} A_n^p = O_n, \forall n \tag{4.4c}$$

$$\sum_{p=1}^{P} \sum_{n=1}^{N} A_n^p . E_l^p . B_n \leq K_l, \forall l \tag{4.4d}$$

$$\sum_{p=1}^{P} A_n^p . Q_p - \sum_{m=1}^{M} Z_n^m . m = 0, \forall n \tag{4.4e}$$

$$\sum_{p=1}^{P} A_n^p . R_p - \sum_{m=1}^{M} Z_{n2}^m . m = 0, \forall (n, n2) \tag{4.4f}$$

Constraints (4.4a) and (4.4b) model the server resource constraints of the NFCMP. Constraint (4.4a) guarantees that each VNF in a policy is placed on one and only one server. Constraint (4.4b) guarantees that the total capacity consumed by all VNFs placed on a server does not exceed total capacity of that server. Constraints (4.4c) to (4.4f) model the network resource constraints of the NFCMP. If a VNF is not the last VNF of a policy, constraint (4.4c) guarantees that the VNF has the required number of paths to its successors. Constraint (4.4d) guarantees that for each link, the total bandwidth consumed by the VNFs does not exceed the total bandwidth of that link. Constraints (4.4e) and (4.2f) guarantee that the path(s) selected for a VNF to send traffic to its successor, starts from the server where the VNF resides (source server), and ends in the server where the VNF's successor resides (destination server).

Evaluation

We implemented the ILP formulation of the NFCRMP in CPLEX [28] and carried out 3 set of experiments to calculate the time taken to find a solution to implement 10, 30 and 50 NFs with ILP approach in the case of new policy requests provisioning. We considered a small NFC with a k-fat tree architecture and assumed an environment with 2 pods and 4 servers, where each pod is connected 2 servers. The experiments were carried out in a machine with an Intel core i7-4500u processor and 8GB of RAM. ILP took 2.3, 4.6 and 7.2 hours respectively to find the optimal solution for 10, 30 and 50 NFs.

4.2 Iterated Local Search (ILS) based Resource Manager model

The resource allocation for the VNFs, specifically finding an optimal solution for the VNFs placement in the cloud, is a proven NP-hard problem [13, 14]. Furthermore, as shown in the previous Section 4.1.2, our results show that finding the optimal solution even for a few NFs with the ILP Equation (4.1) can take hours and is consequently not suitable to meet traffic changes in the NFV context. Instead of finding optimal solutions (e.g., solutions returned by an ILP solver), we believed it is more realistic to look for good feasible configurations fast. We have explored approximation techniques: heuristic based approaches that can be used to find a good feasible configuration fast.

First, we investigate our hypothesis, by proposing an Iterated Local Search (ILS) approach to implement the resource allocation algorithms. We model the problem as finding the *best fitted* solution according to an ILS model of the problem, after a fixed amount of repeated procedures have been explored. The two main responsibilities of the Resource Manager module – new policy requests provisioning, and scaling of existing policy requests – are implemented independently but both rely on ILS as the mechanism to allocate resources.

In this Section, we describe the proposed ILS based resources allocation algorithms in following aspects:

1. Present a description of the two ILS based algorithms for new policy requests provisioning, and scaling of existing policy requests, including the local search approach used: Sub-section 4.2.1
2. Present a comprehensive analysis of the performances of the ILS based algorithms: Sub-section 4.2.2
3. A summary of the performance evaluation: Sub-section 4.2.3

4.2.1 Overview

Iterated Local Search (ILS) is one of the most popular single-solution based meta-heuristics due its simplicity but at the same time powerful approach [4, 5, 23]. The essence of the ILS meta-heuristic is, ILS iteratively builds a sequence of solutions generated by the embedded heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic. ILS extends a problem-specific local search method by introducing a perturbation at each new local optimal solution, before restarting the search for a new local optimal solution.

As shown in Algorithm 1, ILS is built through four main steps: (1) Generation of an Initial Solution, (2) Local Search, (3) Perturbation and (4) Acceptance Criterion.

Algorithm 1 Iterated Local Search for the Resource Manager

1: **procedure** ITERATED LOCAL SEARCH
2: $S_0 = GenerateInitialSolution$
3: $S^* = LocalSearch(S_0)$
4:
5: *Repeat*:
6: $S' = Perturbation(S^*, history)$
7: $S^{*\prime} = LocalSearch(S')$
8: $S^* = AcceptanceCriteria(S^*, S^{*\prime}, history)$
9: *until termination condition met*

ILS aims at avoiding the disadvantages of random restarts by exploring the region of feasible solutions using a walk that steps from one local optimal solution S^* to a nearby one. Given the current solution S^*, a change or perturbation is first applied leading to an intermediate feasible solution, S'. Then, a Local Search is applied to S' to obtain a new local optimal solution, $S^{*\prime}$. If $S^{*\prime}$ passes an acceptance test, it becomes the new current solution; otherwise, one returns to the previous one, S^*.

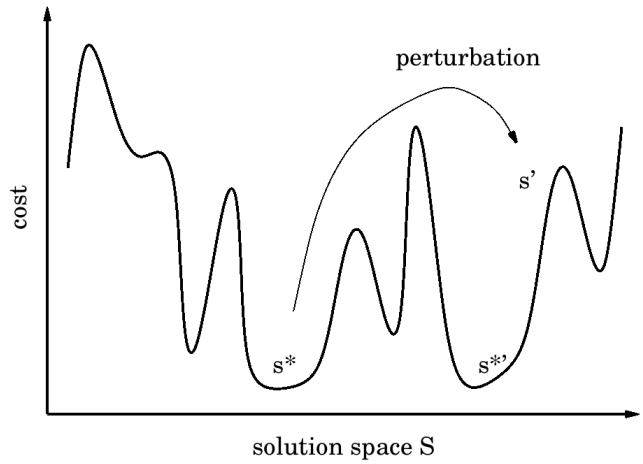

Figure 4.1: Pictorial representation of ILS [4, 5]

The Figure 4.1 shows a pictorial representation of ILS. Starting with a local minimum S^*, applying a perturbation leads to a solution S'. Then the applying a Local Search, leads to a new local minimum $S^{*\prime}$ that may be better than S^*.

The ILS process starts with the first phase of generating an initial feasible

solution. The Resource Manager selects a server and a path for the VNF, within the given physical network constraints and previously allocated resources for the existing policies. The server resources requirement for each VNF to serve the initial experted traffic rate, is derived using the inbound traffic rate and the resource profile of each VNF type. A resource profile for each VNF includes: processing capacity of the VNF, required minimum number of CPU cores and bandwidth demand transformation by the VNF. We have considered a simple Depth First Search (DFS) approach to select this initial server and path. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across.

The seconds phase of ILS is the local search, which is very important, because it must be able to improve the current solution in a very efficient way. We apply a simple iterative improvement local search, i.e. as soon a better solution is found, the process will restart using this solution as the current one. The efficiency of local search depends on the neighbourhood structures. A blind search on an un-organized neighbourhood structure may need considerable amount of time to find a better solution. We have observed that, by defining the search on the neighbourhood structure according to the specific network architecture (4-fat tree or BCube or VL2), helps the ILS process to improve the solution fast. More details on this can be found on Section 4.2.3.

A neighbourhood for the local search is defined by first choosing a type of transaction to obtain a new feasible solution from the current one: move, and then defining the neighbourhood as the set of solutions that can be obtained from the current one, by applying the same type of move. We define a simple move, by changing a VNF from one server to another feasible server: "1-opt-VNF move", and afterwards changing paths of the VNF affected.

For each VNF in each policy, its neighbours can be found by applying "1-opt-VNF moves". As shown in Algorithm 2, the local search continues through the neighbourhood, until a better solution is found. It is important to remark that the sequence of searching the neighbours heavily affect the computational performance of the algorithm. An objective function is used to measure how good a solution is. We use two different objective functions, one for the new VNFs provisioning: derived from the equation 4.1, and another for the scaling out/in: derived from the equation 4.3. As these objective functions are minimization functions, a solution is better only if its objective value is less than the current solution's objective value.

The main drawback of local search is that it can get trapped in local optima that are significantly worse than the global optimum. ILS escapes from local optima by applying perturbations to the current local minimum [4, 5]. Therefore, the perturbation process is the third phase of ILS. Generally, the local search should not be able to undo the perturbation, otherwise one will fall back into the local optimum just visited. However, a random

Algorithm 2 Local Search Procedure

```
1: procedure LOCAL SEARCH
2:
3:     Let S be an untested neighbour of S'
4:     If Z(S) < Z(S') then S' = S end if
```

move in a neighbourhood of higher order than the one used by the local search process can achieve this and will lead to a satisfactory algorithm. If the perturbations phase is designed by taking into account properties of the problem and are well matched to the local search algorithm, better results can be obtained [4, 5]. However, if the perturbation is too strong, ILS may behave like a random restart, so better solutions will only be found with a very low probability. On the other hand, if the perturbation is too small, the local search will often fall back into the local optimum just visited and the diversification of the search space will be very limited.

The perturbation process in our ILS based algorithms are implemented by applying n number of "1-opt-VNF moves", where $2 < n <<$Number of VNFs. It works by randomly selecting n number of VNFs (in randomly selected policies), and changing each selected VNF from one server to another feasible server: applying "1-opt-VNF move", and afterwards changing paths of the VNF affected.

ILS does a randomized walk in S^*: the space of the local minima. The perturbation phase together with the local search phase defines the possible transitions from a current solution s^* in S^*, to a neighbouring solution $s^{*\prime}$, which is also in S^*. The procedure acceptance criteria then determines whether $s^{*\prime}$ is accepted or not as the new current solution. The acceptance criteria procedure has a strong influence on the nature and effectiveness of the walk in S^*. It can be used to control the balance between intensification and diversification of that search.

As the acceptance criteria of our ILS based algorithms; a solution obtained after the perturbation and local search is accepted only if it improves the current solution, according to the previously mentioned objective functions. As the objective functions we have used are minimization functions, solution is accepted, only if its objective value is less than the current solution's objective value.

New policy requests provisioning: global ILS approach

For the new policy requests, first, the Resource Manager performs a simple DFS, and selects a server and a path for each VNF in each new policy request considering the expected traffic load by the policy, within the given physical network constraints and previously allocated resources for the existing policies. In the simple DFS method, servers and paths are selected by searching through the whole search space and selecting the

first solution we come across. This is known as the initial solution and is used as the initial input for the ILS algorithm. Within the local search procedure, the Resource Manager tries to perform "global ILS approach" to the given solution, where it applies "1-opt-VNF move", to all VNFs of all new policy requests (a standard full local search). In the perturbation process, "1-opt-VNF move" is applied to randomly selected n number of VNFs of new policy requests.

In the acceptance criteria phase, the objective function in equation 4.5 (derived from the ILP equation 4.1) is used to measure how good a solution is. It takes into account: number of servers used, links used and links congestion. As the objective function is a minimization function, a solution obtained after the perturbation and local search is accepted only if its objective value is less than the current solution's objective value. Unless specifically mentioned, for all our experiments, we assumed equal weights for all the parameters in the objective function.

Minimize

$$w_1 \frac{1}{M}.X + w_2.U + w_3(1 - \frac{1}{L}.Y) \tag{4.5}$$

X = No. of servers used
Y = No. of links used
M = Total no. of servers
L = Total no. of links
U = Avg of link capacity used percentages
w_1 to w_3 = Weighting factors

Scaling of existing policy requests: local ILS approach

For the scaling, the Resource Manager starts with the current state and search for the re-assignment of resources (servers and paths) for the set of VNFs that are scaling, using DFS method. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The current solution that describes the current state of the system is modified according to the new servers and paths found. The modified solution is given as the input to the ILS algorithm. Within the local search procedure, in contrast to the "global ILS approach", which is performed during the initial resource allocation process, when scaling we adopt a "local ILS approach". Because we want to minimize the changes to current configurations, local search and perturbation are carried out only to the VNFs which were changed because of the scaling (not to the all VNFs of the policies that are scaling).

In the acceptance criteria phase, the objective function in equation 4.6 (derived from the ILP equation 4.3) is used to measure how good a solution is. For the scaling out/in of VNFs, in addition to minimizing the required server and network resources, the Resource Manager also aims at minimizing the number of changes to the current server and links configurations. As the objective function is a minimization function, a solution

obtained after the perturbation and local search is accepted only if its objective value is less than the current solution's objective value. Unless specifically mentioned, for all our experiments, we assumed equal weights for all the parameters in the objective function.

Minimize

$$w_1 \frac{1}{M}.X + w_2.U + w_3(1 - \frac{1}{L}.Y) + w_4 \frac{1}{M}.C + w_5 \frac{1}{L}.D \qquad (4.6)$$

X = No. of servers used
Y = No. of links used
M = Total no. of servers
L = Total no. of links
C = Total servers changed
D = Total links changed
U = Avg of link capacity used percentages
w_1 to w_5 = Weighting factors

4.2.2 Evaluation

Experiments results, described in the previous section, show that the ILP approach is not suitable for large networks with dynamic requirements, but the ILS approach can find reasonable solutions fast. We have therefore conducted a more comprehensive evaluation of the performance of the NFC Management System when using our proposed ILS approach. The rest of the Sub-sections focus on this evaluation and specifically on the performance of the Resource Manager module when using the ILS approach for large networks with dynamic requirements.

Evaluation for new policy requests provisioning

Following sub-sections describe the results of the experiments that were carried out to evaluate the performances of ILS for one of the main functions of the Resource Manager: new policy requests provisioning. Again, all experiments were carried out in a machine with an Intel core i3 processor and 20GB of RAM. As described in Section 3.4, we assume that we are going to provide services for policy requests of 4 large enterprises, therefore the Resource Manager has to handle new policy requests that consists of a total of 400 VNFs.

Configuration solution improvement by ILS process

The ILS approach takes an initial solution as the input to the local search process, and tries to improve the given initial solution using local search and perturbations. Therefore, we can use these initial solutions as the baseline to compare the solutions given by the ILS process after 20 repeated procedures. We carried out our experiments for large networks with three different architectures (a 4-fat tree, a VL2 and a BCube): 50

rounds of experiments for each architecture, to explore how the quality of the initial solution was improved by the ILS process over the repeated procedures. For each round of experiment, we derived a fixed set of policies (average of 100 policies) that include a total of 400 VNFs and try to find the initial solutions with DFS approach in a 128 server environment for the three architectures separately. In the DFS approach, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. We assumed that each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units.

As the main goal of our optimization is to reduce the average link utilization, so that the network is less congested and future scaling requirements are minimized, the results produced by ILS reduced the average link utilization by 32%, 27.2% and 7% (compared to the DFS solution) respectively for the three architectures.

Effect of the number of repeated procedures

There is this natural intuition that, if we run the ILS process for more number of repeated procedures, ILS will be able to improve the quality of the solution more. To explore the effect of the number of repeated procedures and how the ILS process improves the initial solution over the repeated procedures, we conducted 30 round of experiments. For each round, we derived a new set of policies (average of 100 policies) that include a total of 400 VNFs.

First, we tried to find the initial solutions with DFS, and then improve the solution using ILS process. We count the times that the fitness value was improved during the ILS process. We assumed a 64 server environment in a 4-fat tree architecture where each server has an initial capacity of 2000 units, each link has an initial capacity of 6000 units and each VNF requires 75 server capacity units. Figure 4.2 shows how the initial objective value (given by DFS) was improved over the repeated procedures. It reflects the average objective value of 30 rounds of experiments. The important observation was that most of the improvements in the objective function (60% from total number of improvements) happened early on (during first 50 repeated procedures) and after that improvements decreased significantly.

Effect of the number of policies changed in perturbation

To better understand the effect of the number of policies that is allowed to change in perturbation phase (n), we carried out 4 sets of experiments: 5, 10, 15 and 20 policies were changed simultaneously. We conducted 30 rounds from each type of experiment set-up. We have assumed a 4-fat tree architecture with 64 servers. For each round of experiments, we used a fixed set of policy requests that consists of 400 VNFs as the initial policy requests. We counted the times that the objective value was improved

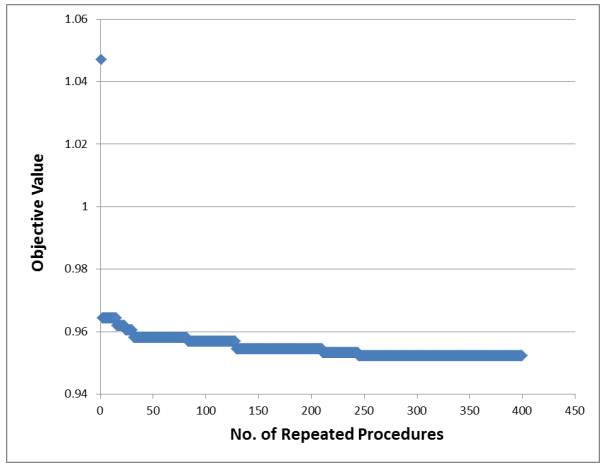

Figure 4.2: Improvement of the initial solution over the repeated procedures

during the ILS process. The important observation was that 35% of the improvements happened when only 5 policies were changed. Only 17% of the improvements happened when 20 policies were changed.

This is because, changing lots of policies in the perturbation phase caused the distribution of the VNFs all over the servers and, the chance of recovering from that less optimal configuration was less. Therefore, finding a better solution in next local search step was difficult. On the other hand, changing only 5 policies in perturbation cause less distraction and chances of leading to a better optimal solution were high. As mentioned in the Section 4.2.1, when the perturbation is too strong (changing lots of policies), ILS behaved like a random restart, so better solutions were found with a very low probability.

Effect of the number of servers

To explore how the ILS approach performs for large networks, we have carried out a set of experiments to calculate the total time taken by the ILS approach to provide a configuration solution for new policy requests provisioning (we have used 87 fixed policy requests that consists of 400 VNFs) in large networks. These total times include the time taken by the Resource Manager: (1) to perform DFS to come up with an initial solution and (2) to run the ILS process over 20 repeated procedures to improve the initial solution. As we have observed that even without performing a full local search, by just performing "1-opt-VNF move" to the 20% of the new policy requests in each local search round, ILS produced better results, for these experiments, we have used the same method.

The effect of total number of servers for the timing has been examined for three network architectures (4-fat tree, BCube and VL2) separately. We have conducted 50 rounds of experiments from each type and calculated the average. We assumed that each server has an initial capacity of 1000 units and each link has an initial capacity of 6000 units. We defined the capacity requirements of VNFs, in a way that 50% of the server capacities would be filled.

First, in all three types of architectures, the timings for DFS process is significantly smaller compared to timings for ILS process. For a network with 128 servers, timings are: (1) 4-fat tree 7589 μs, (2) BCube 7304 μs and (3) VL2 7336 μs and growth of the graph with respect to number of servers is linear.

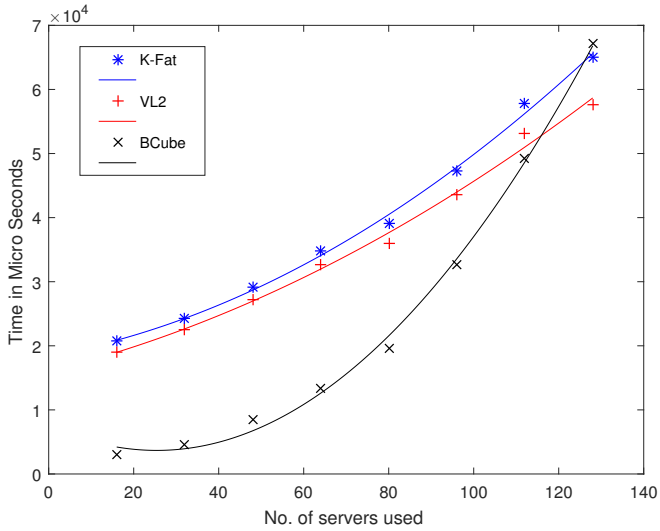

Figure 4.3: Factors effecting ILS timing: No. of Servers

.

Second, in all three architectures, the growth of the time taken for ILS process is quadratic with respect to the number of servers. This is because, when we increase the number of servers by one, 2 to 6 paths (depending on the network architecture) are added to each source destination server pair. In other words, 2 to 6 paths are created between newly added server and each of existing servers. Therefore, the number of times the local search with "1-opt-path move" is performed for each source destination server pair, is increased quadratically for each VNF.

Figure 4.3 shows the comparison of time taken for ILS process with 20 repeated procedures in different architectures when there are 16, 32, 48, 64, 80, 96, 112 and 128 servers in the CSP network. Fitting the plots into a quadratic polynomial of the form "$p1 * x^2 + p2 * x + p3$", we get within

the 95% confidence bounds the coefficient $p1$, for k-fat tree to be 1.938 (varying from 1.086 to 2.79), BCube to be 5.998 (varying from 4.899 to 7.097) and VL2 to be 1.309 (varying from 0.2461 to 2.372).

The growth of the graphs are very much similar in 4-fat tree and VL2 architectures where they share similar type of physical topology design. On the other hand, BCube has a totally different physical topology design and has a different growth in the timings. Therefore, it is very clear that the timings are not only depends on the number of servers, but also depends on the physical topology design, specially number of links and paths in the network.

Evaluation for scaling of existing policy requests

The second responsibility of the Resource Manager is to find on-line configuration solutions to implement dynamic scaling requirements of existing policy requests, according to the traffic changes. We have conducted experiments to explore the behaviour of ILS based algorithm that was designed for resource allocation for scaling requirements of existing policies.

In the algorithm we have adopted to the following scaling procedure. When there is a requirement for a VNF to be scaled, first we try to do a vertical scaling. We check whether the server and the path, that is currently used by the existing VNF instance, can handle the total resource requirement. If yes, then we do not need to change the current network configurations (we can use the same path), and we can allocate more resources to the existing VNF. If not, we try to perform migration scaling: we search for a new server and a path that can handle the total resource requirement. In this case, we assume that the existing VNF is migrated [128] to the new server and allocate additional resources.

The following Sub-sections describe the performance evaluation of the ILS approach when handling these re-allocation of resources in a NFC with 128 servers.

Effect of the local ILS approach

The "global ILS approach" allows changing configurations of any policy of the system during the local search and perturbation process, and may provides solutions with better resource allocations. However, these solutions may require drastic re-arrangements of the current configurations, disturbing the current workload of the CSP network, making them impractical in real scenarios. Instead, we can use this method to provide us with a baseline to compare against a "local ILS approach" where we limit changes only to configurations of VNFs that are scaling.

We conducted experiments for both ILS based global and local ILS approaches to compare performances with respect to the resources allocation efficiency over several days. As mentioned in Chapter 3.4, since in this thesis we did not explore the problem of monitoring the resources

and determining when to scale the VNFs according to the dynamic traffic changes, we used statistics presented in existing work and replicated the traffic changes over a day with relevant scaling triggers and events. We repeated the data for single day for three times to get the data to emulate 3 consecutive days, and therefore we had total of 132 events for 3 days. There are two types of events: (1) when the traffic change has reached the threshold, resources have to be reallocated to increase/decrease at least one VNF instance or (2) when the traffic change has not reached the threshold, modify the bandwidth usages of the links of the paths that were effected by the traffic change, to reflect the new traffic amount passing by.

The global ILS approach is considered as the baseline, where it only tries to minimize the required server and network resources (e.g., links and average link utilization). For the local ILS approaches, we explore 2 cases. First, we considered a situation where all parameters of the objective function are considered: "local ILS approach 1". In addition to minimizing the required server and network resources, it also tries to minimize the number of changes in server and links configurations. Second, we considered a situation where only the parameters relevant to changes are considered: "local ILS approach 2". The local ILS approach 2 strategy represented the scaling solutions that in theory minimally disturb the traffic (e.g., packet drops, latency), because it tried to minimize the changes to the servers and links.The weights of fitness function (given in equation 4.6) in these cases are shown in Table 4.3.

Fitness Function Usage	w1	w2	w3	w4	w5
Global ILS approach	1	1	1	0	0
local ILS approach 1	1	1	1	1	1
local ILS approach 2	0	0	0	1	1

Table 4.3: Different usages of fitness function given in equation 4.6 for global vs local ILS approaches (varying the weights)

Once the solutions have been provided for the 132 events for 3 consecutive days by the ILS approaches (global and two local methods), we manually calculated the value of the each objective function of the provided solutions, assuming that changes do not count (i.e., making the weights of the parameters relevant to server and link changes equal to zero) and compared the global and local ILS approaches. For each set of experiments, we have used a fixed set of policy requests that consists of 400 VNFs. We have assumed a 64 server network, where each server has an initial capacity of 1000 units, each link has an initial capacity of 6000 units and each VNF requires 100 server capacity units.

The objective value comparison (an average from 5 sets of experiments) for scaling events of 3 consecutive days for a 4-fat tree and a VL2 architecture network are shown in Figure 4.4 and Figure 4.5 respectively. For k-fat tree architecture, in the beginning, both local ILS approaches pro-

vided solutions as good as global ILS approach, but over the time, the global ILS approach produced solutions with better resource allocations than the other two. For VL2 architecture, from the beginning, global ILS approach produced better solutions than the other two. However, the local ILS approaches solutions do not diverge a lot from the global ILS approach solutions. Furthermore, the figure clearly shows that both local ILS approaches, followed essentially the same behaviour (modulo a translation in the y axis) that the behaviour of the baseline: global ILS approach, if we smooth the curves. For both architectures, local ILS approach 1 produced better solutions than local ILS approach 2.

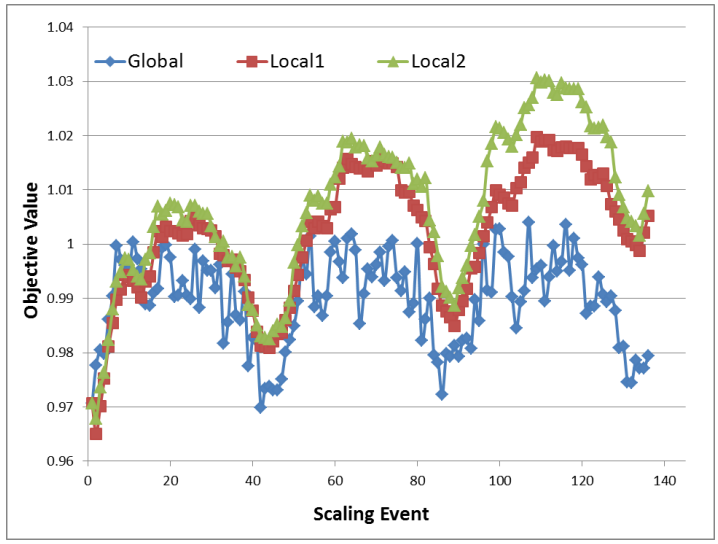

Figure 4.4: Fitness value comparison: KFat tree architecture

As we have shown in the results of both architectures, the objective values grew during most of the scaling events of each day. The reason is, in the traffic model presented in [2], the traffic is increasing until late night of each day. Each architecture's objective values were effected by different parameters of the objective function. First, in both architectures, the number of servers used were very similar while the number of links used and links utilization made the difference in the objective values. Second, when we compare the local ILS approaches and the global ILS approach in 4-fat tree as well as VL2 architecture, the difference was on link utilization. Third, the VL2 architecture's local ILS approach objective values increased fast over the days, with respect to the 4-fat tree. This is because, in the VL2 architecture, servers are located in a more compact manner and it has fewer paths between servers inside the same pod. Links of these paths got more congested, and when traffic was increasing the links utilization also increased fast. In contrast, in the 4-fat tree architecture, servers were

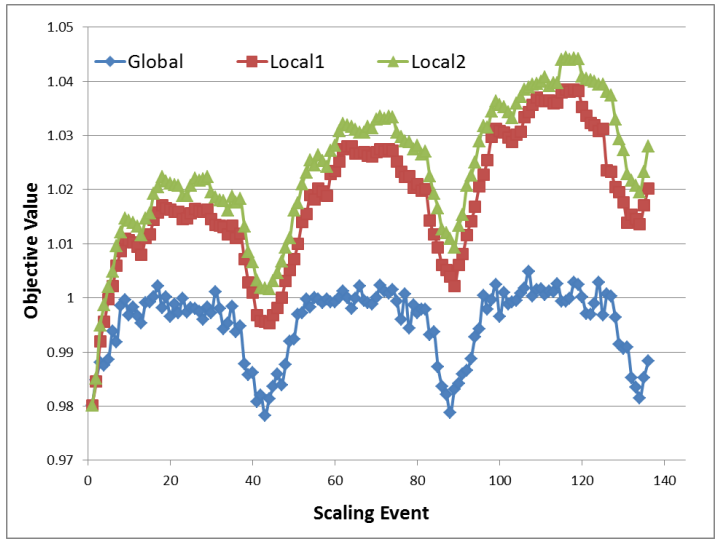

Figure 4.5: Fitness value comparison: VL2 architecture

not compact and had more paths between each other. Therefore, it was possible to use more links and made the links less congested. Hence, the 4-fat tree architecture had more smooth effect on the parameters of the objective function.

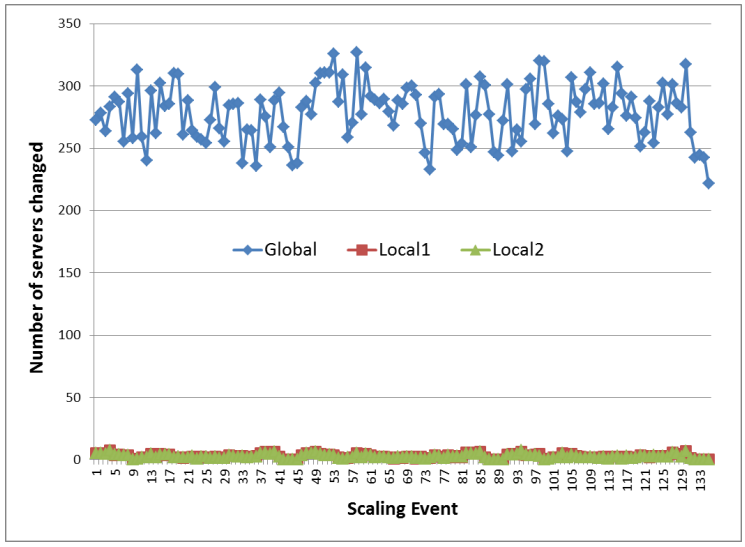

Figure 4.6: Server changes comparison: KFat tree architecture

Even though the global ILS approach provided better solutions, as expected, those solutions needed the largest number of changes to the current configurations. As shown in the figure 4.6, in the comparison of server changes needed to the current configuration solutions given by each ILS approach, the global ILS approach caused changes in the order of magnitude compared to the changes caused by local ILS approaches. The solutions given by the local ILS approaches had fewer server changes from their previous configuration because the local ILS approaches performed local search and perturbation process only on the solutions of policies that are scaling. In addition, similar to the server changes,the solutions given by the global ILS approach has most links changes from their previous configurations.

When we compare the two local ILS approaches, most of the time both methods had the same number of server and links changes, making the two methods essentially the same. We have noted that these servers and links changes are the unavoidable changes due to the scaling requirements, and not necessarily caused by the local search and perturbation process. But it is important to note that, local ILS approach 1 (which additionally minimizes usage of servers and links congestion) gave better objective values than local ILS approach 2 and lead to a better server and network resources utilization without incurring in a larger number of changes.

4.2.3 Summary

We introduced two new resource allocation algorithms for the Resource Manager module to: (1) provision new VNFs policy requests and (2) scale out/in of existing VNFs, based on ILS approach. For the new policy requests provisioning, we focused on minimizing the required server and network resources (e.g., average link utilization). For the scaling out/in of VNFs, in addition to minimizing the required server and network resources, we focused minimizing the number of changes in server and links configurations.

We explored the evolution of the ILS based algorithms for large networks, over a full day traffic patterns based on more realistic data. The results show that, although ILS approach may not provide the optimal solution, ILS can decide the computing and network allocations for hundreds of policies (around 400 VNFs) in a 128 server environment and find reasonable solutions on the order of milliseconds. Most of the improvements in the objective function (60% from total number of improvements) happened early on (during first 50 repeated procedures). Moreover, our results show that the ILS process provided an average objective value improvement percentage up to 8.45% over the initial solution (the baseline) with a reduction of average link utilization up to 32% for the three architectures (a 4-fat tree, a BCube and a VL2). In the evaluation of the algorithms over the time, our results show that, the “local ILS approach” provides reasonable solutions with lesser changes to the current configurations, and

moreover, they do not diverge from the "global ILS approach" solutions over time. The figure clearly shows that local ILS approaches (1) and (2), if we smooth the curves, will follow essentially the same behaviour (modulo a translation in the y axis) that the behaviour of the baseline: "global ILS approach".

As mentioned earlier, the efficiency of local search depends on the neighbourhood structures. A blind search on an un-organized neighbourhood structure may need considerable amount of time to find a better solution. We have observed that, by defining the search on the neighbourhood structure according to the specific network architecture (4-fat tree or BCube or VL2), helps the ILS process to improve the solution fast. For an instance, in a 4-fat tree architecture, trying to improve the solution by searching for a server which is in a different pod (instead of searching for a server that is in the same pod), allows the ILS to improve the solution fast. This is because, the use of new paths to cross the pod, helps ILS to distribute the load over different links and reduces the average link utilization. Therefore, we can introduce a constraint for the search on the neighbourhood, so that it only checks for the servers which are in different pods. The definition of the "different pod" and how to identify a different pod, depend on the network architecture. For an instance, in a 4-fat tree architecture, if the path between 2 servers has four or more links, the servers are in two different pods. Therefore, we can define our constraint for the search on the neighbourhood as "search for servers in which the path from the previous VNF's server has four or more links". In this way, instead of blindly going through the search space, by introducing constraints based on the network architecture helps the ILS to improve the solution fast. However, the limitation of this method is, the implementation of the algorithm depends on the architecture and a change to the network requires a change in the algorithm implementation.

4.3 Genetic Programming (GP) based Resource Manager model

In the previous Section, we introduced an ILS based Resource Manager model for the resource allocation. However, as we discussed, the proposed ILS based model has a limitation: the efficiency of the local search phase depends on the efficiency of the search on the neighbourhood structure, and it is directly related to the network architecture. Therefore, we wanted to explore more on approximation techniques that are agnostic of the network architecture, so that the resource allocation algorithms will be general enough for any type of network architecture. We modelled the resource allocation with another approximation technique: Genetic Algorithmic (GA), which is a heuristic based approach that is agnostic of the network architecture. It is general enough, in a way that a change to the network does not require a change in the algorithm implementation.

Therefore, we model the resource allocation problem as finding the *best fitted* solution according to a Genetic Algorithmic (GA) model of the problem, after a fixed amount of generations have been explored. The two main responsibilities of the Resource Manager module – new policy requests provisioning, and scaling of existing policy requests – are implemented independently but both rely on Genetic Programming (GP) as the mechanism to allocate resources.

In this Section, we describe the proposed GP based resources allocation algorithms in following aspects:

1. Present a description of the two GP based algorithms for new policy requests provisioning, and scaling of existing policy requests, including the genetic operations used: Sub-section 4.3.1
2. Present a comprehensive analysis of the performances of the GP based algorithms: Sub-section 4.3.2
3. A summary of the performance evaluation: Sub-section 4.3.3

4.3.1 Overview

GAs are part of evolutionary computing and were introduced as a computational analogy of adaptive systems [129]. They are modelled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation, inducing operators such as mutation and crossover. A fitness function is used to evaluate individuals, and reproductive success varies with fitness.

GAs can be described by the following five key steps [129]:

1. Generate an initial population $F(0)$ with n full solutions

2. Compute the fitness value $u(f)$ for each individual full solution f in the current population $F(t)$
3. Generate the next population $F(t+1)$, by selecting i best full solutions from $F(t)$
4. Produce offspring by applying the genetic operators to population $F(t+1)$
5. Repeat from Step 2 until a satisfying solution

Following the terms used in GA, a possible configuration state of the NFC (represented by the servers and paths assignments for VNFs) is considered as a full solution f, if it is an allocation of server and network resources for all the policies in the system. We call a configuration where only one of the policies has been allocated resources, a partial solution. If there are m number of policies in the NFC, then a full solution contains m number of partial solutions, each representing the allocation of resources (i.e., servers and paths) for each policy. The population $F(t)$ consists of n full solutions which represents different possible configuration states for the NFC.

We have considered two types of genetic operators to produce offspring: (1) mutation and (2) crossover. The crossover is a convergence operation which is intended to pull the population towards a local min or max. On the other hand, the mutation is a divergence operation which is intended to occasionally break one or more members of a population out of a local min/max space and potentially discover a better space. Since the end goal is to bring the population to convergence, crossovers happen more frequently (typically every generation). The mutation, being a divergence operation, should happen less frequently, and typically only affects a few members of a population in any given generation.

In our implementation, mutation is achieved via two independent operations: re-placement and re-wire. In the re-placement mutation we try to change the currently allocated server of a VNF – we remove the VNF from the current server and try to place it in a different server. In our initial work, we tried to change the server of a single VNF of a selected policy. However, we have observed that trying to change the placement of all VNFs of a policy and place them in a different (single) server provides better solutions than trying to change the placement of a single VNF of the policy. Therefore, we try to change the server of all VNFs of the selected policy, and try to place all the VNFs of that policy in a different server. Specifically, we select a random full solution from the population and randomly pick a partial solution from the selected full solution. We, then, attempt to find a new server where all VNFs in that partial solution can be placed on. If a new server is available to place the selected VNFs, then we find the necessary paths between selected VNFs and their successors by considering the new placement.

The next mutation is the re-wiring, where we try to change the path between two given VNFs and find a different path. Similar to re-placement mutation, we first select a random full solution from the population and randomly pick a partial solution from the selected full solution. Then, we select a random VNF in the partial solution and attempt to find a new path to its successor.

As for crossovers, we first select two random full solutions from the population and randomly pick partial solution from each selected full solution. Then, we check whether the configuration given in the first partial solution can be applied to the second partial solution and vice versa. If both ways are possible, then the configurations of partial solutions will be changed accordingly.

Each generation of the GP approach goes through mutations and crossovers. The newly generated solutions are evaluated according to a fitness function. We use two different fitness functions, one for the new VNFs provisioning: derived from the equation 4.1, and another for the scaling out/in: derived from the equation 4.3. As these fitness functions are minimization functions, a solution is better only if its fitness value is less than the current solution's fitness value.

New policy requests provisioning: global GP approach

For the new policy requests, the Resource Manager uses network's traffic, topology data, server constraints and the client requirements as inputs. Within the given physical network constraints and previously allocated resources for the existing policies, first, for each VNF in each new policy request, the Resource Manager selects: (1) a server depending on the server capacity requirement of the requested VNF and (2) a path(s) depending on the expected traffic load for the requested VNF. The server resources requirement for each VNF to serve the initial experted traffic rate, is derived using the inbound traffic rate and the resource profile of the VNF type. A resource profile of a VNF includes: processing capacity of the VNF, required minimum number of CPU cores and bandwidth demand transformation by the VNF (compress or amplify the traffic).

We have considered two types of initial selections: (1) Depth First Search (DFS), and (2) Random. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The random method searches servers and paths randomly anywhere in the network, until a feasible configuration is found. The configuration state (the servers and paths allocation) that the Resource Manager comes up with for a new policy request, i.e, a partial solution. Combination of all partial solutions (each representing a policy) forms a full solution.

Second, the Resource Manager applies the fitness functions given in the equation 4.7 (derived from the ILP equation 4.1), to each full solution.

Unless specifically mentioned, for all our experiments, we assumed equal weights for all the parameters in the fitness function. Third, the full solutions that return small values are preferred, and selected as the best solutions for the next generation population. Fourth, the Resource Manager performs "global GP approach" where it applies genetic operators (mutations and crossover) on randomly selected partial solutions of randomly selected full solutions to generate offsprings, i.e., new full solutions. The last three steps are repeated until x number of generations are explored and the best full solution is selected as the configuration for the new VNFs provisioning.

Minimize

$$w_1 \frac{1}{M}.X + w_2.U + w_3(1 - \frac{1}{L}.Y) \tag{4.7}$$

X = No. of servers used
Y = No. of links used
M = Total no. of servers
L = Total no. of links
U = Avg of link capacity used percentages
w_1 to w_3 = Weighting factors

Scaling of existing policy requests: local GP approach

For the scaling, the Resource Manager starts with the current state and search for the re-assignment of resources (servers and paths) for the set of VNFs that are scaling, using DFS/random methods. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The random method searches servers and paths randomly anywhere in the network, until a feasible configuration is found. Partial solutions relevant to the scaling are modified according to the servers and paths found. The fitness function given in the equation 4.8 (derived from the ILP equation 4.3), is used to measure how good each full solution is. Unless specifically mentioned, for all our experiments, we assumed equal weights for all the parameters in the fitness function.

In contrast to the "global GP approach", which is performed during the initial resource allocation process, when scaling we adopt a "local GP approach". Because we want to minimize the changes to current configurations, mutations and crossovers are carried out only to the VNFs which were changed because of the scaling (not to the all VNFs of the policies that are scaling). As mentioned earlier, the process is continued until x number of generations are explored and the best full solution is selected as the configuration for re-assignment of the policy.

Minimize

$$w_1 \frac{1}{M}.X + w_2.U + w_3(1 - \frac{1}{L}.Y) + w_4 \frac{1}{M}.C + w_5 \frac{1}{L}.D \tag{4.8}$$

X = No. of servers used
Y = No. of links used
M = Total no. of servers
L = Total no. of links
C = Total servers changed
D = Total links changed
U = Avg of link capacity used percentages
w_1 to w_5 = Weighting factors

4.3.2 Evaluation

Experiments results, described in the previous section, show that the ILP approach is not suitable for large networks with dynamic requirements, but the GP approach can find reasonable solutions fast. We have therefore conducted a more comprehensive evaluation of the performance of the NFC Management System when using our proposed GP approach. The rest of the Sub-sections focus on this evaluation and specifically on the performance of the Resource Manager module when using the GP approach for large networks with dynamic requirements.

Evaluation for new policy requests provisioning

In the following Sub-sections, we will describe the results of the experiments that were carried out to evaluate the performances of GP for one of the main functions of the Resource Manager: new policy requests provisioning. Again, all experiments were carried out in a machine with an Intel core i3 processor and 20GB of RAM. As described in Section 3.4, we assume that we are going to provide services for policy requests of 4 large enterprises, therefore the Resource Manager has to handle new policy requests that consists of a total of 400 VNFs.

Configuration solution improvement by GP process

The GP approach takes an initial solution as the input to the GP process, and tries to improve the given initial solution using genetic operations. Therefore, we can use these initial solutions as the baseline to compare the solutions given by the GP process after 200 generation. We carried out our experiments for large networks with three different architectures (a 4-fat tree, a VL2 and a BCube): 50 rounds of experiments for each architecture, to explore how the quality of the initial solution was improved by the GP process over the generations. For each round of experiment, we derived a fixed set of policies (average of 100 policies) that include a total of 400 VNFs and try to find the initial solutions with DFS approach in a 128 server environment for the three architectures separately. In the DFS approach, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. We assumed that each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units.

As the main goal of our optimization is to reduce the average link utilization, so that the network is less congested and future scaling requirements are minimized, the results produced by GP reduced the average link utilization by 28.7%, 14.9% and 3.2% (compared to the DFS solution) respectively for the three architectures.

Effect of the number of generations

There is this natural intuition that, if we run the GP process for more number of generations, GP will be able to improve the quality of the solution more. To explore the effect of the number of generations and how the GP process improves the initial solution over the generations, we conducted 30 round of experiments. For each round, we derived a new set of policies (average of 100 policies) that include a total of 400 VNFs.

First, we tried to find the initial solutions with DFS, and then improve the solution using GP process. We count the times that the fitness value was improved during the GP process. We assumed a 128 server environment in a 4-fat tree architecture where each server has an initial capacity of 2000 units, each link has an initial capacity of 6000 units and each VNF requires 75 server capacity units.

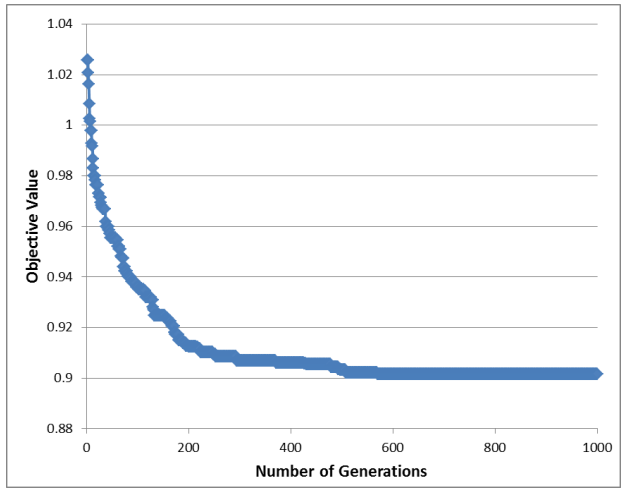

Figure 4.7: Improvement of the initial solution over the generations

Figure 4.7 shows how the initial objective value (given by DFS) was improved over the generations. It reflects the average objective value of 30 rounds of experiments. The important observation was that most of the improvements in the fitness function (52% from total number of improvements) happens early on (during first 100 generations) and after that improvements decrease significantly. In fact there were very few improvements after 400 generations: 6% from the total number of improvements.

Effect of the number of servers and nodes in the NFC

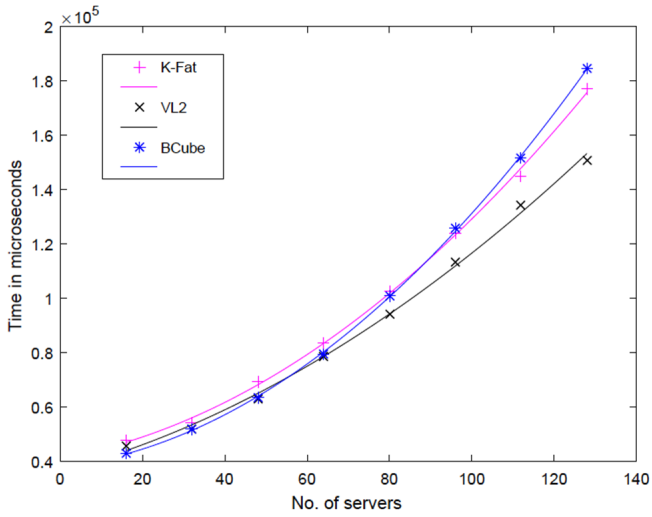

Figure 4.8: Factors effecting GP timing: No. of Servers

To better understand how the GP approach performs for large networks, we carried out a set of experiments to calculate the total time taken by the GP approach to provide a solution for new policy requests provisioning (we used 87 fixed policy requests that consists of 400 VNFs) in large networks. These total times include the time taken by the Resource Manager: (1) to perform DFS to come up with an initial solution and (2) to run the GP process over generations to improve the initial solution.

With the current implementation of the algorithm, most of the steps of the DFS process, such as finding a server or a path for a VNF in the policy, are performed in logarithmic time (the server capacities are stored in a sorted balanced tree and operations to the tree such as searching and updating can be done in a logarithmic time). However, in the GP process, when we perform a genetic operation and try find an improved solution to grow the population, we keep a copy of the original solution. We observed that the time taken for the GP process is dominated by this copying process. The complexity of copying the original solution, depends on the size of data structures that store servers and links current usage information. In our implementation, a link is represented as a connection between two nodes, where a node can be a server or a switch of the NFC. Even though it is not necessarily that there is a link between each and every node in the network, we used an 2D array to store links usage information, with a row and a column representing each node of the NFC. The total number of nodes in the network depends on two factors: (1) the number of servers in the NFC and (2) physical topology of the NFC (Section 3.4). When we increase the

number of servers, the 2D array that stores links usage information grows quadratic. Therefore, when we perform genetic operations, timing for the process of copying the original solution grows quadratic too.

The effect of total number of servers for the timing was examined for three network architectures (4-fat tree, BCube and VL2) separately. We conducted 50 rounds of experiments from each type and calculated the average. We assumed that each server has an initial capacity of 1000 units and each link has an initial capacity of 6000 units. We defined the capacity requirements of VNFs, in a way that 50% of the server capacities is filled.

First, in all three types of architectures, the timings for DFS process is significantly smaller compared to timings for GP process. For a network with 128 servers, timings are: (1) 4-fat tree 5489 μs, (2) BCube 5204 μs and (3) VL2 5236 μs and growth of the graph with respect to number of servers is linear. Second, in all three types of architectures, the timings for the GP process is dominated by the process of copying original solution during genetic operations. Figure 4.8 (Left) shows the comparison of time taken for GP process with 200 generations in different architectures when there are 16, 32, 48, 64, 80, 96, 112 and 128 servers in the NFC. We observed, in all three architectures, the growth of the graph is quadratic with respect to the number of servers and when they are plotted in the same figure, three graphs fall on top of each other. Fitting the plots into a quadratic polynomial of the form "$p1 * x^2 + p2 * x + p3$", we get within the 95% confidence bounds the coefficient $p1$, for k-fat tree to be 6.107 (varying from 4.661 to 7.553), BCube to be 5.948 (varying from 5.431 to 6.465) and VL2 to be 3.972 (varying from 2.308 to 5.637).

As we mentioned earlier, the total number nodes in the network depends on two factors: the number of servers and the physical topology. In a situation where there are fixed number of servers in the NFC, the total number of nodes in the NFC will depend on the physical topology. Therefore, the timings for the GP process with a fixed number of servers in different network architectures will vary, depending on the total number of nodes. We explored the time taken for GP process with 200 generations in different architectures with respect to different number of nodes when there are 16, 32, 48, 64, 80, 96, 112 and 128 servers in the NFC. In all three architectures, the growth of the graph is quadratic with respect to the number of nodes in the network and when they are plotted in the same figure, three graphs fall on top of each other.

Effect of the state of the NFC

To better understand the effect of the state of the NFC, to the improvements to the solution during the GP process, we carried out 10 set of experiments in four types of 128 server environments of a data center with a 4-fat tree architecture network: (1) an environment where 80% of the server and links capacity is full: Very tight (2) an environment where only 70% of the server and links capacity is full: Tight, (3) an environment

where only 50% of the server and links capacity is full: Medium and (4) an environment where only 30% of the server and links capacity is full: Loose. We assumed that each server has an initial capacity of 1000 units and each link has an initial capacity of 6000 units. We used 92 fixed policy requests that consists of 400 VNFs. We count the times that the fitness value was improved during the GP process.

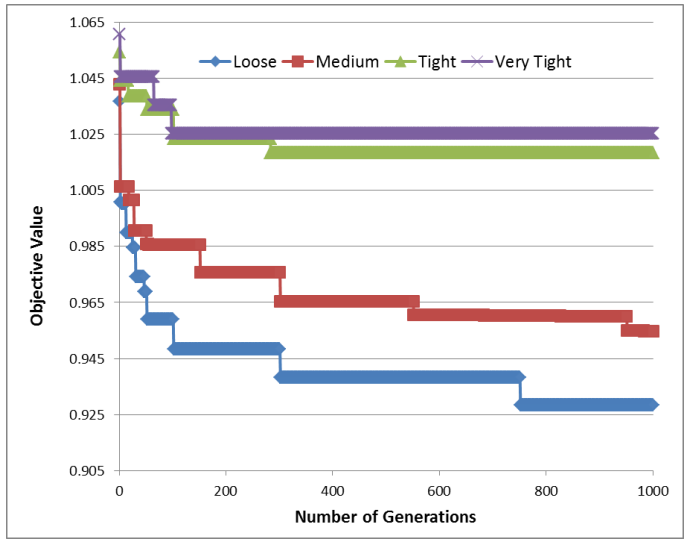

Figure 4.9: Improvement of the initial solution over the generations with respect to different states of NFC

Figure 4.9 shows how the initial objective value (given by DFS) was improved over the generations for each type of environment. It reflects the average objective value of each 10 sets of experiments. The first observation is that, in all types of environments, most of the improvements (80% from total number of improvements) in the fitness function happens early on (during first 100 generations), and after that improvements decrease significantly. The second observation is that, the environments with loosely tight resource availability get more improvements (33% from total number of improvements) than tighter environments (15% from total number of improvements).

Effect of the order of policy requests

Since we are processing policies in a new provisioning request sequentially, we needed to check the impact of the order policies in the results. We used fixed 83 policies that includes 400 VNFs and processed them in 100 random orders for an environment of 128 servers in a 4-fat tree architecture. Our results showed that the order of the policies does not impact the quality

of the solution. Only five different fitness function values were obtained with an average value of 1.072 and a standard deviation of 0.0058.

Evaluation for scaling of existing policy requests

The second responsibility of the Resource Manager is to find on-line configuration solutions to implement dynamic scaling requirements of existing policy requests, according to the traffic changes. We have conducted experiments to explore the behaviour of GP based algorithm that was designed for resource allocation for scaling requirements of existing policies.

In the algorithm we have adopted to the following scaling procedure. When there is a requirement for a VNF to be scaled, first we try to do a vertical scaling. We check whether the server and the path, that is currently used by the existing VNF instance, can handle the total resource requirement. If yes, then we do not need to change the current network configurations (we can use the same path), and we can allocate more resources to the existing VNF. If not, we try to perform migration scaling: we search for a new server and a path that can handle the total resource requirement. In this case, we assume that the existing VNF is migrated [128] to the new server and allocate additional resources.

The following Sub-sections describe the performance evaluation of the GP approach when handling these re-allocation of resources in a NFC with 128 servers.

Effect of the number of VNFs scaling

To better understand how the NFC behaves when handling different number of VNFs scaling out simultaneously, we carried out 4 sets of experiments (30 rounds from each): (1) 30 VNFs were scaling out simultaneously, (2) 20 VNFs were scaling out simultaneously, (3) 10 VNFs were scaling out simultaneously, and (4) 2 VNFs were scaling out simultaneously. We have assumed a 4-fat tree architecture with 128 servers, where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF initially requires 100 server capacity units. For each round of experiments, we have used a fixed set of policy requests that consists of 400 VNFs as the initial policy requests. We count the times that the fitness value was improved during the GP process.

Figure 4.10 shows how the initial objective value (given by DFS) was improved over the generations when different number of VNFs are scaling. It reflects the average objective value of each 30 rounds of experiments. The observation is that there is little benefit in running the GP process when the number of VNFs scaling simultaneously is small. The number of improvements when 2 VNFs were scaling was 6% from total number of improvements. One can avoid running the GP process and use directly the DFS solution. As the number of VNFs scaling simultaneously increases, the potential gains provided by the GP optimization also increases. The

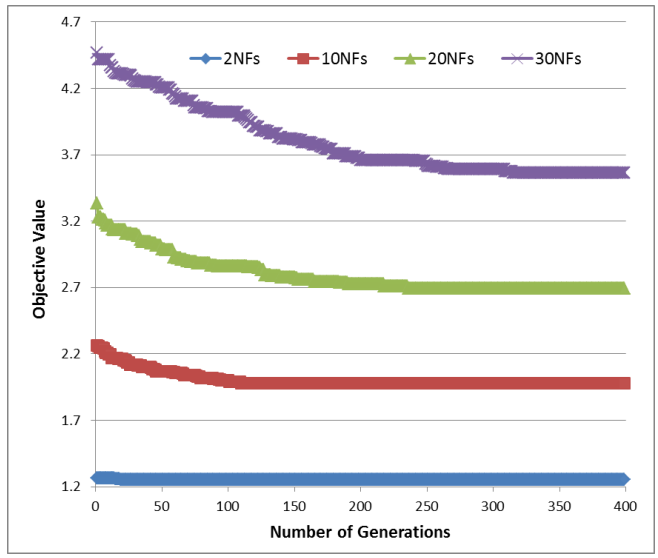

Figure 4.10: Improvement of the initial solution over the generations with respect to the number of VNFs scaling

number of improvements when 30 VNFs were scaling was 41% from total number of improvements. In all cases, most of the improvement in the fitness function (66% from total number of improvements) happens early on (during first 100 generations), and after that the improvements decrease significantly.

Effect of the local GP approach

Although the "global GP approach" where we are allowed to change configurations of any policy of NFC during the genetic operations, may provide better resource allocations, the solutions may require drastic rearrangements of the current configurations, hence making them impractical in real scenarios. However, we can use this method to provide us with a baseline to compare against a "local GP approach" where we limit changes to configurations of policies that are scaling. We conducted experiments for both GP based global and local GP approaches to compare performances with respect to the resource allocation efficiency over several days. As mentioned in Chapter 3.4, since in this thesis we did not explore the problem of monitoring the resources and determining when to scale the VNFs according to the dynamic traffic changes, we used statistics presented in existing work and replicated the traffic changes over a day with relevant scaling triggers and events. We repeated the data for single day (we derived 42 significant events over the 24 hours of traffic data for two times to get the data to emulate 2 consecutive days, and therefore we had total of 84 events for 2 days. There are two types of events: (1) when the

traffic change has reached the threshold, resources have to be reallocated to increase/decrease at least one VNF instance or (2) when the traffic change has not reached the threshold, modify the bandwidth usages of the links of the paths that were effected by the traffic change, to reflect the new traffic amount passing by.

We called the global GP approach the baseline, i.e., the minimization of the parameters relevant to server and links usage. For the local GP approach, we focused on 2 limited cases: (1) all parameters are considered and (2) only the parameters relevant to changes are considered. Since the second case tried to minimize the changes to the servers and links, it represented the scaling solutions that in theory minimally disturb the traffic (e.g., packet drops, latency). The weights of fitness function (given in equation 4.8) in these cases are shown in Table 4.4.

Fitness Function Usage	w1	w2	w3	w4	w5
Global GP approach	1	1	1	0	0
Local GP approach 1	1	1	1	1	1
Local GP approach 2	0	0	0	1	1

Table 4.4: Different usages of fitness function given in equation 4.8 for global vs local GP approaches (varying the weights)

Once the solutions have been provided for the 84 events for 2 consecutive days by the GP approachs (global and two local methods), we manually calculated the value of the each fitness function of the provided solutions, assuming that changes do not count (i.e., $w_4 = w_5 = 0$) and compared the global and local GP approaches. For each set of experiments, we have used a fixed set of policy requests that consists of 400 VNFs. We have assumed a 128 server network, where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units.

The fitness value comparison (an average from 5 sets of experiments) for the events of 2 consecutive days for a 4-fat tree architecture network is shown in figure 4.11. The global GP approach produced solutions with better resource allocations than the other two. However, the local GP approaches solutions do not diverge from the global GP approach solutions over time. Furthermore, the figure clearly shows that both local GP approaches, followed essentially the same behaviour (modulo a translation in the y axis) that the behaviour of the baseline: global GP approach, if we smooth the curves.

A comparison of server changes needed in the configuration solutions (an average from 5 sets of experiments) given by (1) global GP approach, (2) local GP approach 1 and (3) local GP approach 2, after processing each event is shown in the Figure 4.12. As expected, the global GP approach caused the largest number of changes. The solutions given by the local GP approaches had fewer server changes from their previous configuration

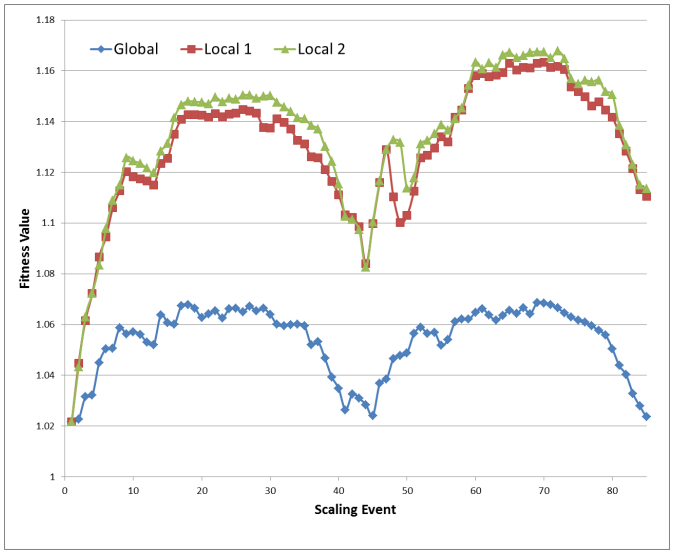

Figure 4.11: Fitness value comparison: KFat tree architecture

because the local GP approaches performed genetic operations only on the partial solutions that are scaling. The interesting observation was that, most of the time both local GP approaches had the same number of server changes, making the two methods essentially the same. We have noted that these server changes are the unavoidable changes due to the scaling requirements, and not necessarily caused by the genetic operations.

In addition, similar to the server changes, we have observed that the solutions given by the global GP approach has most links changes from their previous configuration. Most of the time both local GP approaches have the same number of links changes. As shown earlier, local GP approach 1 (which additionally minimizes usage of servers and links congestion) gave better fitness values than local GP approach 2. Therefore, local GP approach 1 provided solutions with better server and network resources utilization without making many changes. Although we have included the local GP approaches comparison results only for a 4-fat tree architecture network because of the limited pages, we have observed that this behaviour is the same for the other two architectures: BCube and VL2. Therefore, in the rest of the experiments, we have used only local GP approach 1.

Effect of the NFC architecture

Going further, we have compared the behaviour of local GP approach 1 with global GP approach for other architectures: BCube and VL2 architectures for 2 consecutive days. The fitness values (an average from 5 sets of experiments) for BCube and VL2 architectures for 2 consecutive days

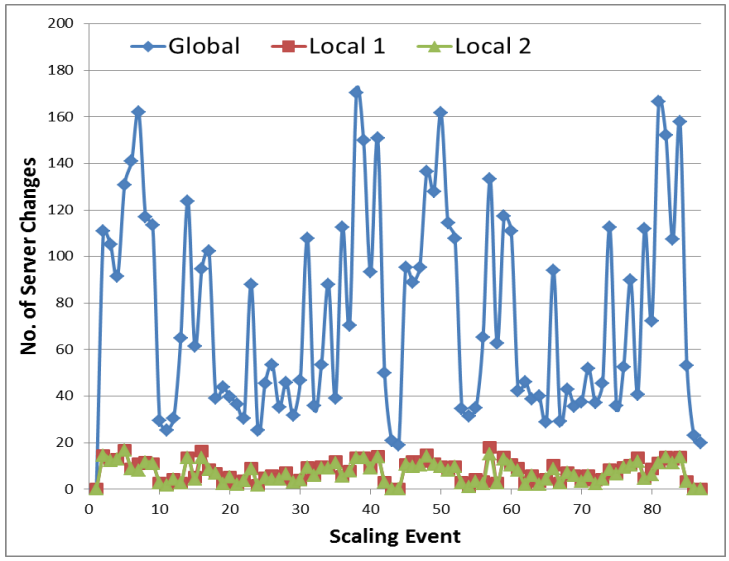

Figure 4.12: Server changes comparison: KFat tree architecture

are shown in Figure 4.13 and 4.14 respectively. As mentioned earlier, for each set of experiments, we have used a fixed set of policy requests that consists of 400 VNFs. We have assumed a 128 server network, where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units.

Similar to the 4-fat tree architecture, the global GP approach produced solutions with better resource allocations than the local GP approach for BCube and VL2 architectures. Also, they followed essentially the same behaviour (module a translation in space) of the global GP approach: the baseline. In the results of all three architectures, during most of the scaling events of each day, the fitness values grew. The reason is, in the traffic model we are using, the traffic is increasing until late night of each day.

We have observed that each architecture's fitness values are effected by different parameters of the fitness function. For all three architectures, number of links used and links utilization made the difference in the fitness values and the number of servers used were very similar. In the BCube architecture, the difference between local and global GP approach was due to the fact the local GP approach always used fewer links than the global GP approach. While the global GP approach freely used more links over time, the local GP approach hesitated to use more links because we were trying to minimize number of server and links changes in the local GP approach. Therefore, the solutions given by the local GP approach were more congested than the solutions given by the global GP approach.

In the VL2 and 4-fat tree architectures, the number of links used was

similar for both the local GP approach and the global GP approach, while the difference was on link utilization. When considering the fitness values increase for each day, the VL2 architecture's fitness values for the local GP approach increased fast with respect to the 4-fat tree and BCube. In the VL2 architecture, servers are located in a more compact manner and it has fewer paths between servers inside the same pod. Therefore, the links got more congested, and when traffic was increasing the link utilization also increased fast. The 4-fat tree architecture has more paths and therefore the servers were not compact. It tried to use more links and made the links less congested. Hence, the 4-fat tree architecture had more smooth effect on the parameters of the fitness function.

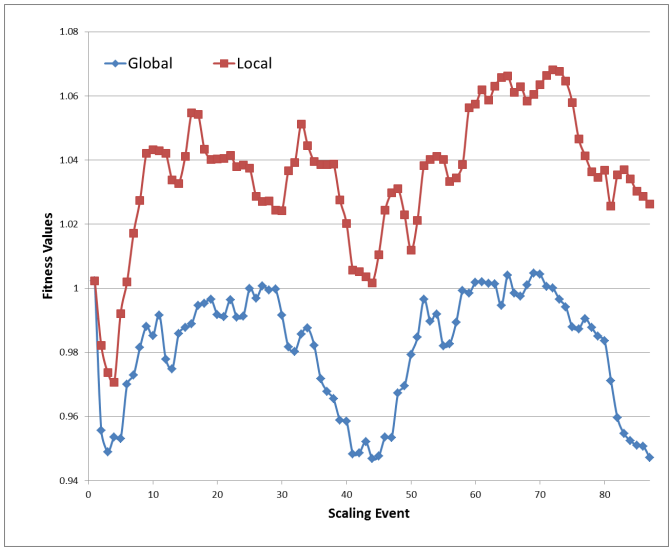

Figure 4.13: Fitness value comparison: BCube architecture

4.3.3 Summary

We introduced two new resource allocation algorithms for the Resource Manager module to: (1) provision new VNFs policy requests and (2) scale out/in of existing VNFs, based on GP approach. For the new policy requests provisioning, we focused on minimizing the required server and network resources (e.g., average link utilization). For the scaling out/in of VNFs, in addition to minimizing the required server and network resources, we focused minimizing the number of changes in server and links configurations.

We explored the evolution of the GP based algorithms for large networks, over a full day traffic patterns based on more realistic data. The results show that, although GP approach may not provide the optimal solution,

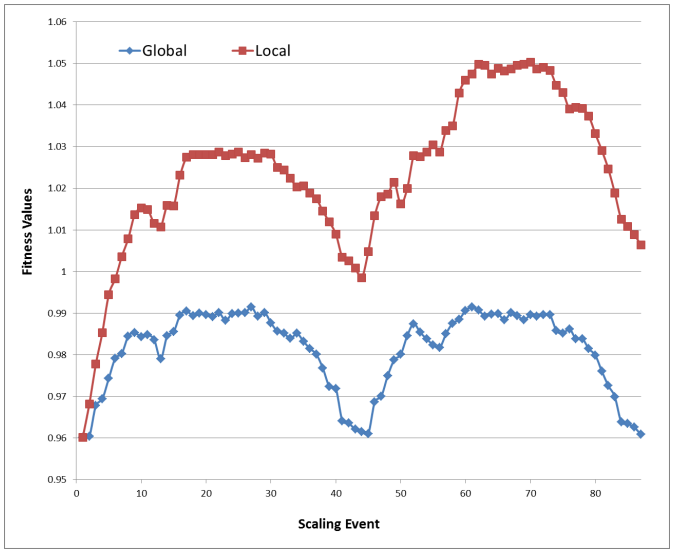

Figure 4.14: Fitness value comparison: VL2 architecture

GP can decide the computing and network allocations for hundreds of policies (around 400 VNFs) in a 128 server environment and find reasonable solutions on the order of milliseconds. Most of the improvements in the objective function (80% from total number of improvements) happens early on (during first 100 generations). Moreover, our results show that the GP process provided an average objective value improvement percentage up to 7.87% over the initial solution (the baseline) with a reduction of average link utilization up to 28.7% for the three architectures (a 4-fat tree, a BCube and a VL2). In the evaluation of the algorithms over the time, our results show that, the "local GP approach" provides reasonable solutions with lesser changes to the current configurations, and moreover, they do not diverge from the "global GP approach" solutions over time. The figure clearly shows that local GP approaches (1) and (2), if we smooth the curves, will follow essentially the same behaviour (modulo a translation in the y axis) that the behaviour of the baseline: "global GP approach".

As opposed to the ILS based model, in which the efficiency of local search depends on the neighbourhood structures and neighbourhood structures are directly related to the network architecture, the GP approach is agnostic of the network architecture. It is general enough, in a way that a change to the network does not require a change in the algorithm implementation.

4.4 Performances comparison for Resource Manager models

As discussed in previous Sections, we have implemented the resource allocation algorithms for the Resource Manager module using different approaches. Specifically: (1) ILP, (2) ILS and (3) GP. Further more, both the ILS and GP processes rely on either (1) DFS or (2) random approach to find the initial solution. Each approach has its own advantages as well as disadvantages, in terms of: (1) execution time of the algorithm, (2) quality of the solution, (3) scalability of the algorithm for large scaled networks and (4) agnosticism to the data center architecture etc. The selection of which algorithm to use, might depend on the requirements and optimization goals of the CSP. So it is important to understand the advantages of each approach and the scenarios where each approach would be better to use.

In this Section, we compare the approaches that we used to implement the resource allocation algorithm in following aspects:

1. Performance comparison for a small scaled network: Sub-section 4.4.1
2. Performance comparison for a large scaled network: Sub-section 4.4.2

4.4.1 Small scaled networks

We start our comparison by exploring the performances of different approaches for a small scaled network. We compare the heuristics based approaches: ILS and GP with the traditional ILP approach in the case of new policy requests provisioning. However, as explained in earlier chapters, both the ILS and GP processes rely on either (1) DFS, or (2) random approach to find the initial solution. As DFS and random approaches are yet another two different approaches of resource allocation, we have included the initial solutions found with DFS and random approaches to our comparison too.

We have considered a small NFC with a k-fat tree architecture. We have assumed an environment with 2 pods and 4 servers, where each pod is connected 2 servers, and carried out our experiments in a machine with an Intel core i7-4500u processor and 8GB of RAM.

Configuration solution timing

We have implemented the ILP formulation of the NFCRMP in CPLEX [28]. We have conducted a set of experiments to compare the time taken to find a solution for new policy requests provisioning by ILP, ILS and GP approaches. We have considered a set of policy requests, which have total

of 10 VNFs. For the ILS process, the total time includes both (1) to find an initial solution using DFS, and (2) the ILS process over 20 repeated procedures. For the GP process, the total time includes both (1) to find an initial solution using DFS, and (2) the GP process over 200 generations. We used 20 repeated procedures for ILS and 200 generations for GP, because we have observed that, after those limits the improvements are very rare for each approach [see Sub-section 4.2.2 and 4.3.2].

ILP took 1.5 hours to provide the optimal solution while ILS took only 2.9 milliseconds and GP took only 3.2 milliseconds. The results confirm that although the ILP formalisation of the problem gives the optimal solution, the ILP computational time requirement makes it not suitable even for a few VNFs. On the other hand, the ILS and GP approach can find solutions in milliseconds.

Configuration solution quality

In the next set of experiments, we compared the quality of the solution for new policy requests provisioning provided by ILP, ILS and GP approaches. As explained in earlier chapters, both the ILS and GP processes rely on either (1) DFS, or (2) a random approach to find the initial solution. We carried out separate ILS and GP processes experiments with both types of initial solutions. We explored different classes of problems where we assume that the 10 VNFs are distributed over one, two or three policies. By varying the capacity requirements of VNFs, we observed that there are different classes of these problems, where the differences are based on number of servers required by these VNFs. We made sure to select 3 cases in which the DFS would not give the optimal solution, because we wanted to explore how the ILS and GP processes improves the solution given by DFS. Specifically: (1) 10 VNFs belong to three policies, but all of them fit onto a single server, (2) 10 VNFs belong to two policies, and they fit onto two servers and (3) 10 VNFs belong to three policies, but they fit onto a single server.

In all 3 sets of experiments, DFS gave better initial solution than random. Therefore, when the DFS solution was given as the input to the ILS and GP processes, in all 3 cases, our ILS and GP algorithms were able to find the optimal solution. As described in previous sections, the DFS method is a good bin-packing strategy, and therefore the solutions given by the DFS uses minimal number of servers required. Also, it introduces less inter-rack traffic, as it tries to place VNFs of a policy in the same server as much as possible. Since the random method selects servers and paths randomly that can be anywhere in the network, it uses much more servers and introduces high inter-rack traffic, as the policy is splitted and VNFs are placed in servers that are in different pods. Because of these reasons, the initial solutions provided by the DFS was better than the random approach. However, as the solution provided by the DFS tended to use fewer links and those links were congested, ILS and GP were able

to improve the solution by using different paths with different links to distribute the traffic and reduce the average link utilization.

4.4.2 Large scaled networks

As shown in the previous section, our results show that finding the optimal solution even for a few NFs with the ILP Equation (4.1) can take hours and is consequently not suitable to meet traffic changes in the NFV context. On the other hand, the heuristics based approaches find acceptable solutions fast. Therefore, for the large scaled networks, we continue our performances comparison with only heuristic based approaches.

Depth First Search (DFS) approach vs Random approach

The ILS and GP approaches both take an initial solution as the input, and tries to improve the given initial solution. As explained in Sections 4.3 and 4.2, we use (1) DFS or (2) a random approach to find the initial solution. Therefore, we can use these DFS and random solutions as the baseline to compare the solutions given by the GP process after 200 generation. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The random method searches servers and paths randomly anywhere in the network, until a feasible configuration is found.

First, we carried out experiments (50 rounds) to compare the quality of the solutions provided by DFS and random approaches for large networks. For each round of experiment, we derived a fixed set of policies (average of 100 policies) that include a total of 400 VNFs and try to find solutions with DFS and random in a 128 server environment in a 4-fat tree architecture network. We assumed that each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units. Similar to the results observed in smaller networks (Section 4.4.1) and for the same reasons, in all the experiment rounds, the initial solutions provided by the DFS was better than the random approach.

ILS approach vs GP approach

As it is proven that the initial solution provided by DFS is better than random, we use the DFS solution as the initial solution (baseline) and compared the performances of the proposed ILS based algorithm with the GP based algorithm for new policy requests provisioning. The comparison included the quality of the solution provided by the each approach as well as the time taken by the each approach to find a solution.

We conducted separate experiments assuming different network architectures: (1) a k-fat tree, (2) a VL2, and (3) a BCube, and for each type of network architecture, we conducted 30 round of experiments. For each

round, we derived a new set of policies that include a total of 400 VNFs and assumed a 128 server environment. First, we tried to find initial solutions with DFS, and then improved the initial solution using ILS process with 20 repeated procedures and GP process with 200 generations separately. We used 20 repeated procedures for ILS and 200 generations for GP, because we have observed that, after those limits the improvements are very rare for each approach [see Sub-section 4.2.2 and 4.3.2].

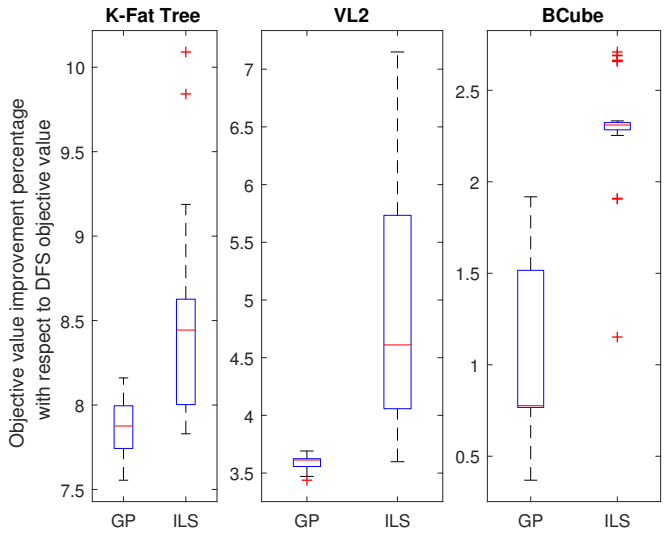

Figure 4.15: Objective value improvements: ILS and GP

As explained in the previous sections, with the ILS algorithm, the standard full local search performs "1-opt-VNF move" to the all the VNFs sequentially as in the new policy requests order, and tries to improve the solution, while the perturbation performs (1-opt-VNF move) randomly to the VNFs. However, we observed that even without performing a full local search, by just performing "1-opt-VNF move" to the 20% of the new policy requests in each local search round, ILS produced better results than GP for all three types of architectures, in all 30 sets of experiments.

Figure 4.15 shows the objective value improvement percentages for ILS and GP separately, with respect to the initial solution given by DFS. The important part is that, most of the quartile boxes do not overlap. As the main goal of our optimization is to reduce the average link utilization, so that the network is less congested and future scaling requirements are minimized, the results produced by ILS reduced the average link utilization by 32%, 27.2% and 7% (compared to the DFS solution) respectively for the three architectures. Moreover, the results produced by GP reduced the average link utilization by 28.7%, 14.9% and 3.2% (compared to the DFS solution) respectively for the three architectures.

In addition, as shown in Table 4.5, for all three architectures, ILS was faster than GP.

Architecture	Time for GP in μs		Time for ILS in μs	
	Average	SD	Average	SD
KFat	167742.6	4245.47	71369.46	4032.97
VL2	153589.8	2541.53	65920.46	4010.66
BCube	173704.56	4672.85	105443.2	22798.8

Table 4.5: Time comparison: ILS and GP

Chapter 5

VNFs scaling methods

Despite some initial efforts [15, 16] the dynamic scaling of VNFs presents still many open challenges. For example, one of the challenges is deciding the scaling method: i.e., whether to use vertical scaling (i.e., allocation/release of host and bandwidth resources to/from a VNF instance) or migration (i.e., running VNFs are paused, serialized and transferred to different servers with more resources) or horizontal scaling (i.e., installation/removal of VNF instances). A second challenge is how to resolve potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration and therefore current network activities are minimally disturbed, and at the same time optimize usage of server and network resources [17].

As described Chapter 4, for our research work in dynamic resource allocation, to achieve scaling, we adopted to a simple scaling procedure: first we tried to allocate resources for a vertical scaling, and if it failed, then we tried to allocate resources for migration scaling. However, it is important to explore more on each scaling method to decide which scaling method is suitable for which situation. Therefore, we explored different scaling approaches in depth, specifically how the optimization is effected by the scaling approach and the optimization objectives.

We expanded our ILS algorithms with three different scaling methods: vertical, migration and horizontal to compare their performances. We considered a single optimization goal: maximize the accepted bandwidth of scaling requests while ensuring a new constraint: that the delay experienced by each packet of an accepted scaling request (inside the NFC), does not exceed its requested deadline. We compared the different characteristics of the solutions provided by scaling approaches such as accepted bandwidth ratio, resource utilization etc.

In this chapter, we present a discussion on following aspects:

1. Practical aspects of different scaling approaches: Section 5.1

2. Optimization aspects of different scaling approaches: Section 5.2
3. Implementation of the ILS based resource allocation algorithm: 5.2.1
4. A comparison between different scaling approaches: Sub-section 5.2.3
5. A comparison between different optimization objectives: Sub-section 5.2.3
6. A summary of the evaluation: Sub-section 5.2.4

5.1 Scaling approaches for VNFs: practical aspects

Vertical, horizontal and migration based scaling approaches are widely adopted resolutions in the implementation of VM based dynamic resource allocation. There are two different aspects to look at when deciding which approach to use: (1) selecting an approach based on the optimization goals and (2) selecting an approach considering the real implementation. First of all, we present a general discussion on these scaling approaches in-terms of the real implementation [17].

Vertical scaling is a basic feature of a VNF which adjusts logical partition of multiple metrics (e.g. CPU, Memory, Bandwidth, I/O etc.) in a VNF. Dynamic Logical Partition Resizing (DLPAR) allows users to logically attach and detach resource to and from a logical partition's division without rebooting. On the other hand, horizontal scaling changes the number of VNF instances, which involves running applications on two or more separate VNFs hosted on same or different servers. Further, live migration of VNF instances can be employed to scale bandwidth and host resources. The running VNFs are paused, serialized and transferred to a different server with more resources, where they are once again scheduled for execution with additional resources. Following sub sections describe the differences of scaling approaches in-terms of different characteristics.

Physical limitations

The physical limitation of vertical scaling is the spare computational resources of the VNF's current server. When the computational resource demand of a VNF exceeds this limitation, vertical scaling will fail and has to change to the live migration approach where the VNF has to be transferred to a different server with more resources. However, even for the live migration approach, the limitation of the spare computational resources of the server applies, because the maximum computational resource that can be allocated to the VNF is limited by the maximum computational resource capacity of the server. On the other hand, for the horizontal scaling, within fine-grained VNFs, the physical limitation is the overall spare

resources inside the cloud infrastructure. Theoretically, cloud provider can offer as many VNFs as possible in response to clients' demands.

Time for the reconfiguration

The vertical scaling approach is efficient with respect to the time for reconfiguration. The DLPAR technology used in vertical scaling takes less than 1 second for reconfigurations on pHyp hypervisorors and VMWare ESX3s [130]. However, the duration of reconfiguration for horizontal scaling and migration is higher, ranging from tens of milliseconds to a second, because it includes the time to copy the new instance in a new server and then initialize the instance [54, 55]. The process of coping the instance involves VNF migration and time cost of migration is higher than the time cost of metrics adjustment.

Cost of software license

Horizontal scaling introduces an additional cost due to the additional software licenses needed for the new VNF instances. However, vertical scaling and migration do not need additional software licences, as they do not change the number of VNF instances.

Complexity of server consolidation

In general, resource fragments after a long term scaling requires the CSP to perform server consolidation. The server consolidation is an approach to the efficient usage of computer server resources in order to reduce the total number of servers used [131]. The practice of server consolidation was developed in response to the problem of server sprawl, a situation in which multiple, under-utilized servers take up more space and consume more resources than can be justified by their workload. The consolidation after horizontal scaling is much more complicated, because the fragments caused by it are irregular.

Coordination

Vertical scaling and migration, allows CSP to allocate a single VNF for the clients total traffic with no coordination or latency among multiple VNF instances. However, horizontal scaling introduces new VNF instances and therefore, the CSP must ensure synchronization between VNF instances. Moreover, horizontal scaling needs a gateway and workload balancer that distributes the traffic to multiple VNF instances, which makes it resource-consuming, that provides more throughout at expense of complexity.

5.2 Scaling approaches for VNFs: optimization aspects

As mentioned earlier, the second aspect to look at when deciding which scaling approach to use is, selecting an approach based on the optimization goals. We considered a single optimization goal: maximize the accepted bandwidth of scaling requests while ensuring a new constraint: that the delay experienced by each packet of an accepted scaling request (inside the NFC), does not exceed its requested deadline.

We assumed that the end-to-end delay experienced by each packet inside the NFC, is composed of following two components: (1) processing delay of each VNF in the policy request and (2) transmission delays between VNFs of the policy request [15]. Even though we have limited the end-to-end delay to these two components, and considered only the delay inside the NFC, one can easily add more delay components such as: (1) the extra delay caused by moving NFs to the cloud, (2) the delay caused by creating and configuring new VNF instances, (3) the delay caused by migrating traffic from one VNF instance to another etc.

In this section, we describe the process that was carried out to compare the effect of different scaling approaches (vertical, migration and horizontal) with respect to the resource optimization, when handling scaling events over a day.

5.2.1 Resource allocation algorithm

We used the same ILS based resource re-allocation algorithm described in Chapter 4.2 to explore three scaling approaches: (1) vertical, (2) migration and (3) horizontal. The Resource Manager uses the current state and search for the re-assignment of resources (servers and paths) for all the VNFs of the scaling policy. The ILS based resource re-allocation algorithm follows the same procedure as described in Chapter 4.2, but with two differences. First is the way the search is performed for resource re-allocation, which is based on the scaling approach that is used. Second is the optimization goal of the Resource Manager, that is to maximize the accepted bandwidth of scaling requests and ensure that delay experienced by each packet of accepted scaling requests, do not exceed their required deadlines.

For the vertical scaling, the scaling concept is to increase/decrease server resources to/from the server where the VNF currently resides, and increase/decrease bandwidth resources to/from the paths that VNF currently uses. When a policy has to be scaled out, for each VNF of the scaling policy, the Resource Manager checks whether the server and the path, that is currently used by the VNF, can handle the total bandwidth demand, while ensuring the traffic of the policy request meet its deadline. If the total bandwidth demand of the scaling request can't be satisfy, then

the Resource Manager tries to allocate resources from the current server and the path, to accept as much as possible bandwidth demand, while ensuring the deadline. When a policy has to be scaled in, for each VNF of the scaling policy, the Resource Manager decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses.

For the migration scaling, the concept is to transfer the VNF to a different server with more resources that can handle the total bandwidth demand. When a policy has to be scaled out, we assumed it as a new policy. Also, we assumed that the new policy's bandwidth request is the total bandwidth demand: (current bandwidth +/- change). For each VNF of the scaling policy, the Resource Manager searches for a server and a path to handle the total bandwidth demand, using a DFS method, while ensuring the traffic of the policy request meet its deadline. If the Resource Manager can't find resources to handle the total bandwidth demand of a scaling request, then the Resource Manager tries to find resources to accept as much as possible bandwidth demand, while ensuring the deadline. After the new policy has been implemented, the resources allocated to the existing policy are removed. This is done because, it is expected that both implementation would need to co-exist for sometime to be able to handle session affinity to ensure the correct behaviour of the VNF. When a policy has to be scaled in, for each VNF of the scaling policy, the Resource Manager decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses.

For the horizontal scaling, the concept is to install/remove VNF instances or paths to handle the extra bandwidth demand (the traffic change). Therefore, when a policy has to be scaled out, we assumed it as a new policy. Also, we assumed that the new policy's bandwidth request is the extra bandwidth demand (the traffic change) of the existing policy. We called it the "child policy". For each VNF of the policy that is scaling, the Resource Manager searches for a server and a path to handle the extra bandwidth demand, using a DFS method, while ensuring the traffic of the policy request meet its deadline. If the Resource Manager can't find resources to fully satisfy the extra bandwidth demand of a scaling request, then the Resource Manager tries to find resources to accept as much as possible bandwidth demand, while ensuring the deadline. When a policy has to be scaled in, for each VNF of the scaling policy, the Resource Manager decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses. If this decrease means removing total resources from a child policy, then we remove that child policy entirely.

5.2.2 Experimental set-up

We modified the experimental setup as follows: we assumed that we are going to provide services for policy requests of 6 large enterprises. There-

NF type	Processing capacity	CPU cores required
IP firewall	900 Mbps	4
Application firewall	900 Mbps	4
WAN optimizer	900 Mbps	4
Proxy	900 Mbps	4
Gateway	900 Mbps	4
VPN	900 Mbps	2
Load balancer	900 Mbps	4
IDS/ IPS	600 Mbps	8

Table 5.1: Processing capacities and CPU requirements of VNFs

fore, the system handled around 130 policies with 600 NFs. We assumed a 128 server environment in a 4-fat tree architecture where each server has an initial capacity of 24 cpu cores and each link has an initial capacity of 10 Gbps.

We considered 7 types of VNFs: IP firewalls, Application firewalls, WAN optimizers, Proxies, Gateways, VPNs, Load balancers, and IDS/IPS as described in Section 3.4. Furthermore, we made more precise specifications of the VNFs. The computational requirements for each VNF to serve a specific amount of traffic, can be derived using the inbound traffic rate and the resource profile of each VNF type. A resource profile for each NF includes: (1) processing capacity of the VNF, (2) required minimum number of CPU cores and (3) bandwidth demand transformation by the VNF (compress or amplify the traffic).

Following the existing works [41, 97], Table 5.1 shows the assumed processing capacities and the required number of CPU cores for different VNFs in our experiments.

The bandwidth demand transformations are associated with traffic-scaling VNFs. For an example, certain VNFs, such as Redundant eliminators and Caches compress traffic, while other NFs such as packet multiplication and encryptions amplify the traffic. Following the existing works [132], Table 5.2 shows the assumed traffic scaling factors for different VNFs in our experiments. A traffic scaling factor less than 1 implies traffic compression while a traffic scaling factor greater than 1 implies traffic amplification.

As mentioned in the previous Chapter, since we do not explore the problem of scaling triggering in this work, we used the scaling requirement patterns for a day presented in [22] to replicate the traffic changes over a day. However, for VNFs scaling methods evaluation, we considered traffic changes of every hour (previously we considered the traffic of every 30 minutes), and we assumed the scaling triggers and derived the scaling requirements. Since we are using more precise specifications of VNFs for the VNFs scaling methods evaluation, the consequences of the traffic

NF type	Traffic scaling	Traffic scaling factor
IP firewall	No	-
Application firewall	No	-
WAN optimizer	Yes	0.65
Proxy	Yes	0.65
Gateway	No	-
VPN	Yes	1.5
Load balancer	No	-
IDS/ IPS	Yes	1.5

Table 5.2: Bandwidth demand transformations of VNFs

changes are different now. We call the scaling requirements of each hour as a "scaling event" and we have 20 scaling events for the full day.

Following the work in [133], we assumed that the end-to-end delay requirement (deadline) for a packet within a CSP network is 1 ms. In our algorithm model, we have assumed that the end-to-end delay experienced by each packet (inside the NFC) is composed of: (1) processing delays of VNFs in the policy request and (2) transmission delays between VNFs of the policy request [15]. We assumed that the processing delay of a VNF is equal to $\frac{1}{cap}$, where cap is the processing capacity of that VNF [15]. The transmission delay between two VNFs is calculated based on allocated path, where the delay of each link in the path is assumed to be 0.001 ms [134].

5.2.3 Evaluation

In this section, we will describe the results of the experiments that were carried out to compare the effect of different scaling approaches. All experiments were carried out in a machine with an Intel core i3 processor and 20GB of RAM.

For each experiment round, first we started with the initial resources allocation algorithm, to find initial allocations for policies for their initial expected traffic. We ran the algorithm with the two different methods of finding the initial solution: (1) DFS and (2) random. Then we took the solutions of two methods (DFS and random), and use them separately to run the resource allocation algorithm for scaling. We ran the algorithm with three scaling methods (vertical, migration and horizontal) separately, over the 20 scaling events of a full day.

Therefore, we can summarize the different approach as follows:

1. Initial allocation with DFS and scaling with vertical: (DFS-Vertical)
2. Initial allocation with DFS and scaling with migration: (DFS-Migration)

3. Initial allocation with DFS and scaling with horizontal: (DFS-Horizontal)
4. Initial allocation with Random and scaling with vertical: (Random-Vertical)
5. Initial allocation with Random and scaling with migration: (Random-Migration)
6. Initial allocation with Random and scaling with horizontal: (Random-Horizontal)

We repeated this process for 10 experiment rounds, where for each experiment round, we derived a fixed set of policies (average of 130 policies) that include a total of 600 VNFs, as described in Section 5.2.2. For each experiment round, scaling events were derived based on the policy set used and the traffic data, following the process described in 5.2.2.

Percentage of bandwidth dropped

The optimization goal of our algorithm was to maximize the accepted bandwidth of scaling requests with a constraint that ensures the delay experienced by each packet of accepted scaling requests, do not exceed their deadlines. Therefore, we checked the percentage of bandwidth dropped, for each scaling approach, starting with DFS and random based initial allocation to have a better comparison. There are two reasons for an amount of bandwidth or total bandwidth from a scaling request to be get dropped: (1) the cloud infrastructure does not have enough resources or (2) the cloud infrastructure has enough resources, but the resources cannot satisfy the deadline constraint. We calculated the Percentage of bandwidth dropped for each event as (Total of not accepted bandwidth for all the scaling policies of the event / Total bandwidth requested by all the scaling policies of the event * 100). Vertical scaling had the highest percentage of bandwidth dropped: average of 49.6%. The next was migration scaling: average of 3.89%, while horizontal scaling had the lowest percentage of bandwidth dropped: average of 0.12%, making it the best scaling approach, in terms of the optimization goal of accepting maximum bandwidth as possible.

Figure 5.1 shows the percentage of bandwidth dropped for the migration scaling, over scaling events of the day. Vertical scaling is always limited by the spare computational resources of the VNF's current server because vertical scaling is done in the same server that the VNF resided at that time. However, starting with a random initial allocation and continuing the day with vertical scaling, provided significant better results than starting with a DFS initial allocation. The DFS followed a bin-packing strategy and did not leave free CPU units in servers to be used for future bandwidth demands. On the other hand, with random initial allocation, servers were not fully packed, therefore there were CPU units left in servers to be used for future bandwidth demands.

Figure 5.2 shows the percentage of bandwidth dropped for the migration scaling, over scaling events of the day. With the migration scaling, when

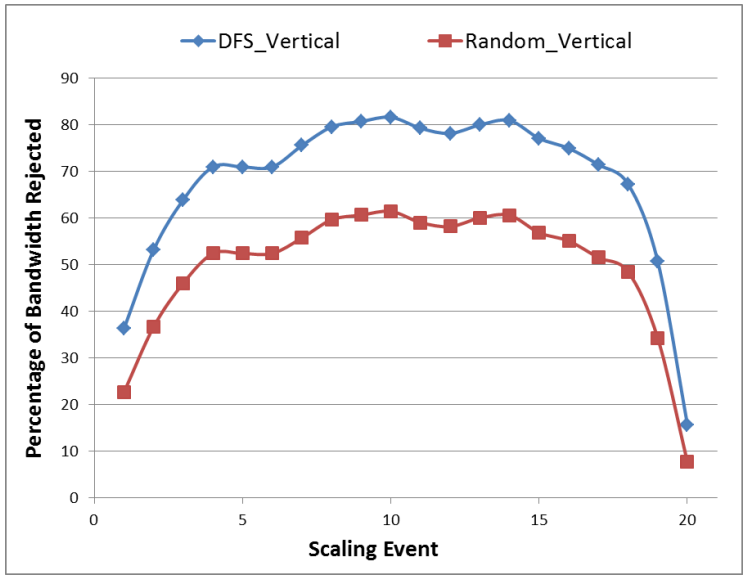

Figure 5.1: Percentage of bandwidth dropped: Vertical scaling

a policy has to be scaled out, we assumed it as a new policy. Also, we assumed that the new policy's bandwidth request is the total bandwidth demand of the existing policy. The first observation was, when the system is not fully tight (more than 30% of the resources are free), the Resource Manager was able to find resources to satisfy the total bandwidth demand. But when the system became tighter over time, the Resource Manager was not able to find resources to satisfy the total bandwidth demand. The second observation was, when the bandwidth demand increased over time, the migration scaling faced the problem of physical resources limitation of a server, because the maximum computational resource that can be allocated to the VNF is limited by the maximum computational resource capacity of the server. Therefore, the Resource Manager was not able to find servers to satisfy the total bandwidth demand. The final observation was, for the migration scaling, for the scaling events at the beginning of the day (until 8th event), there was no significant difference between starting with a DFS or a random initial allocation. However, when the system gets tighter over the time, percentage of bandwidth dropped tended to be slightly different for two methods. For some events, starting with DFS initial allocation gave better results while for other events, starting with random initial allocation gave better results. Therefore, starting with a DFS or a random based initial allocation, did not provide significant different effect to continuation with the migration scaling. The factors effected the continuation with migration scaling were tightness of the system and physical resources limitation of servers.

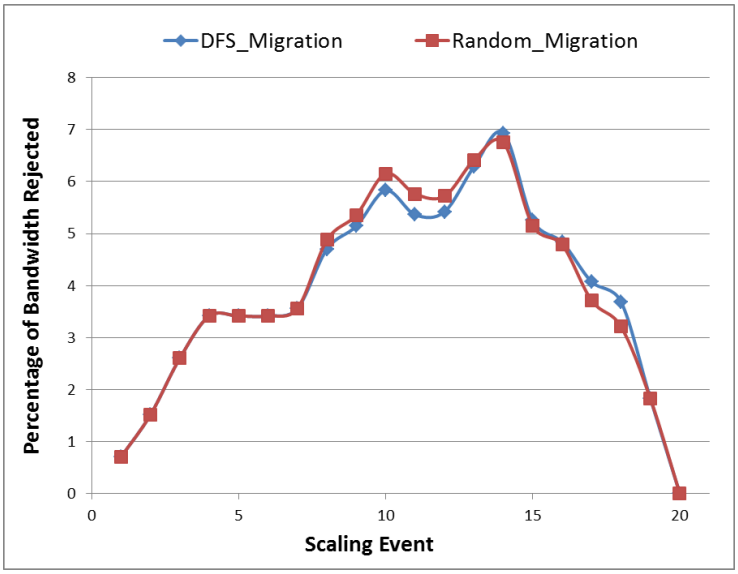

Figure 5.2: Percentage of bandwidth dropped: Migration scaling

Figure 5.3 shows the percentage of bandwidth dropped for the migration scaling, over scaling events of the day. With the horizontal scaling, when a policy has to be scaled out, we assumed it as a new policy. Also, we assumed that the new policy's bandwidth request is the extra bandwidth demand (the traffic change) of the existing policy. The main observation was, most of the time (when the system is not tight or tight), the Resource Manager was able to find resources to satisfy the total extra demand. Therefore, the percentage of dropped bandwidth was minimum. For the scaling events at the beginning of the day (until 8th event), there was no significant difference between starting with a DFS or a random initial allocation. However, when the system gets tighter over the time, starting with DFS initial allocation gave better or equal results, having a low percentage of bandwidth dropped.

Allocated CPU units

The optimization goal of our algorithm was to maximize the accepted bandwidth of scaling requests. We can equate that value with earning and contrast that against the CPU units allocated for the VNFs to process the accepted bandwidth as a measurement of expenses. This implies an approximated measurement of profit. [2] The number of allocated CPU

[2]Even though we have not explicitly converted the accepted bandwidth requests or allocated CPU units to monetary values, once they are converted, it is easy to calculate the profit.

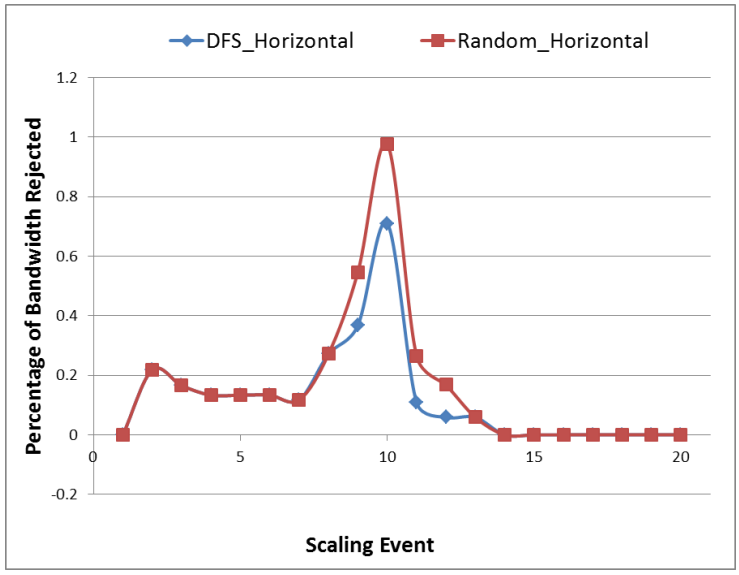

Figure 5.3: Percentage of bandwidth dropped: Horizontal scaling

units for the VNFs to process the accepted bandwidth when following different approaches are shown in Figure 5.4. We compared the allocated CPU units by these approaches with a base line: the number of CPU units required to satisfy the total bandwidth demand of the system (for all policies) at each scaling event. This is calculated without considering the maximum capacities of servers or links, therefore the baseline is the best case, which is not practically implementable.

As the vertical scaling has the highest percentage of dropped bandwidth, it has the lowest number of CPU units allocated. The migration scaling's number of CPU units allocated is closer to the baseline, having the number of allocated CPU units closer to the baseline. According to the baseline, the horizontal scaling has allocated CPU units more than required. With the horizontal scaling, when a policy has to be scaled out, we assumed it as a new policy. Also, we assumed that the new policy's bandwidth request is the extra bandwidth demand (the traffic change) of the existing policy. We have observed that some VNFs in the existing policy, had the enough resources to satisfy the total bandwidth demand, while some VNFs in the existing policy did not. As the maximum processing capacity of a VNF for a given amount of resources depends on the VNF type, the capability to handle the total bandwith demand with already allocated resources also depends on the VNF type. A strategy of re-using VNFs of the existing policy that could satisfy the total bandwidth is complicated. We might need to create new instances for some VNFs of the policy that can't satisfy the total bandwidth, and distribute the traffic by having workload balancers

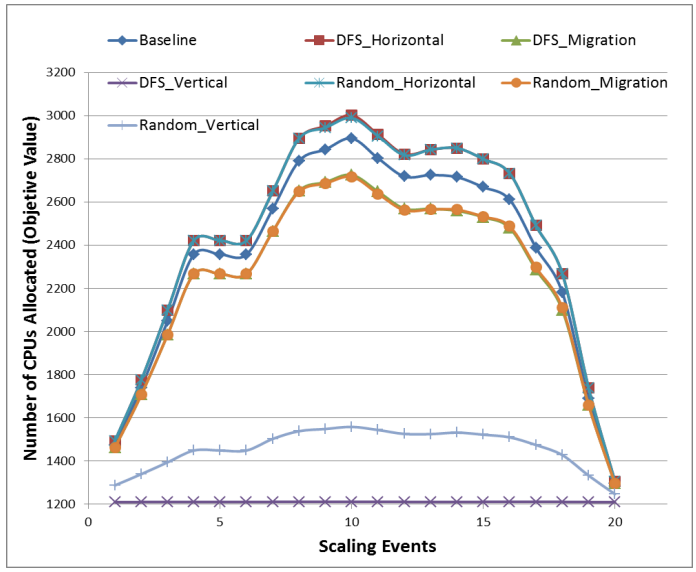

Figure 5.4: Comparison of allocated CPU units

within the policy chain. Therefore, even though there were VNFs in the existing policy that could satisfy the total bandwidth demand with the already implemented VNF instance, we still created a new VNF instance, because the concept was to implement a new policy to satisfy the extra bandwidth demand, as it needs a single workload balancer to distribute the traffic over policy instances. For both the migration scaling and horizontal scaling, starting with a DFS or random initial solution did not provide significant difference in the results.

Server utilization

As shown in the previous sections, the horizontal approach has allocated more CPUs and accepted more bandwidth than migration and vertical approaches. However, we have observed that to allocate more CPUs and to accept more bandwidth, horizontal approach used more and more servers. Therefore, one can consider this scenario as a situation with conflicted objectives: whether to use more server resources and accept more bandwidth requests or to use less server resources and accept less bandwidth requests.

Figure 5.5 shows the comparison of number of servers used by each approach (average for 10 experiment rounds) over the scaling events. For the scaling events at the beginning of the day, when started with a random based initial allocation, all three scaling approaches have used more servers. Further more, for the scaling events until the middle of the day, when started with a DFS based initial allocation, both migration and

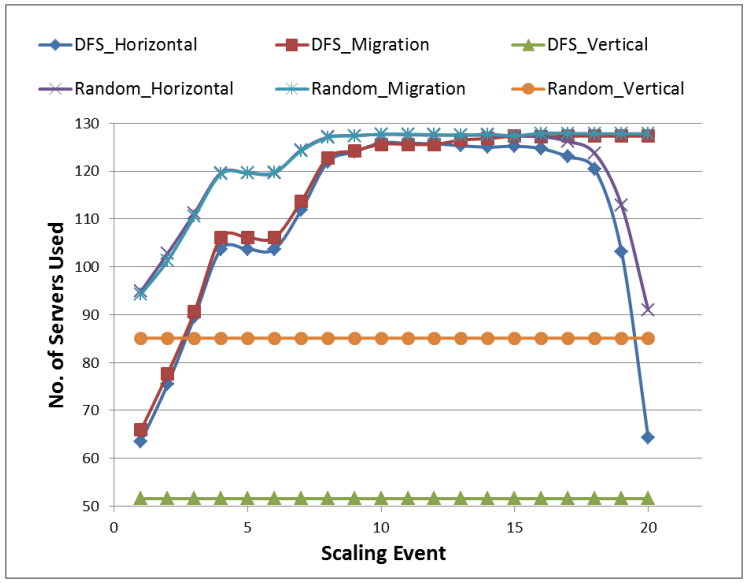

Figure 5.5: Comparison of number of servers used

horizontal scaling approaches have used similar number of servers. Also, when both of them started with a random based approach, they have used similar number of servers too.

However, at the end of the day, when the total traffic rate is getting decreased and policies had to scaled in, the horizontal scaling approach used fewer servers. This is because when scaling in (as a result of traffic decrease), horizontal scaling was able to remove the child policies that were created, as the result of scaling out to accommodate the extra traffic demands (when traffic was increasing). On the other hand, for migration approach, scaling out was achieved by treating the policy as a new policy with total traffic demand, and transferring it to a new server with more resources. Therefore, when scaling in, for migration approach, the process was just to decreases server resources from the server where the VNF currently resides, and decreases bandwidth resources from the paths that VNF currently uses. It did not change the placement of VNFs and therefore, the number of servers used did not change.

Changes to the current configuration

As we mentioned earlier, when scaling VNFs to satisfy the new traffic amount, there might be different conflicted objectives. When re-allocating resources to satisfy bandwidth requests as much as possible, we might not want to cause many changes to current configuration, specifically current placement of the VNFs, so that the current network activities are mini-

mally disturbed.

We use the term "a server change", to refer when the placement of a VNF changed, to satisfy the scaling requirement. Figure 5.6 shows the comparison of number of server changes, when satisfying the scaling requirements using horizontal and migration scaling approaches (average for 10 experiment rounds) over the scaling events. For the migration scale, "a server change" means the existing VNF instance has to migrated to a new server to satisfy the scaling request and now the traffic has to re-directed to the new server. For the horizontal scale, "a server change" means, that a new VNF instance has to instantiate in a new server to satisfy the scaling request and now the traffic has to be balanced between multiple VNF instances. When using vertical scale, there are no server changes, because for the vertical scaling, scaling is done by adjusting CPU and memory metrics in the server, that the VNF is currently residing. Therefore, we have not included vertical scaling in the Figure.

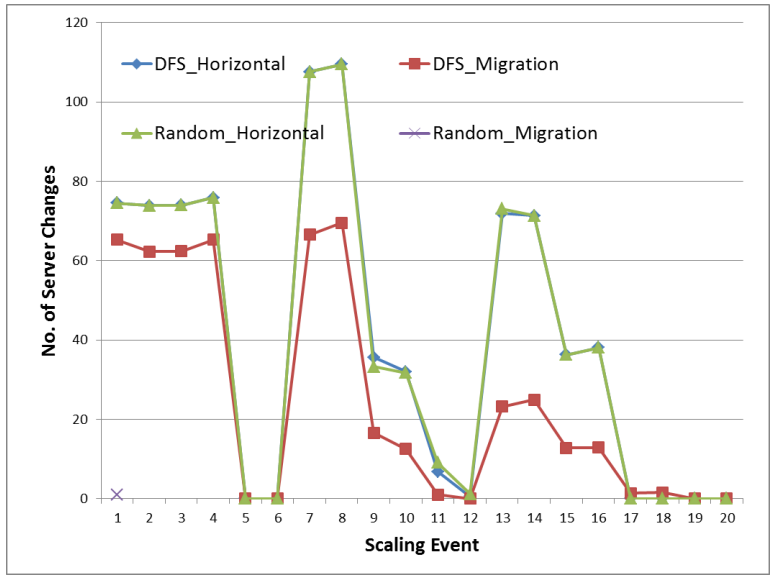

Figure 5.6: Number of server changes

The main observation is that, for both horizontal and migration scaling approaches, there was no significant effect from starting with a DFS based or a random based initial allocation, with respect to the number of server changes. Furthermore, horizontal scaling caused more server changes than migration scaling. This is because horizontal scaling was accepting more bandwidth requests more than the migration scaling, at the cost of changing servers. However, the consequence is that during the system adjustments, the horizontal scaling might cause more traffic loss, directly affecting the earnings and profit.

5.2.4 Summary

As mentioned in Chapter 2, the existing research work on scaling of VNFs is very limited, most of them focus only on the complexities of the scaling technologies. They do not consider the resource allocation or optimization aspects of these scaling technologies. Therefore, we explored different scaling approaches and the optimization perspectives: vertical scaling (allocation/release of computing and bandwidth resources to/from a VNF instance), (2) migration (running VNFs are paused, serialized and transferred to different servers with more resources) and (3) horizontal (installation/removal of VNF instances).

We conducted experiments to check how the optimization is effected by the scaling approach and the optimization objectives. We implemented a resource allocation algorithm to allocate resources for scaling requirements using ILS approach, with the three scaling methods. We considered a single optimization goal: maximize the accepted bandwidth of scaling requests while ensuring a new constraint: that the delay experienced by each packet of an accepted scaling request (inside the NFC), does not exceed its relative deadline. We compared the different characteristics of the solutions provided by scaling approaches such as accepted bandwidth ratio, resource utilization etc.

The Vertical scaling approach had the highest percentage of bandwidth dropped: average of 49.6%, and therefore accepted lowest number of scaling requests. The next was migration scaling approach, with and average of 3.89% of bandwidth dropped. The horizontal scaling approach had the lowest percentage of bandwidth dropped: average of 0.12%, and accepted highest number of scaling requests. Therefore, it can be considered as the best scaling approach, in terms of the optimization goal of accepting maximum bandwidth requests as possible. Our observation is that vertical scaling is always limited by the spare computational resources of the VNF's current server, and therefore, vertical scaling is rarely able to satisfy the scaling requirements. The Migration scaling and horizontal scaling have more freedom in scaling, and able to satisfy most of the scaling requirements.

It is important to note that, each time a VNF has been placed in a new server, paths have to be created dynamically to re-direct the traffic to the new placement of the VNF. This involves modifying routing rules in the switches: in our SDN network, changing OpenFlow rules in the switches dynamically.

With migration scaling, the concept is to transfer the VNF to a different server with more resources to satisfy the total bandwidth request. Therefore, when using migration scaling, the rules of the switches are modified to re-direct the total traffic flow, to the new placement: to the new server that the VNF was migrated. But, when using horizontal scaling, as the concept is to create a new VNF instance to satisfy the extra bandwidth request, the total traffic has to be balanced between multiple instances. Therefore,

the NFC Management System has to use a load balancing mechanism and decide how the load is balanced between multiple instances of the VNF, and then modify the rules in the switches to direct the traffic to each VNF instance accordingly.

Chapter 6

Dynamic load balancing

Traditionally CSPs have been adopting dynamic load balancing approaches to reduce the risk of a single server (or VM) being overloaded. They try to distributed the traffic load across multiple servers (or VMs). In the similar manner, there are situations where CSPs should adopt dynamic load balancing mechanisms for VNFs, to balance the load among multiple VNF instances. For an example, when using horizontal scaling as the scaling approach, VNF instances are added/removed to satisfy the scaling requirement. However, when new VNF instances are added/removed, the traffic has to be balanced among the remaining VNF instances dynamically.

Most of the current load balancing algorithms for cloud are flow-level based, where a traffic flow is a uni-directional sequence of packets sent from a source to a destination. However, with VNFs, the load balancing approaches should look beyond the uni-directional flows and take care of the sessions: which is bi-directional traffic flows between two nodes. This is because, unlike layer 3 forwarding, many VNFs such as firewall, proxy, and VPN perform stateful packet processing: session based packet processing [29]. Therefore, these VNFs require affinity, where traffic for a given flow must reach the instance that holds that flow's state [30]. In such cases, splitting traffic to balance the load, requires extra measures to preserve affinity. For an instance, consider a load balancing scenario where a firewall instance is overloaded with traffic and an additional instance needs to be instantiated to adapt to the workload. The CSP must not only direct some traffic to the new instance, but also move internal flow states associated with the traffic. If not, serious problems might arise; for example, attacks may go undetected because the new VNF instance does not have the necessary information.

Existing solutions that maintain affinity mostly depend on state migration techniques: moving the relevant state from one instance to another. Frameworks like Split/Merge [119] and OpenNF [30] facilitate fine-grained transfers of internal NF state to support fast and safe reallocation of flows

across NF instances. However, these systems require that NF vendors adopt a new programming model or add a non-trivial amount of code to existing NF implementations. Therefore, it is more appropriate to design the load balancing algorithm in a way that the algorithm itself maintain affinity and handle states and sessions, so that the requirement of state migration is avoided [120].

We have proposed a load balancing algorithm that controls sessions to avoid state migration, using consistent hashing techniques. We have built a prototype to explore the proposed load balancing algorithm, and the Load Balancer is deployed as a VNF in the network. In this chapter, we discuss the proposed load balancing algorithm in following aspects:

1. Overview of the consistent hashing concept: Section 6.1
2. Description of the load balancing algorithm: Section 6.2
3. Experimental set-up: Section 6.3
4. Preliminary evaluation: Section 6.4

Furthermore, for the dynamic load balancing, it is important that the network configuration of the cloud infrastructure should be able to be updated dynamically, easily and fast. However, dynamic update of network configuration introduces a new challenge to the CSPs, because when updating the network configuration, the CSPs should try to avoid any inconsistencies in transient traffic and to minimize traffic lost, so that the SLAs are not violated. In this context, we believe that use of Software Defined Network (SDN) infrastructure is a very appropriate approach, as it allows reconfiguring the physical network easily.

Therefore, In this chapter we discuss two dynamic network configurations update mechanisms found in existing works [31]: Section 6.5

6.1 Consistent hashing

Consistent hashing is a scheme that provides hash table functionality to distribute data across a cluster, in a way that the addition or removal of one node does not significantly change the mapping of keys to nodes. In this Section we introduce consistent hashing terminologies and describe the consistent hashing concept.

Why consistent hashing

Consistent hashing was introduced in 1997 as a way of distributing requests among a changing population of web servers [135]. In the context of the Web, imagine a browser requesting a URL. Of course, one could request the page from the appropriate Web server, but if the page is being requested over and over again, it is wasteful to repeatedly download it from the

server. The best idea was to use a Web cache, which stores a local copy of recently visited pages. When a URL is requested, first the local cache is checked for the page. If the page is in the cache, the page is sent directly to the browser; without contacting the original server. If the page is not in the cache, then the page is downloaded from a suitable server as before, and the result is both sent to the browser and also stored in the local cache for future re-use.

The main challenge with caching was where to store the cache. The earliest method was to give each end user their own cache, maintained on their own machine or device, but, later it was realized that if we could implement a Web cache that is shared by many users, for example, all users of campus network, it would be more beneficial. However, remembering recently accessed Web pages of many users might take a lot of storage. Thus, implementing a shared cache at a large scale required spreading the cache over multiple machines. When the cache is spread over multiple machines (multiple caches), and whenever a browser requests a URL and if we want to know whether the Web page has been cached, it was not obvious where to look for a cached copy of the Web page. We could poll all caches for a copy, but that was inefficient and time consuming. It created the requirement of mapping from URLs to caches. The first solution was to use a hash function: a hash function maps objects, like URLs, to "buckets", like caches. However, a simple hash function was not enough for when the number of buckets is not static, but rather is changing over time. Therefore, the exact requirement was *a hash table-type functionality (to store objects and retrieve them later) with the additional property that almost all objects stay assigned to the same bucket even as the number of buckets changes. And the solution was to introduce consistent hashing* [135].

A "good" hash function h should satisfy two properties [135]:

1. It is easy to remember and evaluate, and ideally, computing the function involves just a few arithmetic operations, such as a "mod" operation
2. For all practical purposes, h behaves like a totally random function, spreading objects out evenly and without noticeable correlation across the possible buckets

In addition to these proprieties, a consistent hash function should ensure that neither addition nor removal of buckets cause a total objects to buckets remapping [135]. When a new bucket is added, the objects of its neighbours are shared with it, and when removal takes place, the removed bucket's objects are shared back with its neighbours.

Overview of consistent hashing for load balancing

The initial usage of consistent hashing was distributing requests among a changing population of web servers. However, we can use this concept and

distribute the traffic flows over multiple VNFs instances, in a way that the traffic load processed by VNF instances are balanced.

The key idea of consistent caching is, in addition to hashing the names of all objects x (for example Traffic Flows: TFs), we also hash the names of all the buckets s (for example VNF instances). The object and bucket names need to be hashed to the same range, such as 32-bit values [135]. This approach to consistent hashing can also be visualized on a circle, with points on the circle corresponding to the possible hash values. Buckets and objects both hash to points on this circle; an object is stored on the bucket that is closest in the clockwise direction. Thus, n buckets partition the circle into n segments, with each bucket responsible for all objects in one of these segments.

To have a better understanding of the consistent hashing concepts, lets assume a simple scenario where we have to distribute the traffic flows over multiple VNFs instances. According to the consistent hashing terminologies, Traffic Flows (TFs) are considered as objects and VNF instances are considered as buckets [136].

Figure 6.1-left shows the first step of the consistent hashing. We assume that the range of hash value is (0, 2^{32}-1), and we assume there are 3 VNF instances (3 buckets). We calculate the hash value of each VNF instance (i.e., using IP address and port) and put them into the hash ring. Suppose we have 4 TFs (4 objects) and as the second step, we calculate the hash value of each TF and put them into the hash ring as shown in Figure 6.1-right. According to the consistent hash algorithm, the object TF_A will be located in VNF_1, the TF_D will be located in VNF_3 and the object TF_B, TF_C will be located in VNF_2.

As shown in Figure 6.2-left, assume that the VNF_3 goes down, and the VNF_1, and VNF_2 are still available. In a such situation, with a general hashing algorithm, locations of all objects has to be re-arranged. However, with consistent hashing, only few objects will be re-located. Therefore, only the TF_D will be re-located to VNF_2.

Assume that a new VNF instance: VNF_4 is added to the network as shown in Figure 6.2-right. Then only the TF_B will re-located to the VNF_4.

The challenge with consistent hashing algorithm is, since it is essentially random, it is possible to have a very non-uniform distribution of TFs between VNF instances. For an example, as shown in Figure 6.3-left, it is obvious that the VNF_1 will have to process most TFs and the performance of the hash ring will be decreased.

The solution to this problem is to introduce the idea of "virtual nodes", which are replicas of VNF instances points in the circle. So whenever we add a VNF instance we create a number of points in the circle for it. For the example shown in Figure 6.3-right, where we have 2 VNF instances, we create 3 virtual nodes for each VNF instance: $VNF_{1.1}$, $VNF_{1.2}$, $VNF_{1.3}$, etc. and we will get 6 virtual nodes. The hash ring with virtual nodes is

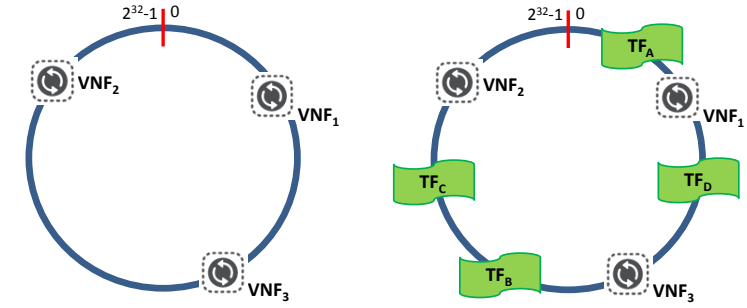

Figure 6.1: Left: Hashing of each VNF instance, Right: Hashing of each TF

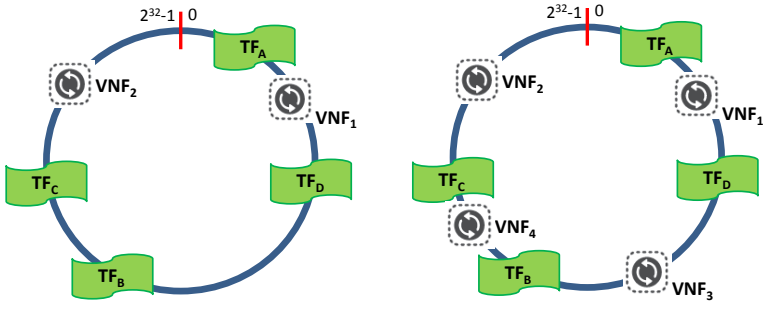

Figure 6.2: Left: An existing VNF instance is removed, Right: A new VNF instance is added

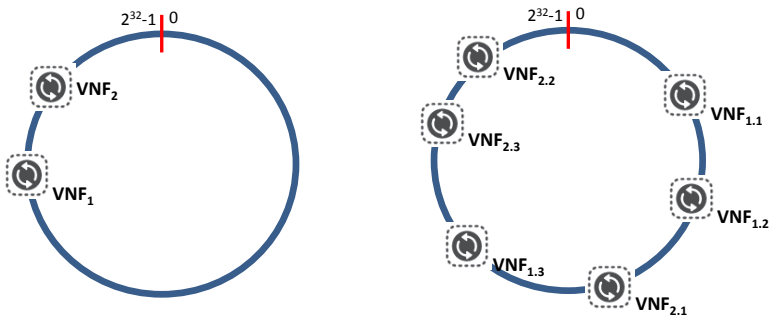

Figure 6.3: Left: An example of non-uniform distribution of TFs between VNF instances, Right: The solution of adding virtual nodes

shown in Figure. All the TFs located to $VNF_{1.1}$, $VNF_{1.2}$ and $VNF_{1.3}$ will be allocated to VNF_1. And it solved the non-uniform distribution of objects between caches.

6.2 Consistent hashing based load balancing

We have proposed a load balancing algorithm using consistent hashing techniques. The algorithm distributes the traffic flows over multiple VNFs instances, in a way that the traffic load processed by VNF instances are balanced. The algorithm itself has been designed to maintain flow affinity and handle states and sessions, so that the requirement of state migration is avoided. In this section we introduce our **Load Balancer** module and the implemented load balancing algorithm.

Whenever a packet is received by the Load Balancer, the Load Balancer should decide, to which VNF instance that this packet will be forwarded to. The Load Balancer should ensure that it maintains the session affinity while balancing the load between VNF instances fairly and efficiently. Therefore, if the packet belongs to an existing active session, the Load Balancer should forward the packet, to the VNF instance that is currently processing the specific session. If the packet does not belong to an existing active session, then the Load Balancer should select a VNF instance to forward the packet.

There are three main functions in our consistent hashing based load balancing algorithm: (1) Initialising the hash function, (2) Session-aware hashing and (3) Dynamic load allocation.

First of all, to initialize the data structures and functions in the algorithm, the Load Balancer module execute the initialising the hash function step. Next, when Load Balancer receives a packet, it decides to which VNF instance that this packet will be forwarded to. For this process, the Load Balancer execute the Session-aware hashing function. Furthermore, the Load Balancer module should support dynamic changes of the network, where the Load Balancer might have to re-distribute the traffic load because: (1) traffic changed over time, (2) a new VNF instances was added, or (3) an existing VNF instance was removed.

In the following Sub-sections, first we describe the terminologies we used in the load balancing algorithm and then we describe each of the main functions of the load balancing algorithm in detail. In addition, we describe the proposed operational model for our load balancing algorithm, where the Load Balancer module is implemented as a VNF in the network.

Notations and terminologies

We assumed a scenario where one instance of a specific type of VNF is not enough to handle the traffic, therefore we have to deploy x number of instances from that VNF type. The Load Balancer has to distribute the traffic to these x VNF instances fairly and efficiently. When distributing the traffic, the Load Balancer has to maintain flow affinity and handle states and sessions of flows. Hence, in our load balancing algorithm, we focused on distributing the traffic in-terms of the sessions.

Following the consistent hashing terminologies, we considered sessions of the traffic flow as our objects, which we want to distribute over the buckets, which are the VNF instances. Therefore, each VNF instance nf has:

1. An ID
2. Slots (virtual nodes).

The Load Monitor monitors traffic going through each VNF instance nf in a given time window tau and the Load Distributor uses historical traffic statistics (from the load monitor) to decide the number of slots assigned to each VNF instance nf.

We assume that a "session" ses has two main properties:

1. A session key
2. A session status

A session key is used to identify the session and it consists of an {inbound IP, inbound port, outbound IP, outbound port}. Given a packet p, if

it is an incoming traffic, then its session key is {srcIP, srcPort, destIP, destPort}. If it is an outgoing traffic, then its session key is {destIP, destPort, srcIP, srcPort}.

A session status is used to identify whether the session is currently active or not and it consists of: (1) the last (refresh/update) time-stamp and (2) a tag. The last (refresh/update) time-stamp is the time of the last seen packet for this session. The tag is the VNF instance ID, to which the traffic of this session will be forwarded to.

Given the time window tau, a session ses is active, only if:

$$now - ses.lastTimestamp < tau \tag{6.1}$$

Furthermore, a packet p is in the existing active session ses, if and only if:

$$\ni ses.[key(ses) == key(p) \quad \text{and} \quad p.timestamp - ses.lastTimestamp < tau] \tag{6.2}$$

The consistent hashing function $hash()$ maps each given packet p to a VNF instance nf id, such that, it respects:

1. Session: inbound and outbound traffic for the same session ses should go through the same VNF instance nf
2. Slots: the probability of assigning a session key to an VNF instance nf is equal to:
   (Slots Of VNF instance nf / Total Slots Of All VNF instances)

Initialising the hash function: hash()

As the initialization step, first, the Load balancer initialize the hash ring that is used for the hashing function in the session-aware hashing algorithm. Given a random number generator seed s and a list of VNF instances $nf_1, nf_2, ..., nf_N$, it generates an array $buckets$ as follows.

As mentioned earlier, each VNF instance nf has an ID and n number of slots assigned to it. Therefore, for each nf_i, the Load balancer adds the $nf_i.ID$ to the array $buckets$ for $nf_i.slots$ times. Hence, size of the array $buckets$ is equals to the total number of slots of all VNF instances. Then the Load balancer shuffle the array $buckets$ using the random seed s.

Session-aware hashing

After the initialization step, next we describe the session-aware hashing function, which is used by the Load Balancer to decide to which VNF instance that a packet will be forwarded to.

Whenever a packet is received by the Load Balancer, the Load Balancer should decide, to which VNF instance that this packet will be forwarded

to. The Load Balancer should ensure that it maintains the session affinity while balancing the load between VNF instances fairly and efficiently. Therefore, if the packet belongs to an existing active session, the Load Balancer should forward the packet, to the VNF instance that is currently processing the specific session. If the packet does not belong to an existing active session, then the Load Balancer should select a VNF instance to forward the packet.

As shown in the Algorithm 3, the Load Balancer starts the process for each packet by, first checking whether the packet is in an already active session using the Key of the packet: {inbound IP, inbound port, outbound IP, outbound port}. If yes, then the Load Balancer uses the *activeSessions*() function to retrieve information about the Session that this packet belongs to. The *activeSessions*() function maps a Key to the Session. Therefore, we pass the Key of the packet: {inbound IP, inbound port, outbound IP, outbound port} to the *activeSessions*() function, and retrieve the Session information. The Session information contains: (1) the last (refresh/update) time-stamp and (2) the tag. As we mentioned earlier, the tag is the VNF instance ID, to which the traffic of this session is currently being forwarded to. The Load Balancer uses the tag and identify the VNF instance that this packet should be forwarded to. Furthermore, the Load Balancer also modify the last time-stamp of the session to reflect the current time as the last time-stamp. Finally, the Load Balancer returns the ID of the selected VNF instance as the output.

If the packet is not in an already active session, then the Load Balancer should decide, to which VNF instance that this packet will be forwarded to. For that, the Load Balancer uses the consistent hashing function. We use a simple hashing function *hash*(), which takes the Key of a packet as the input and returns a random integer in $[0, 2^{32}]$ as the result. Furthermore, the result of the *hash*() function is forwarded to the modulo operator with the size of the *buckets* array. Finally, the position of the *buckets* array pointed by this modulo operation, defines the VNF instance that this packet will be forwarded to. Note that the *buckets* array is constructed with VNF instances IDs according to the number of slots that each VNF instance is assigned. Since the packet is not in an already active session, this packet is assumed to be the first packet of a new Session. Therefore, an entry for a new Session is created, with current time as the new Session's last time-stamp and the selected VNF instance ID as the new Session's tag. Furthermore, the new Session's status is saved as active. Finally, the Load Balancer returns the ID of the selected VNF instance as the output.

Dynamic load allocation

As mentioned earlier, the Load Balancer should support dynamic changes of the network, and therefore the Load Balancer might have to re-distribute the traffic load due to three reasons:

1. The traffic load changed over time

Algorithm 3 Session-aware hashing

```
1: procedure HASH(P: PACKET)
2: IF p is in an active session THEN:
3:     sesStatus ← activeSessions[key(p)]
4:     sesStatus.lastTimestamp ← p.timestamp
5:     RETURN sesStatus.tag
6: ELSE:
7:     InstanceID ← buckets[hash(key(p))%buckets.size]
8:     newSesStatus.lastTimestamp ← p.timestamp
9:     newSesStauts.tag ← InstanceID
10:    activeSessions[key(p)] ← newSesStatus
11:    RETURN InstanceID
12: END IF
```

2. A new VNF instances was added
3. An existing VNF instance was removed

First of all we will introduce the definitions used in dynamic load allocation procedure.

For the past-time window tau, let $t_1, t_2, ..., t_N$ be the traffic statistics: total incoming and outgoing packets going through each VNF instance nf_i. Thereofre for all N VNF instances, the total network traffic be:

$$T = sum(t_1, t_2, ..., t_N)$$

Let $l_1, l_2, ..., l_N$ be the number of slots assigned each VNF instance nf_i, and the total slots be:

$$L = sum(l_1, l_2, ..., l_N)$$

Therefore, we can calculate the probability p_i of the hash function to assign a (random) session to the VNF instance nf_i as follows:

$$p_i = l_i/L \tag{6.3}$$

In a perfect world where the all sessions are uniformly distributed, we would have:

$$T.p_i = t_i \tag{6.4}$$

However, since some sessions may have many more packets than others, we introduce the session bias bi for each VNF instance nf_i, such that,

$$T.b_i.p_i = t_i$$

Therefore, the session bias bi for each VNF instance nf_i can be calculated as:

$$b_i = t_i/(T.p_i) \tag{6.5}$$

Now we can derive the equations to re-allocate traffic load for previously mentioned three situations.

First, to simply redistribute the traffic load whenever traffic changed, we can compute new values for $p_1, p_2, ..., p_N$ as $p_1^{new}, p_2^{new}, ..., p_N^{new}$.

In ideal situation, for each VNF instance nf_i,

$$T.b_i.p_i^{new} = T/N$$

Therefore,

$$p_i^{new} = 1/(N.b_i)$$

$$p_i^{new} = (T/N).(p_i/t_i) \tag{6.6}$$

Second, in a situation where a new VNF instance nf_{N+1} is added, we modify p_i^{new} for $i <= N$ as,

$$p_i^{new} = (T/(N+1)).(p_i/t_i)$$

and,

$$p_{N+1}^{new} = 1/(N+1) \tag{6.7}$$

Third, in a situation where a VNF instance nf_k is removed, we modify p_i^{new} for $i! = k$ as,

$$p_i^{new} = (T/(N-1)).(p_i/t_i)$$

and,

$$p_k^{new} = 0 \tag{6.8}$$

As we have updated values for $p_1^{new}, p_2^{new}, ..., p_N^{new}$, as the final step, we can compute $l_1, l_2, ..., l_N$, assuming L is fixed.

Operational model of the dynamic load balancing

The main requirement of load balancing comes with the horizontal scaling approach, where the policies are scaled by adding/removing VNF instances. Since now there are multiple VNF instances to handle traffic, the traffic load has to be balanced between the existing VNF instances. We propose a model where the load balancing algorithm, with the Load Balancer module runs as a VNF in the network. We specifically focus on scenarios where the load balancing has to be done after horizontal scaling of policies. In this Sub-section, we describe the operational aspect of our load balancing model.

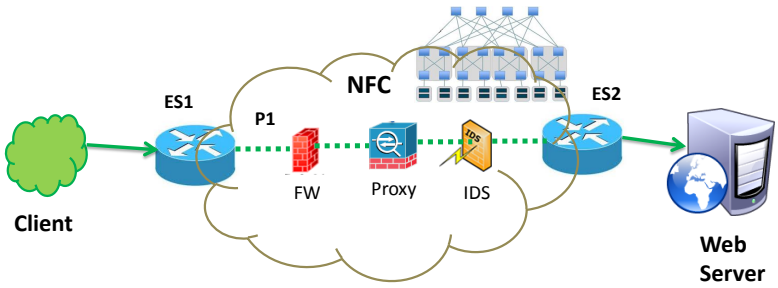

Figure 6.4: Implementation of the original policy $P1$

Let's assume a scenario where the NFC had a client that requests his web traffic to go through a chain of VNFs: firewall-proxy-IDS. As shown in Figure 6.4, to satisfy the client policy request, we have already implemented policy $P1$: a VNF chain of firewall-proxy-IDS in the NFC. The client traffic enters to the NFC through the edge switch $ES1$, traverse through the VNF chain inside the NFC. Then the traffic goes out of the NFC through the edge switch $ES2$ and reach the destination Web server. The replies from Web server, enters to the NFC through the edge switch $ES2$, traverse through the VNF chain inside the NFC. Then the traffic goes out of the NFC through the edge switch $ES1$ and reach the client.

Furthermore, over the time, because of the traffic increase, $P1$ has to be scaled out using horizontal scaling approach. Following the horizontal scaling concept, we treat this scaling requirement as a new policy; where we install new VNF instances of firewall, proxy and IDS for the policy in NFC. As shown in Figure 6.5, we call the newly implemented policy as the child policy $P1_1$.

Once the child policy $P1_1$ is created, it also connected to the edge switches $ES1$ and $ES2$. The client traffic that comes to the $ES1$ must be distributed over the policies $P1$ and $P1_1$ to traverse through the required VNFs chain. Therefore, the Load Balancer module has to take the decisions on load balancing and split the traffic accordingly at the $ES1$.

However, in our scenario there are two types of traffic flows, going towards two opposite directions: (1) traffic from client to web server (web requests)

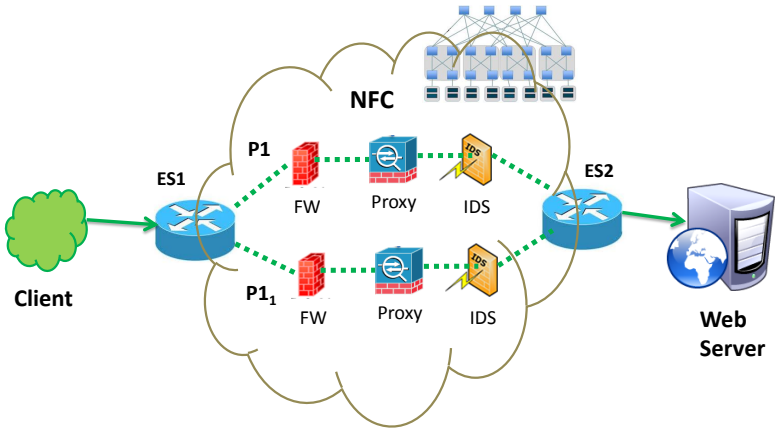

Figure 6.5: Implementation of the child policy $P1_1$ to satisfy the scaling requirement

and (2) traffic from web server to client (replies). In a such situation, the Load Balancer should maintain flow affinity and handle states and sessions considering both directions of the traffic flow: requests and corresponding replies. Therefore, the Load Balancer module must ensure that, if a traffic session went from client to web server through the VNFs of $P1$, then the reply for that traffic session also should go through the VNFs of $P1$. In the same way, if a traffic session went from client to web server through the VNFs of $P1_1$, then the reply for that traffic session also should go through the VNFs of $P1_1$.

To handle traffic in both directions (requests and corresponding replies), we implement a pair of Load Balancer modules edge switch $ES1$ and $ES2$. As shown in Figure 6.6, the two Load Balancer modules $LB1$ and $LB2$ are implemented as two VNFs connected to $ES1$ and $ES2$.

For the pair of Load Balancers $LB1$ and $LB2$, each Load Balancer runs its own consistent hash function and Load Monitor. The Load Balancer connected to edge switch $ES1$ (i.e., $LB1$) acts as the master and the other (i.e., $LB2$) connected to edge switch $ES2$ acts as the slave.

When a new child policy (with new VNF instances) added or an existing child policy is removed, the master Load Balancer is notified by the NFC Management System. Also, when the traffic load has to be re-balanced, the master Load Balancer is notified by the NFC Management System. According to the new information sent by the NFC Management System,

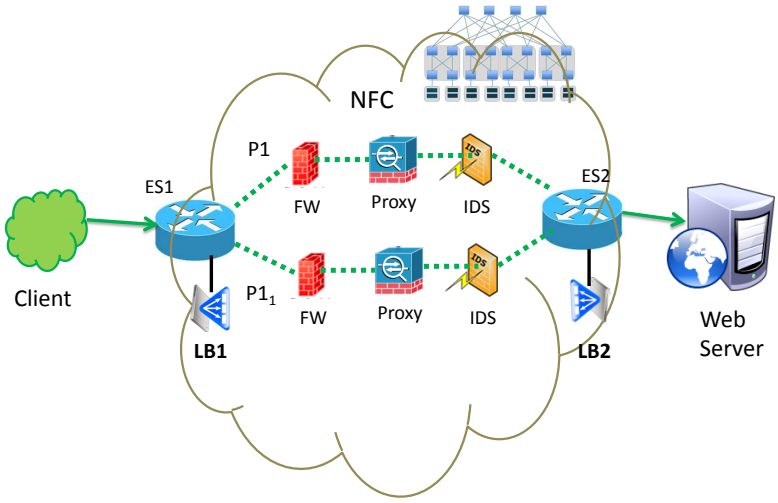

Figure 6.6: Implementation of the Load Balancers

the master Load balancer has to update his hash function (i.e, number of VNF instances and number of slots for each VNF, the traffic assigned to each VNF instance, etc.) Furthermore, the master Load Balancer has to communicate with the slave Load Balancer and send the new information to slave Load Balancer, so that the slave Load Balancer also can update its hash function. Therefore, in order to have a consistent hash function between $LB1$ and $LB2$, information about: (1) currently active sessions, (2) VNF instance IDs and (3) slots assignments are shared and synchronised, through a separate control channel between $LB1$ and $LB2$.

6.3 Experimental Set-up

We have conducted set of preliminary experiments to evaluate the performances of the session-aware load balancing algorithm. We have assumed a SDN based network and implemented the prototype of our network in "mininet" [137]: a SDN simulation environment and used "Ryu" [138] as the SDN controller. The OpenFlow based SDN switches are configured using the "Ryu" controller to send the traffic through the network. Furthermore, we have implemented VNFs as mininet hosts that run specific NF software. Moreover, the Load Balancers are also implemented as VNFs running in mininet hosts, that are connected to OpenFlow switches.

Following Sub-sections describe the experimental set-up with respect to:

1. Implementation of the Load Balancer VNFs

2. Experimental use cases

Implementation of the Load Balancer VNFs

In our prototype implementation, we have assumed that the Load Balancer is a VNF, that can be deployed in anywhere in the network. The functions of the Load Balancer are programmed as software modules, so that the Load Balancer can be implemented using a VM.

There are three main software modules running in our Load Balancer VNF and they are described in details in this Sub-section:

1. Packet Capture module
2. Session-aware consistent hashing algorithm
3. Tag Inserter module

The functions of a Load Balancer VNF starts, when the Load Balancer receives a packet from the OpenFlow switch which it is connected to. For each packet received, the Load Balancer should decide to which VNF instance that the packet will be forwarded to. This is done by the previously described, session-aware consistent hashing algorithm. To maintain the session affinity, the Load Balancer first checks whether the packet is in an already active session using the Key of the packet: {inbound IP, inbound port, outbound IP, outbound port}. Therefore, the Key of the packet is the main input to the session-aware hashing algorithm. To retrieve the Key of each packet coming to the Load Balancer VNF, we have developed a separate module called **Packet Capture** using PcapPlusPlus [139]. PcapPlusPlus is a multi-platform C++ based network sniffing, packet parsing and manipulation framework.

We are running the Packet Capture module and session-aware hashing algorithm together in the Load Balancer VNFs as follows:

1. For each received packet, we retrieve the Key of the packet: {inbound IP, inbound port, outbound IP, outbound port}, using the Packet Capture module
2. Then the Key of the packet is passed to the session-aware hashing algorithm as the input
3. The hashing algorithm uses the Key of the packet and decides to which VNF instance that this packet will be forwarded to
4. Finally, the hashing algorithm returns the ID of the VNF instance that this packet should be forwarded to

When hashing algorithm returns the ID of the VNF instance that this packet should be forwarded to, the Load Balancer VNF can use that ID to forward the packet to the correct VNF instance. For that, the packet

forwarding rules of the network can be defined considering the ID of the VNF instance, that the packet should be forwarded to. In this situation, we take the advantage of SDN network and its technologies, where we can define packet forwarding rules in OpenFlow switches, based on different properties of the packet [122].

We create packet forwarding rules in OpenFlow switches, based on the packet's {source IP, destination IP and VLAN tag}. We insert a VLAN tag to the packet, where the VLAN tag is equals to the ID of the VNF instance that the packet should be forwarded to. We have developed another module called **Tag Inserter** using PcapPlusPlus [139], which intercepts the packet and insert the VLAN tag. Once the tag is inserted by the Tag Inserter the packet is sent back to the OpenFlow switch which is connected to the VNF Load Balancer. The OpenFlow switch follows the rules that were defined based on the packet's {source IP, destination IP and VLAN tag} and forwards the packet to the correct VNF instance.

A flow chart for the full process of a Load Balancer VNF is shown in Figure 6.7.

Experimental use cases

In this Sub-section we introduce the use cases that we used to evaluate our Load Balancer, specifically its session-aware hashing algorithm.

As shown in Figure 6.8, we have assumed a simple scenario, where the NFC had a client that requests his web traffic to go through a VNF. As shown in Figure 6.4, to satisfy the client policy request, we have already implemented the VNF in the NFC. The client traffic enters to the NFC through the edge switch $S1$, traverse through the VNF inside the NFC. Then the traffic goes out of the NFC through the edge switch $S2$ and reach the destination Web server. The replies from Web server, enters to the NFC through the edge switch $S2$, traverse through the VNF inside the NFC. Then the traffic goes out of the NFC through the edge switch $S1$ and reach the client.

Furthermore, over the time, because of the traffic increase, we assume that the VNF has to be scaled out using horizontal scaling approach. Following the horizontal scaling concept, as shown in Figure 6.9, we install new VNF instance(s) in the NFC. Once the new VNF instance(s) are implemented and paths are created accordingly, the traffic has to be balanced between existing VNF instances. Therefore, we implement the pair of Load Balancers, $LB1$ and $LB2$ as VNFs running in hosts and connected them to the edge switches $S1$ and $S2$. $LB1$ acts as the master Load Balancer while $LB2$ acts as the slave Load Balancer. Furthermore, in order to have a consistent hash function between $LB1$ and $LB2$, and to share and synchronize the required information, we have created a direct control channel between $LB1$ and $LB2$.

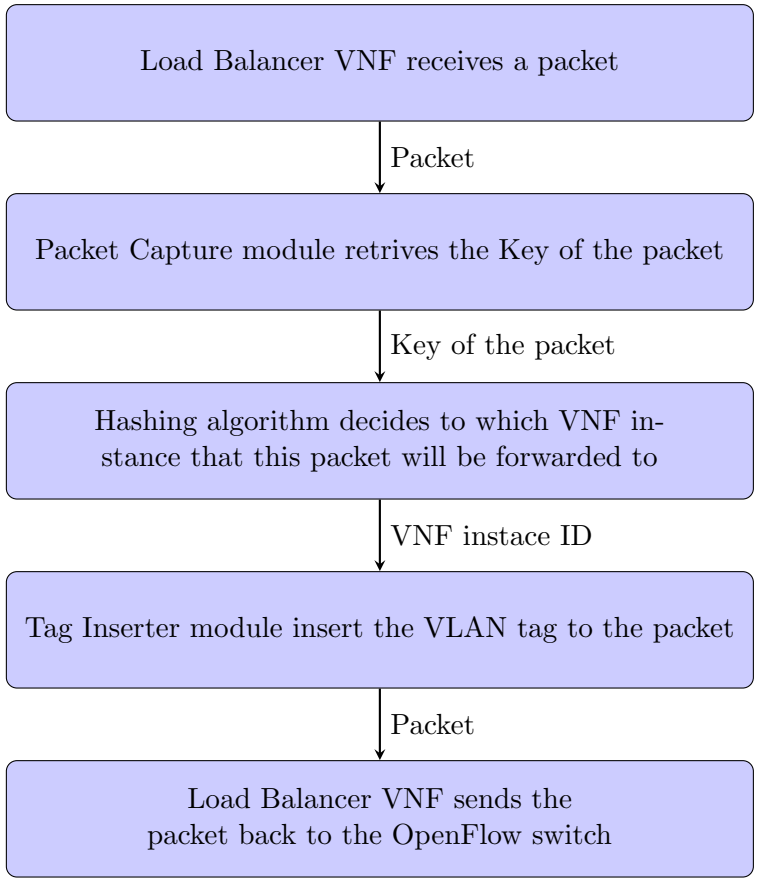

Figure 6.7: Flow chart for the process of a Load Balancer VNF

6.4 Preliminary evaluation

Following the experimental set-up and use cases described in previous sections, we have conducted preliminary experiments to evaluate the session-aware load balancing algorithm for its accuracy and performances. We have considered three scenarios:

1. The system starts with more than one VNF instance, so from the beginning the traffic has to be balanced between multiple VNF instances: **Static situation**
2. The system starts with more than one VNF instance, but one VNF instance is removed dynamically, so the load has to be re-balanced: **Cool down situation**
3. The system starts with only one VNF instance, but one more VNF instance is added dynamically, so the load has to be re-balanced: **Warm up situation**

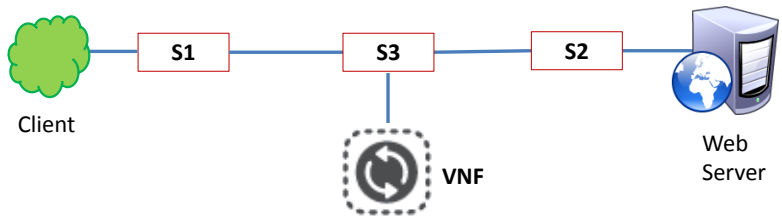

Figure 6.8: A simple network with one VNF instance

For our preliminary experiments, we have assumed that the session time-out is 15 seconds. If the Load Balancer does not receive any packet belongs to a particular session for 15 seconds, the Load Balancer consider that session as a non-active session. Furthermore, the Load Monitor module reports traffic statistics every 1 second, so that the Load Balancer can take decisions on re-distribution of the traffic.

In this section, we present the preliminary results for performances of session-aware load balancing algorithm for the three mentioned scenarios with the assumptions we have made.

Static situation

The first scenario is where we assume that the system starts with more than one VNF instance, so from the beginning the traffic has to be balanced between multiple VNF instances.

To create a such scenario, we assumed that there are two VNF instances: n_1 and n_2, that have been implemented in the system, so from the beginning the traffic has to be balanced between these two VNF instances. We used *iperf* traffic generator [140] and generated 100, 200, 300, 400 and 500 parallel connections. We counted the number of packets received by each VNF instance n_i and calculated the percentage of packets received by each VNF instance n_i (with respect to the total number of packets received by all VNF instances). We conducted 5 experiment runs to get an average value. Figure 6.10 shows the percentage of packets received by

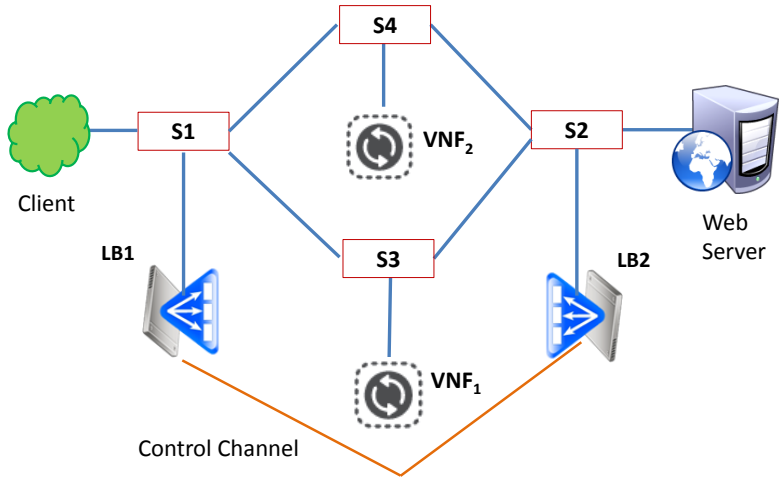

Figure 6.9: Horizontally scaled and Load Balancers are implemented

each VNF instance n_i, when there were 100, 200, 300, 400 and 500 parallel connections. The results confirm that the proposed session-aware hashing algorithm balances load evenly (within 1.5% of ideal).

Furthermore, to confirm the fairness of the algorithm, we assumed that there are three VNF instances: n_1, n_2 and n_3, that have been implemented in the system, so from the beginning the traffic has to be balanced between these three VNF instances. We used *iperf* traffic generator [140] and generated 100 and 300 parallel connections. As mentioned earlier, we counted number of packets received by each VNF instance n_i and calculated the percentage of packets received by each VNF instance n_i (with respect to the total number of packets received by all VNF instances). We conducted 5 experiment runs to get an average value. The results confirm that the proposed session-aware hashing algorithm balances load evenly among three VNF instances (within 1.6% of ideal).

Cool down situation

The second scenario is where we assume that the system starts with more than one VNF instance, but one VNF instance is removed dynamically. So the Load Balancer has to re-balance the load among remaining VNF instances dynamically.

To create a such scenario, we assumed that there are three VNF instances that have been implemented in the system, so from the beginning the traffic is balanced between these three VNF instances. Moreover, in the

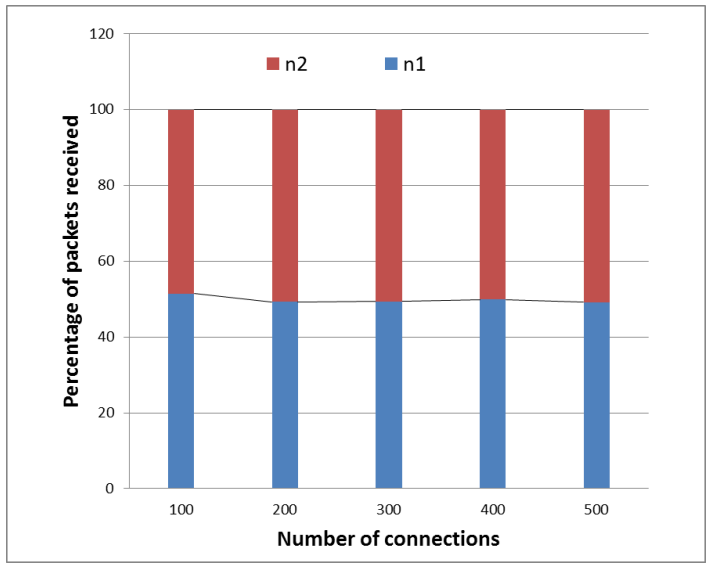

Figure 6.10: Load balance for two VNFs, varying number of parallel connections

middle of the experiment (after 20 seconds from the beginning of the experiment), we removed one VNF instance, so that the Load Balancer had to re-balance the load among remaining two VNF instances dynamically.

When a VNF instance is removed, there are two important aspects to be taken care of. First, even thought the VNF instance is removed, the Load Balancer might be forced to keep forwarding the traffic of a particular session to the removed VNF instance, until the time-out of that particular session expires. This is because of the session aware nature of the Load Balancer and it tries to maintain the session affinity. Second, after the VNF instance is removed, the percentage of the traffic that has been sent to that VNF instance, has to assigned to the remaining VNF instances. The Load Balancer has to consider current traffic loads of remaining VNFs and do the re-distribution of workload fairly. Carrying out this full process and for the system to become stable again, it might take sometime. We called this as the "cool down period" of the removed VNF instance, and in this preliminary evaluation we tried to get a general idea on this cool down period.

To create continuous traffic we used *httperf* traffic load generator [141] and generated 200 connections with a rate of 2 connections per second. From the beginning we assumed that there are three VNFs running: n_1, n_2 and n_3. After 20 seconds from the beginning of the experiment, we removed VNF instance n_3. We used time intervals of 20 seconds to capture the number of packets received by each VNF instance. Then we calculate

the packet rate (throughput) of each VNF instance for each of those time intervals as number of packets for time interval T divided by 20. The average results of 5 experiment runs are shown in Figure 6.11.

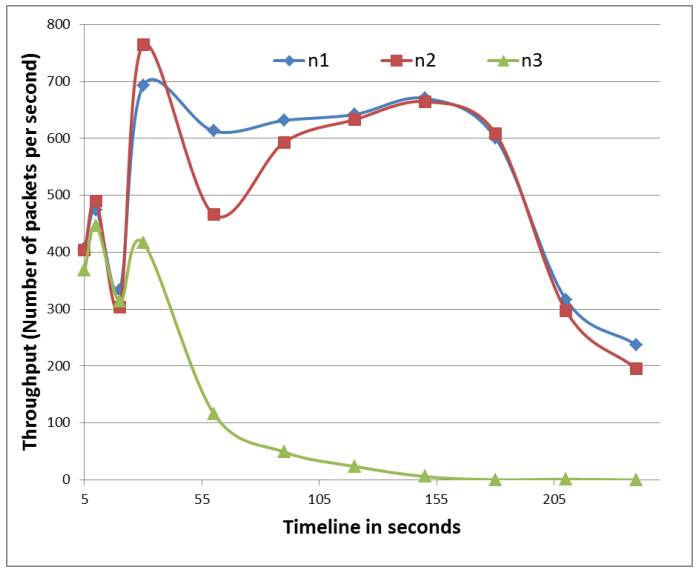

Figure 6.11: Re-balance of the load for removal of one VNF instance

As it is shown in Figure 6.11, until 20 seconds, the load is balanced between three VNF instances fairly. Once the n_3 is removed at 20 seconds, the load is re-distributed over n_1 and n_2. Around 120 seconds, the balance of the load between n_1 and n_2 became fair and stable. However, because of the session affinity, until 150 seconds, there were some packets forwarded to n_3.

Warm up situation

The third scenario is where we assume that the system starts with only one VNF instance, but one more VNF instance is added dynamically. So the Load Balancer has to re-balance the load among VNF instances including the new VNF instance dynamically.

To create a such scenario, we assumed that there is only one VNF instance is implemented in the system at the beginning, so there is no requirement for the load balancing at the beginning. Moreover, in the middle of the experiment (after 20 seconds from the beginning of the experiment), we added one VNF instance, so that the Load Balancer had to re-balance the load among two VNF instances dynamically.

When a VNF instance is added, the Load Balancer has to split the traffic between more than one instance. However, splitting traffic requires extra

measures to preserve affinity. Therefore, even thought a new VNF instance is added, the Load Balancer might be forced to keep forwarding the traffic of a particular session to the old VNF instance until the time-out of that particular session expires. Therefore, it might take sometime for the traffic load between VNF instances to become fair and stable. We called this as the "warm up period" of the new VNF instance, and in this preliminary evaluation we tried to get a general idea on this warm up period.

To create continuous traffic we used *httperf* traffic load generator and generated 200 connections with a rate of 2 connections per second. From the beginning we assumed that there is only one VNFs is running: n_1. After 20 seconds from the beginning of the experiment, we added VNF instance n_3. We used time intervals of 20 seconds to capture the number of packets received by each VNF instance. Then we calculate the packet rate (throughput) of each VNF instance for each of those time intervals as number of packets for time interval T divided by 20. The average results of 5 experiment runs are shown in Figure 6.12.

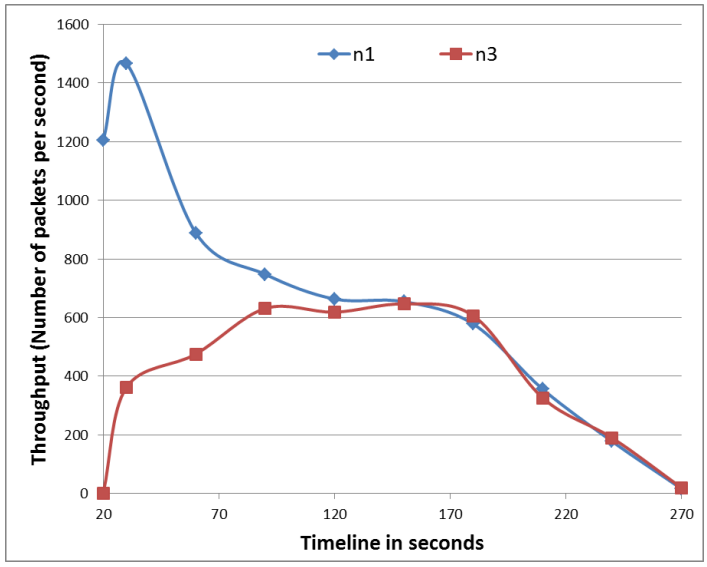

Figure 6.12: Re-balance of the load for addition of one VNF instance

As it is shown in Figure 6.12, until 20 seconds, the load is handled solely by n_1. Once the n_3 is added at 20 seconds, the load is re-distributed over n_1 and n_3. Around 150 seconds, the balance of the load between n_1 and n_3 became fair and stable.

6.5 Dynamic network configurations update

In previous sections, with dynamic load balancing concepts, we explored how to re-distribute the traffic load between VNF instances over the time. However, for the dynamic load balancing process, it is important that the network configuration of the cloud infrastructure, should be able to be updated easily and fast [18], to reflect new packet forwarding rules based on the new traffic distribution.

Therefore, dynamic update of network configuration introduces a new challenge to the CSPs, because when updating the network configuration, the CSPs should try to avoid any inconsistencies in transient traffic and to minimize traffic lost, so that the SLAs are not violated. In this context, we believe that use of Software Defined Network (SDN) infrastructure is a very appropriate approach, as it allows reconfigure the physical network easily.

In this section we discuss two dynamic network configurations update mechanisms found in existing works [31]:

1. configuration rules are updated in all switches simultaneously
2. versioning tags are used to maintain per-flow consistency

We have implemented the physical structure of the NFC in a SDN test bed, which was deployed with five TP-Link TL-WR1043ND switches. OpenWRT and OpenFlow v1.3 was installed to make them SDN compatible switches [122]. As an extension to the physical SDN, two OpenFlow software switches were connected. The SDN test bed was tested with four different SDN controllers: OpenFlow [122], POX [142], Ryu [138] and FloodLight [143]. Apart from the physical network, mininet simulation environment [137] was also used for the research activities. We have used bro firewalls [144] and iptables [145] as the potential VNFs that can be offered as a service in the NFC.

The first method is a simple approach, where configuration rules are updated in all switches simultaneously. Even though this is a simple mechanism, this can introduce inconsistencies in transient traffic and traffic lost. For an example, lets assume a specific traffic flow entered to the network and switches at the entry perimeters forwarded the traffic flow, based on their current rules. However, when this specific traffic flow was going through the network, the configuration rules were updated in all switches simultaneously. Now the traffic flow which is going through the network and currently in the middle and exit perimeters of the network might be dropped by the switches in the middle and exit perimeters of the network, because those switches might not have appropriate rules for these traffic flows. The rules relevant to these traffic flow might have been already deleted because the configuration rules were already updated.

The second method: per-flow consistency update mechanism was introduced in [31] to avoid any inconsistencies in transient traffic and reduce

traffic lost. The update mechanism works by stamping every incoming packet with a version number and modifying every configuration so that it only processes packets with a set version number. To change from one configuration to next, it first populates the switches in the middle of the network with new configurations guarded by the next version number. Once that is completed, it enables the new configurations by installing rules at the perimeter of the network that stamp packets with the next version number. This method makes network updates faster and cheaper, by limiting the number of rules or switches affected.

We implemented both methods in the physical test-bed and evaluate the methods for the number of packet lost. We conducted 25 experiment runs for each method. As shown in Figure 6.13, the first implementation (simultaneous updates) caused an average of 7% packet loss while second implementation (with per-flow consistency) reduced it to average of 1.5%.

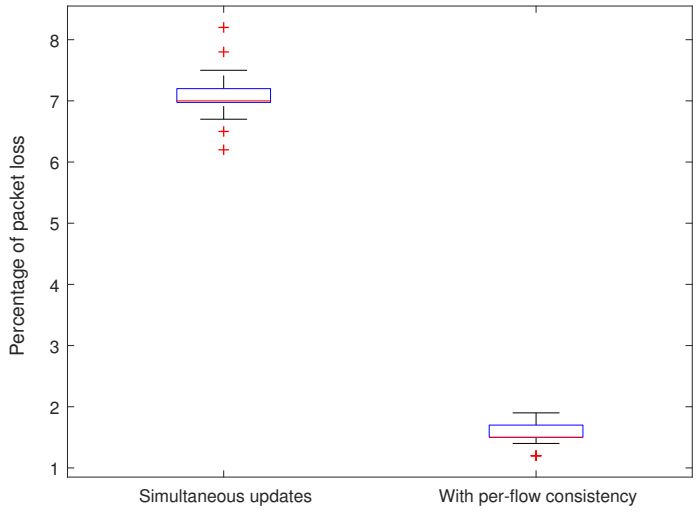

Figure 6.13: Percentage of packet loss

Chapter 7

Final Remarks

> "The outcome of any serious research can only be to make two questions grow where only one grew before."
>
> *Thorstein Veblen (1908)*

The concluding chapter of the thesis aims to present a summary of the work presented in the dissertation, along with the key results and conclusions. Furthermore, it discusses the possible directions for future work, which can be opportunities and guidances for new research works.

7.1 Conclusions and summary

In this section we discuss summary, key results and conclusions from the thesis, organized based on the chapters of the thesis. Broadly, the research aimed to explore different aspects of VNFs management, specifically with cloud infrastructure related operations: provisioning, configuring, maintaining and scaling of the VNFs, as well as configuring and updating of the cloud network.

Starting with Chapter 2, first we presented an overview of the background material necessary for the thesis. It introduced a concrete terminology for NFV concepts and technical concepts useful to understand the thesis work. It then provided an overview of the state of the art for cloud management and offering VNFs as a service.

Before moving to the research aspects of the thesis, Chapter 3 introduced the experimental environment, data-sets and use cases we used to explore different management aspects of VNFs.

We presented our experimental NFV platform: the Network Function Center (NFC) in Section 3.1 and its architecture in Section 3.2. The main

concept of the NFC is a cloud infrastructure and a service provider, that **offerers VNFs as a service** to clients on a subscription basis. The overall architecture of a NFC consists of two main components: a physical infrastructure, and a management system for the infrastructure. We assumed general data center architectures (k fat tree [24], VL2 [26] and BCube [25]) and a SDN based network for the physical infrastructure of the NFC. Section 3.3 described the NFC Management System. The process NFC Management System was built around five key modules: (1) Resource Manager, (2) Topology Manager, (3) Flow Manager, (4) Elasticity Manager and (5) Rules Generator. Each module is responsible for different tasks, such as: resource allocation, monitor cloud infrastructure, decide scaling requirements, scale VNFs , updates the network configurations, etc. Conceptually, the Resource Manager, Topology Manager, Elasticity Manager and Flow Manager can be seen as management applications. The Rules Generator as an extension to the SDN network. In this thesis we focused on exploring the Resource Manager and the Rules Generator.

While building the NFC and exploring different management aspects of the NFC, it was important to have more realistic use cases and datasets that can be used for the design and evaluation of the algorithms, with respect to the cloud management. Since there were no publicly available real data sets on different aspects of VNFs (i.e, VNF chains, traffic passing through them, etc.). The Section 3.4 described the use cases and data set generation process, that was necessary for design and evaluation of the algorithms used by the management system. We have used data from previous empirical analyses [6, 27] and made some assumptions to derive the required data. We developed four programs to model the gathered data and generate the required data. All gathered data and data modelling programs are publicly available at [124].

After the introduction to our experimental environment, Chapter 4 to Chapter 6 focused on describing our research contributions; specifically on achieving the objectives of the research.

In Chapter 4, we discussed the resource allocation approaches for VNFs. We described the Resource Manager module of the NFC Management System, which is responsible for the resource allocations (server and network resources), for VNFs in the cloud. We explored two approximation based methods: **Iterated Local Search (ILS)** and **Genetic Programming (GP)** approaches that can find reasonable solutions fast. We used a traditional resource allocation method: **Integer Linear Programming (ILP)** based approach to compare the quality of the solutions provided by the approximation approaches.

In this thesis, we focused on resource allocation for two situations: the initial resource allocation and (2) the resource re-allocation for scaling. Once the client request for a set of VNFs has been accepted, the resource allocation for the initial placement of these VNFs can be done in the order

of minutes, and then the new VNFs can be deployed accordingly. But the resource re-allocation for the scaling requirements of existing VNFs happens during the middle of the operations, where the already deployed VNFs are processing traffic. Therefore, the resource re-allocation for the scaling requirements of existing VNFs is a time critical on-line problem. The solutions have to be given in the order of milliseconds, so that the disturbances and damages to current operations are minimal. Therefore, we specifically explored the resource re-allocation for the scaling requirements of existing VNFs problem in depth, as it is an interesting on-line resource allocation problem. We formulated the **Network Function Center Resource Management Problem (NFCRMP)** as a set of ILP equations. It addresses both (1) the **resource allocation for new VNFs** provisioning, and (2) the **resource re-allocation for the scaling of existing VNFs** to support traffic changes. For *resource allocation for new VNFs*, the goal was to minimize the required resources (i.e., number of servers, number of links, and average link utilization). For *resource re-allocation for scaling of existing VNFs*, the aim was to adjust the resources to satisfy the traffic changes and, at the same time, minimize the number of configuration changes to reduce potential service disruptions, and performance degradation.

We implemented the approximation algorithms using ILS and GP based approaches and compared their solutions quality with the optimal solutions provided by the implementation of ILP formulation of the NFCRMP in CPLEX [28]. For the comparison in a small network, the results showed that generating the optimal solutions takes 1.5 hours with ILP, but only few milliseconds with ILS and GP.

Second, we conducted a comprehensive evaluation of the proposed ILS and GP approaches for large networks assuming different network architectures: (1) k-fat tree [24], (2) BCube [25], and (3) VL2 [26]. We used a more realistic traffic pattern generated based on [27]. As the both ILS and GP process relies on an initial solution, we used simple Depth First Search (DFS) and random approaches to find an initial solution, and considered this solution as the baseline to compare solutions given by ILS and GP approaches. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The random method searches servers and paths randomly anywhere in the network, until a feasible configuration is found. Our results showed that both ILS and GP algorithms can decide server and network allocations for hundreds of policies (around 400 VNFs) in a 128 server environment and find reasonable solutions in milliseconds. Furthermore, ILS produced better results than GP for all three types of architectures. As the main goal of our optimization was to reduce the average link utilization, so that the network is less congested and future scaling requirements are minimized, the results produced by ILS reduced the average link utilization up to 32% and the results produced by GP reduced the average link utilization up to 28.7%.

However, we observed that, for the ILS based resource allocation algorithms to be more efficient, they have to be designed with conditions based on the network architecture. In other words, efficiency of ILS procedure is not agnostics to the network architecture. Therefore, for a general situation where we might need resource allocation algorithms that are agnostics to the network architecture, GP based approach can provide reasonable solutions.

Third, we study the quality of the ILS and GP generated solutions over time. Because both the ILS and GP approaches are approximations, they run the risks of diverging from the optimal solution over time: whenever we need to scale, we adopt a "local approach" algorithm to find solutions. These "local approach" solutions may gradually diverge from the "global approach" solution given that they strive to minimize not only the required resources, but also the number of changes in the network. We, therefore, compared the solutions computed by our proposed ILS and GP approaches, with their respective "global approach" solutions, generated by ignoring the need to minimize the number of changes in the network. Our results showed that, although the "global approach" provides better resource allocations, the solutions require drastic re-arrangements to the current configurations, and therefore it is impractical in real scenarios. In contrast, "local approach" provides reasonable solutions with lesser changes to readjust configurations, and without diverging from the "global approach" solutions over time. Also, the local approach, if we smooth the curve, will follow essentially the same behaviour (modulo a translation in the y axis) that the behaviour of the baseline: "global approach".

In Chapter 5, we explored different scaling approaches for VNFs: (1) **vertical scaling** (allocation/release of computing and bandwidth resources to/from a VNF instance), (2) **migration** (running VNFs are paused, serialized and transferred to different servers with more resources) and (3) **horizontal** (installation/removal of VNF instances). We expanded our ILS algorithms to allocate resources for scaling requirements with the three scaling methods. We considered a single optimization goal: maximize the accepted bandwidth of scaling requests while ensuring a new constraint: that the delay experienced by each packet of an accepted scaling request (inside the NFC), does not exceed its relative deadline. We compared the different characteristics of the solutions provided by scaling approaches such as accepted bandwidth ration, resource utilization etc.

The Vertical scaling approach had the highest percentage of bandwidth rejection: average of 49.6%, and therefore accepted lowest number of scaling requests. The next was migration scaling approach, with and average of 3.89% of bandwidth rejection. The horizontal scaling approach had the lowest percentage of bandwidth rejection: average of 0.12%, and accepted highest number of scaling requests. Therefore, it can be considered as the best scaling approach, in terms of the optimization goal of accepting maximum bandwidth requests as much as possible. Our observation is that vertical scaling is always limited by the spare computational resources

of the VNF's current server, and therefore, vertical scaling is rarely able to satisfy the scaling requirements. The Migration scaling and horizontal scaling have more freedom in scaling, and able to satisfy most of the scaling requirements.

In Chapter 6, we discussed our work on exploring dynamic load balancing approaches for VNFs, where we proposed a **load balancing algorithm with session control, based on consistent hashing techniques**. We argued that, it is more appropriate to design the load balancing algorithm in a way that the algorithm itself maintain affinity and handle states and sessions, so that the requirement of state migration is avoided. We have conducted preliminary experiments for a small scaled network, to verify the accuracy and basic performances of the proposed session-aware hashing algorithm. We have considered three scenarios: (1) the system starts with more than one VNF instance, so from the beginning the traffic has to be balanced between multiple VNF instances, (2) the system starts with more than one VNF instance, but one VNF instance is removed dynamically, so the load has to be re-balanced and (3) the system starts with only one VNF instance, but one more VNF instance is added dynamically, so the load has to be re-balanced. Our preliminary results show that the proposed session-aware hashing algorithm balances load evenly (within 1.5% of ideal) within a reasonable time frame.

Furthermore, in Section 6.5, we explored two **dynamic network configurations update mechanisms** found in existing works: (1) configuration rules are updated in all switches simultaneously, and (2) versioning tags are used to maintain per-flow consistency [31]. The versioning mechanism works by stamping every incoming packet with a version number and modifying every configuration so that it only processes packets with a set version number. We implemented the physical structure of our test bed with TP-Link TL-WR1043ND switches. As the Software Defined Network (SDN) technologies based infrastructure allows to reconfigure the physical network easily, we configured switches for OpenFlow compatibility. Our results showed that when versioning tags are used to maintain per-flow consistency [31], the packet loss was low, with an average of 1.5%.

7.2 Future directions

There are several directions for future work based on the thesis. The high-level objective of the thesis was to explore different aspects of VNFs management, specifically with cloud infrastructure related operations: provisioning, configuring, maintaining and scaling of the VNFs, as well as configuring and updating of the cloud network. We proposed the NFC to offer VNFs as a service, where its management system was built around five key modules: (1) Resource Manager, (2) Topology Manager, (3) Flow Manager, (4) Elasticity Manager and (5) Rules Generator. Each module is responsible for different tasks, such as: resource allocation, monitor

cloud infrastructure, decide scaling requirements, scale VNFs , updates the network configurations, etc. All these are broad research topics that can be approached from a number of perspectives and different academic disciplines.

First and foremost, the most immediate future work is on the proposed load balancing algorithm. In this thesis we have proposed a session-aware load balancing algorithm that is implemented using consistent hashing concepts. However, we have only conducted a set of preliminary experiments to see the behaviour of the algorithm, and confirmed the accuracy of the algorithm. But, we need to explore more on efficiency and scalability of the algorithm. Therefore, the most immediate future work will be to do a more comprehensive analysis to evaluate the performances of the algorithm, in-terms of its efficiency and scalability. There are several experiments to conduct, such as to explore the behaviour of the algorithm: (1) with more and real VNFs such as Bro [144] and iptables [145], (2) in a large scaled network, (3) with more traffic load, (4) using different session time-out values etc.

As for the possible future directions, first, the Resource Manager module can be explored more for the dynamic resource allocation approaches. In this thesis, we have explored the Resource Manager module, by proposing resource allocation algorithms based on a traditional resource allocation method: ILP based approach as well as two approximation based methods: ILS and GP approaches. Even though we have used a linear equation as the objective function for our ILS and GP based implementation, it is not necessary for the objective function of ILS or GP to be linear. Further more, we have used only few factors in our objective function: server utilization, link utilization and changes to the current configurations. There are many other linear and non-linear factors that might affect the NFC: traffic lost, delay, cost of VNFs software license [99], power consumption etc. Therefore, the most immediate possible work on dynamic resource allocation will be exploring the ILS and GP based resource allocation algorithms with different optimization factors and understand how these factors effect the operations of NFC.

The second possible direction is exploring VNFs scaling with live migration approach. There are two main aspects of the research: (1) migration of VNF instances and (2) migration of traffic flows. For the migration of VNF instances, the main concern will be to speed up the process of live VNF migration without degrading the performance of the VNF. As there are lots of existing work on live migration of traditional VMs [32, 51–53], it will be interesting to explore how those existing approaches can be used with VNF instance migrations. For an instance, pre-copy and post-copy are prevalent techniques of live VM migration. Pre-copy is designed to reduce the downtime of VM during its migration, but it incurs high total migration time, even for slightly write intensive applications. Post-copy migration was introduced as an improvement over pre-copy to reduce the total VM migration time, but it accomplishes this at the cost

of increased VM downtime. For the migration of traffic flows in VNFs, the state migration is an important task. Unlike layer 3 forwarding, many NFs such as firewall, proxy, and VPN perform stateful packet processing. When the VNF has been migrated to a new server, the CSP must not only re-direct the traffic to the new placement, but also move the internal flow states associated with the traffic. Thus, efficient state migration is an important and practical issue in VNFs.

The third possible direction is the implementation of the Elastic Manager, which is the least explored module in this thesis. The Elasticity Manager is responsible for monitoring the network and servers to determine when to scale the resource allocation to meet the traffic demands according to the SLAs and QoS agreements. Therefore, the main challenge is to determine when to scale out/in the resource allocations. Finding the exact VNFs(s) or path(s) which are causing the bottleneck and deciding the right number of resources to increase/decrease to achieve the demand is important. The most basic method to scale out/in is to monitor system-level metrics (server and links utilization) and determining whether to scale out/in based on a threshold. However, threshold-based algorithms do not capture the complex interaction among multiple resource parameters (server and links) and the potential diversity of traffic types. Determining the right set of thresholds for them to simultaneously achieve the right SLA and QoS for each type of traffic would be difficult. Often the thresholds are set based on ad-hoc measurements and past experiences. We are planing to explore machine learning techniques, in particular reinforcement learning, to learn the behaviour of the applications and automatically adapt to changes. The learning algorithm can be augmented with heuristics to improve the responsiveness and guide the algorithm itself.

In the process of "Towards making VNFs as a service", we have realized the importance of the practicality of any proposed system. With the concept of NFC, we proposed an integration of different aspects of the VNFs management (resource allocation, monitoring, network configuration updating etc.). Through this thesis, lots of effort has been put into the implementation of each module individually. But the final research direction could be exploring the total integration of these modules, to make sure that the proposed aspects of the management can be implemented in the real life as individual models, as well as they can be integrated to work together and to perform the expected functionalities of the NFC.

We sincerely believe that the dissertation has opened up the new area of research in cloud management for VNFs. With several challenging and interesting research problems in the area, there is significant scope and potential for novel approaches and methodologies to solve these problems. The future directions discussed here provide pointers for further research in the field, and this is perhaps where a new PhD thesis can begin!

Part I

Appendix

Appendix A

Publications

[1] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in the proceeding of IEEE IM 2015.

[2] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Experimental results on the use of Genetic Algorithms for Virtualized Network Functions (VNFs) Scaling Management in a Cloud Center," in the proceeding of IEEE SDN-NFV 2015.

[3] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing Resources Allocation for Virtualized Network Functions in a Cloud Center using Genetic Algorithms," accepted with major revisions for IEEE Transactions on Network and Service Management.

[4] W. Rankothge, H. Ramalhinho and J. Lobo, "Towards Automation of Resource Allocation for Scaling of Virtualized Network Functions," under review for IEEE IM 2017.

Appendix B

Data Generation for the Experiments

To conduct a more realistic evaluation, we needed data on: (1) potential NFs chains (policies), (2) traffic flows passing through these NFs chains, (3) how the dynamic traffic changes affect the NFs (scale out/in) and (4) different data center architectures for the NFC. However, there are no publicly available real data sets on NF chains and traffic that pass through NF chains. We have therefore used data from previous empirical analyses [6, 27] and made some assumptions to derive the required data. We developed four programs to model the gathered data and generate the required data. All gathered data and data modelling programs are publicly available at [124].

B.1 Policy requests generation

When generating policy requests for the NFC, the main factor to be considered is the type (e.g., small, medium, large size network) of the enterprise/user, that is requesting the policies. Depending on the type of the enterprise/user, the total number of NFs required, the number of NFs in a policy and types of the NFs in the policy can vary. The policies (chains of NFs) used in the experiments are generated based on a study about middle-boxes used in enterprise networks [6]. This data set from [6] includes figures about types of enterprise networks, number and types of middle-boxes used in these enterprise networks. According to [6], a chain of NFs consists of 2 to 7 NFs, mostly 2 to 5. So the number of NFs in a policy follows a truncated power-low distribution with exponent 2, minimum 2 and maximum 7.

According [6], as shown in Figure B.1, large scaled enterprises, with 10k-100k hosts can have average: IP Firewalls: 46, Application firewalls: 9,

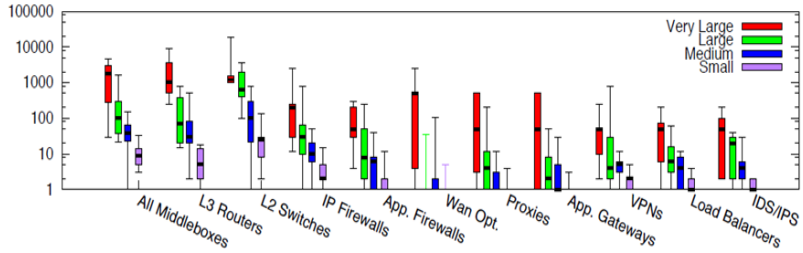

Figure B.1: Box plot of middlebox deployments for small (fewer than 1k hosts), medium (1k-10k hosts), large (10k-100k hosts), and very large (more than 100k hosts) enterprise networks. Y-axis is in log scale. [6]

WAN optimizers: 0, Proxies: 6, Gateways: 3, VPNs: 6, Load Balancers: 7, IDS/IPS: 23 and Total: 100.

Policy requests generation program

We have considered large scaled enterprise networks where each network has 100 NFs. A chain of NFs consists of 2 to 7 NFs and the number of NFs in a policy follows a truncated power-low distribution with exponent 2, minimum 2 and maximum 7. The types of NFs in a policy are selected randomly, with different probabilities based on how many instances of each type of NFs can be there in the enterprise. Policy requests generation program is written in c++.

- Inputs to the program: number of large scaled enterprise networks
- Output of the program: a set of policies for each enterprise with 100 NFs

B.2 Initial traffic distribution

After generating the policy requests, for simulating the traffic, we need traffic data where owners (enterprises/users) of the flows can be identified, so that we can differentiate the traffic passing through each policy. In the real-life situation, the clients traffic passing through the set of NFs will be directed to the different applications as web server, voip server etc according to the clients requests/needs. So the traffic load that each client is expecting can be different based on the applications client is handling [125]. For the experiments, we assume our clients are handling web based applications and the traffic used for the experiments is taken from a study about web traffic [27]. The data set includes HTTP traffic breakdown of 30,000 users for a day which is measured at three different vantage points of an Italian ISP. Traffic breakdown reports HTTP traffic for every 2 hours.

Initial traffic distribution program

We use the traffic data for each enterprise given in [27] at the starting point of the HTTP traffic breakdown, and assume it as the initial total traffic flow that will pass through all the policy chains of the enterprise. We assume that the initial total traffic load is equally distributed over the set of policies of that enterprise. The initial traffic distribution program is written in c++.

- Inputs to the program: the set of policies, initial traffic load
- Output of the program: distribution of the traffic load over policies

B.3 Scaling requirements over the time

In a data center, traffic changes happen throughout the day and according to the amount of these changes, the NFs should be scaled out/in to satisfy the dynamic demands. The limitation of our data set is it lacks the information on how the traffic changes occurred over two hours. It has information only on traffic at each two hours.

According to [2], as shown in Figure B.2, traffic changes on usual days happen gradually over time. Even at events where traffic will be increased in a huge amount (elephant flows), as shown in Figure B.3, the change happens gradually over a 15 minutes time period [3]. So we have assumed that, for the every 2 hours traffic reported in [27], increase/decrease happened uniformly through 2 hours and generated the traffic graph in Figure B.4. It shows traffic flow for 24 hours in 10MBps units for each enterprise.

But in special situations, there can be flash events, where sudden traffic changes occurred within few minutes. So we have to consider two situations: usual traffic patterns and elephant flows where traffic changes gradually and flash events where traffic changes suddenly. To reflect scaling requirements of both situations, we spread the increase/decrease of number of NFs (needed for the full 2 hours traffic change) over 2 hours and add/remove one instance at a time.

According to [7], as shown in Figure B.5, if we add more than one instance at a time, to be ready for the requirements in the future, we are adding more than what is needed and wasting resources. So we define a threshold (Maximum amount of traffic that an instance of a NF can handle) to find how many instances we should add/remove to accommodate traffic change and we will add/remove one instance at a time.

Scaling requirements over the time program

In the scaling requirements over the time program, first we define the threshold L, The maximum amount of traffic that an instance of a NF can handle. If the traffic change of a 2 hours period is greater than L, we

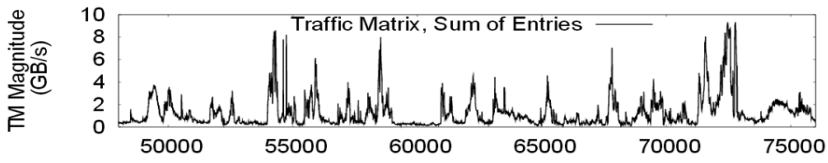

Figure B.2: Traffic in the data-center changes in the magnitude (Time in seconds). [2]

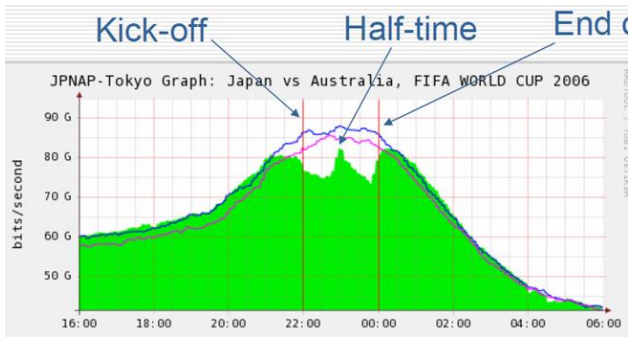

Figure B.3: Traffic statistics from World Cup 2006 [3]

assume that we have to add an extra instance of the NF. We are making an assumption: the traffic flowing through the NF instance is proportionate to the capacity of the NF instance and it is same for all types of NFs. In reality this might not be correct and different NF types may have different connections between traffic flow and capacity.

As the second step, we identify the enterprises whose traffic has changed over each 2 hours from the traffic graph. For each enterprise, we have already generated x number of policies, and assume each policy has a unique traffic flow passing through. When there is a change in the total traffic for that enterprise, it is very unlikely that traffic passing through all the policies of that enterprise contributed to the traffic change. Most probably the traffic change was caused by the traffic passing through sub set of policies. So for enterprises that have a traffic change, we randomly select 5 of its policies, as the policies affected by the traffic change.

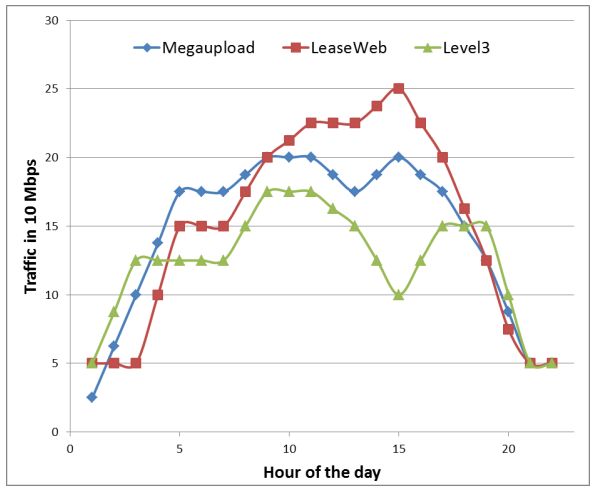

Figure B.4: Traffic over a full day

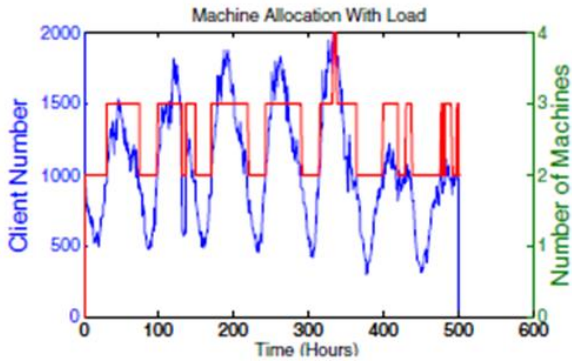

Figure B.5: Machines Allocation [7]

After selecting the policies affected by each enterprise traffic change, the third step is to identify which NF from each policy, needs to be scaled out/in to satisfy the new traffic demands. According to Stratos [10], there are simple approaches we could leverage for deciding which NF(s) to scale. The simplest solution is to scale all NFs in that policy. This guarantees that any bottleneck will be eliminated, but this potentially wastes significant resources and imposes unneeded costs, when only one NF may be the bottleneck. So Stratos performs a set of scaling trials, scaling each NF in the policy, one (VM) instance at a time. They begin by adding a

new instance of the first NF in the chain, monitoring for changes over a fixed time window. If performance improves beyond some threshold, then the new instances is permanently added to the tenants topology. No improvement means that the NF is not a bottleneck, so they discard the new instance. Then move to the next NF in the chain and repeat the process. Their results show that no two NFs will be simultaneously and equally bottlenecked and scaling one NF in the policy at a time is acceptable. Hence assuming the conditions in Stratos, we randomly select a NF from each policy as the bottlenecked NF for which the resource allocation needs to be increase/decrease.

The last step is to decide, from the identified NF instance to scale, how many instances we should add/remove to satisfy the new traffic demand. For each enterprise whose traffic has changed, first we identify the total traffic change over each 2 hours: C from the traffic graph. Then we calculate how many instances had to be add/remove for each enterprise: I based on the threshold L we defined earlier.

$I = C/L$

If we have to add/remove instances, we spread the I over 2 hours (120 minutes). As explained earlier, we are trying to add/remove instance at a time. Therefore, If I = 2, and starting time of the period is T=0, then scaling occurs when T+40 minutes and T+80 minutes. If we dont have to add/remove instances, we have to change the paths of the policies which use overloaded links because of traffic change.

Following the above process, the scaling requirements over the time program is written in c++.

- Inputs to the program: the set of policies, traffic pattern
- Output of the program: a set of policies and NFs effected by traffic changes during each interval and the required add/remove NF instances for each interval

B.4 Topology Generation

We have evaluated the performance of the resource allocation algorithm assuming three different data center network architectures for NFC: (1) k fat tree, (2) VL2 and (3) BCube as shown in figure B.6 . We have assumed environment where there are 16, 32, 48, 64, 80, 96, 112 and 128 servers in the NFC. Therefore, we needed data on: (1) nodes of the network (servers and switches of the network), (2) links of the networn (connecting two nodes), and (3) paths of the network (between each and every server of the network). All these three depends on two factors: (1) the number of servers in the NFC and (2) network architecture of the NFC.

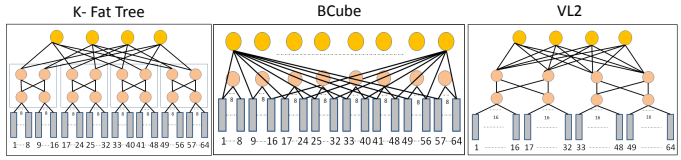

Figure B.6: (1) k fat tree, (2) BCube and (3) VL2

Topology generation program

Following the standards given for (1) k fat tree, (2) VL2 and (3) BCube architectures [24–26], we define the network and equally distribute the number of servers over ToR switches of the network. The topology generation program is written in python.

- Inputs to the program: network architecture and number of servers
- Output of the program: the topology: nodes, links and paths

Bibliography

Each reference indicates the pages that it appears.

[1] ETSI, "Network functions virtualisation: White paper," *SDN and OpenFlow World Congress*, 2013. xi, xiii, 4, 5, 16, 31

[2] K. Srikanth, S. Sudipta, and at el., "The nature of data center traffic: Measurement and analysis," in *ACM SIGCOMM IM 2009*, 2009. xviii, xix, 42, 70, 149, 150

[3] Y. Tarui, "Analyzing the impact of major social events on internet exchange traffic," in *NANOG38*, 2009. xviii, xix, 42, 43, 149, 150

[4] H. R. Lourenco, O. Martin, and T. Stutzle, *A beginners introduction to Iterated Local Search*, 2010. xviii, 59, 60, 61, 62

[5] ——, *Iterated Local Search*, 2010. xviii, 59, 60, 61, 62

[6] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, and at el., "Making middleboxes someone else problem: network processing as a cloud service," in *ACM SIGCOMM '12*, 2012. xix, 3, 19, 41, 136, 147, 148

[7] W. Wang, H. Chen, and X. Chen, "An availability-aware virtual machine placement approach for dynamic scaling of cloud applications," in *Conference on Autonomic and Trusted Computing*, 2012. xix, 17, 44, 149, 151

[8] "Cloud computing," https://www.ibm.com/cloud-computing/learnmore/what-is-cloud-computing. 2, 16

[9] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," in *Journal of Networks and Systems Management 2015*. 2, 3, 5, 20

[10] A. Gember, R. Grandl, A. Anand, and at el, "Stratos: Virtual middleboxes as first-class entities," *Technical Report TR1771*, 2013. 3, 6, 7, 19, 43, 151

[11] J. G. Herrera and J. F. Botero, "Resource allocation in nfv: A comprehensive survey," in *IEEE TNSM 2016*. 5, 6, 23, 47

[12] R. Mijumbi, J. Serrat, J. Gorricho, and at el, "Network function virtualization: State-of-the-art and research challenges," in *IEEE Communications Surveys and Tutorials 2015*. 6, 18

[13] T. Lukovszki, M. Rost, and S. Schmid, "Its a match! near-optimal and incremental middlebox deployment," in *ACM SIGCOMM Computer Communication Review: Jan 2016*. 6, 48, 59

[14] R. Cohen, L. Lewin-Eytan, and at el, "Near optimal placement of virtual network functions," in *INFOCOMM 2015*. 6, 24, 48, 59

[15] Y. Li, L. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *IEEE INFOCOM*, 2016. 6, 9, 25, 29, 47, 95, 98, 101

[16] G. Milad, K. Aimal, S. Nashid, A. Khalid, A. Reaz, and B. Raouf, "Elastic virtual network function placement," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, 2015. 6, 9, 25, 29, 47, 95

[17] M. Ghaznavi, A. Khan, N. Shahriar, and at el., "Elastic virtual network function placement," in *IEEE CloudNet*, 2015. 7, 9, 29, 37, 48, 95, 96

[18] S. Jain, A. Kumar, and et al, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM '13*, 2013. 7, 17, 26, 34, 133

[19] M. Shin, K. Nam, and H. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *ICTC '12*. 7, 17

[20] B. S. Networks, "The open sdn architecture - big switch networks," in *White paper 2011*. 7, 17

[21] T. C. V. C. Mathieu Bouet, Jrmie Leguay, "Cost-based placement of vdpi functions in nfv infrastructures," in *International Journal of Network Management*, 2015. 9, 29, 47

[22] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Experimental results on the use of genetic algorithms for scaling virtualized network functions," in *IEEE SDN/NFV 2015*. 9, 29, 100

[23] H. R. Lourenco, O. Martin, and T. Stutzle, *Iterated Local Search: Framework and Applications. In Handbook of Metaheuristics*, 2010. 10, 48, 59

[24] C. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," in *IEEE Transactions on Computers*, 1999. 10, 45, 136, 137, 153

[25] C. Guo, G. Lu, D. Li, and at el, "Bcube: a high performance, server-centric network architecture for modular data centers," in *ACM SIGCOMM 2009*. 10, 45, 136, 137

[26] A. Greenberg, J. R. Hamilton, N. Jain, and at el, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM 2009*. 10, 45, 136, 137, 153

[27] G. Vinicius, F. Alessandro, M. Marco, and at el., "Uncovering the

big players of the web," in *ICTMA '12*. 10, 41, 42, 136, 137, 147, 148, 149

[28] "Cplex," http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/. 10, 13, 58, 90, 137

[29] Z. Qazi, C. Tu, L. Chiang, and at el, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM '13*, 2013. 12, 19, 25, 27, 30, 33, 37, 38, 39, 111

[30] A. Gember, R. Viswanathan, and at el, "Opennf: Enabling innovation in network function control," in *ACM SIGCOMM 2014*. 12, 27, 30, 111

[31] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM'12*. 14, 40, 112, 133, 139

[32] C. Clark, K. Fraser, and at el., "Live migration of virtual machines." in *NSDI 2005*. 17, 19, 22, 140

[33] D. Batista, G. Blair, F. Kon, and at el, "Perspectives on software-defined networks: interviews with five leading scientists from the networking community," in *Journal of Internet Services and Applications 2015*. 18

[34] M. Bari, R. Boutaba, and at el., "Data center network virtualization : A survey," in *IEEE Communications*, 2014. 19

[35] N. Chowdhury and R. Boutaba., "Network virtualization: State of the art and research challenges," in *IEEE Communications*, 2009.

[36] A. Fischer, J. Botero, and at el., "Virtual network embedding : A survey," in *IEEE Communications*, 2013. 19

[37] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," in *HotSDN '12*. 19

[38] S. Fayazbakhsh, M. K. Reiter, and V. Sekar, "Verifiable network function outsourcing: requirements, challenges, and roadmap," in *HotMiddlebox '13*. 19

[39] V. Sekar, N. Egi, S. Ratnasamy, and at el, "Design and implementation of a consolidated middlebox architecture," in *NSDI '12*, 2012. 19

[40] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *HotNets-XI*, 2012. 19

[41] J. Martins, M. Ahmed, and at el, "Clickos and the art of network function virtualization," in *NSDI '14*, 2014. 19, 28, 100

[42] L. Shi, B. Butler, and at el, "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds." in *IEEE IM '13*, 2013. 19

[43] D. Jayasinghe, C. Pu, and at el, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement." in *IEEE SCC '11*.

[44] K. Konstanteli, T. Cucinotta, K. Psychas, and at el., "Admission

control for elastic cloud services," in *IEEE Cloud '12*, 2012. 19

[45] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," in *JNSM '14*, 2014. 19

[46] "Amazon ec2," http://aws.amazon.com/ec2/. 19

[47] "Microsoft azure," http://www.microsoft.com/azure/default.mspx. 19

[48] Z. Shen, S. Subbiah, and at el., "Elastic resource scaling for multi-tenant cloud systems," in *ACM SoCC 2011*. 19

[49] Z. Gong, X. Gu, and at el., "Press: Predictive elastic resource scaling for cloud systems," in *IEEE CNSM, 2010*. 19

[50] M. Sedaghat, F. Hernandez-Rodriguez, and at el., "A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling," in *ACM CAC 2013*. 19

[51] S. Akoush, R. Sohan, and at el., "Predicting the performance of virtual machine migration." in *MASCOTS 2010*. 19, 140

[52] S. Kikuchi and Y. Matsumoto, "What will happen if cloud management operations burst out?" in *IM 2011*.

[53] M. Nelson, B. Lim, and at el., "Fast transparent migration for virtual machines." in *USENIX 2005*. 19, 140

[54] "Xensource inc," http://www.xensource.com. 19, 97

[55] "Vmware esx server," http://www.vmware.com/products/esx. 19, 97

[56] C. Clark, K. Fraser, and S. Hand, "Live migration of virtual machines," in *NSDI 2005*. 19

[57] C. Jr, G. E.G., and J. M.R., "Approximation algorithms for np-hard problems," in *PWS Publishing Co., Boston*. 20

[58] R. Panigrahy, K. Talwar, and at el., "Heuristics for vector bin packing," in *Technical report, Microsoft Research (2011)*. 20

[59] D. Wilcox, A. McNabb, and at el., "Solving virtual machine packing with a reordering grouping genetic algorithm." in *IEEE CEC 2011*. 20

[60] M. Cardosa and M. K. at el., "Shares and utilities based power consolidation in virtualized server environments." in *IEEE IM 2009*. 20

[61] B. Viswanathan, A. Verma, and S. Dutta, "Cloudmap: workload-aware placement in private heterogeneous clouds." in *IEEE IM 2012*. 20

[62] X. Meng, C. Isci, and at el., "Efficient resource provisioning in compute clouds via vm multiplexing." in *ICAC 2010*. 20

[63] L. Shi, Butler, and at el., "Provisioning of requests for virtual machine sets with placement constraints in iaas clouds." in *IEEE IM 2013*. 20

[64] D. Jayasinghe, C. Pu, and at el., "Improving performance and avail-

ability of services hosted on iaas clouds with structural constraint-aware virtual machine placement." in *SCC 2011*. 20, 21

[65] K. Konstanteli, T. Cucinotta, and at el., "Admission control for elastic cloud services," in *IEEE CLOUD 2012*. 20

[66] D. Breitgand and A. Epstein, "Sla-aware placement of multi-virtual machine elastic services in compute clouds." in *IEEE IM 2011*. 20

[67] P. De and S. Roy, "Vmspreader: multi-tier application resiliency through virtual machine striping." in *IEEE IM 2011*. 20

[68] E. Bin, O. Biran, and at el., "Guaranteeing high availability goals for virtual machine placement." in *ICDCS 2011*. 21

[69] X. Meng, V. Pappas, and at el, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *GIIS '12*. 21, 23, 37, 48

[70] D. Breitgand and A. Epstein, "Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds." in *Infocom 2012*. 21

[71] J. Schad, J. Dittrich, and at el., "Runtime measurements in the cloud: observing, analyzing, and reducing variance." in *VLDB Endow 2010*. 21

[72] M. Wang, X. Meng, and L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers." in *Infocom 2011*. 21

[73] J. Kleinberg, Y. Rabani, and at el., "Allocating bandwidth for bursty connections." in *STOC 1997*. 21

[74] V. Shrivastava, P. Zerfos, and at el., "Application-aware virtual machine migration in data centers." in *Infocom 2011*. 21

[75] X. Wen, K. Chen, and at el., "Virtualknotter: online virtual machine shuffling for congestion resolving in virtualized datacenter," in *ICDCS 2012*. 21

[76] O. Biran, A. Corradi, and at el., "A stable networkaware vm placement for cloud systems." in *CCGrid 2012*. 21

[77] J. Jiang, T. Lan, and at el., "Joint vm placement and routing for data center traffic engineering." in *Infocom 2012*. 21

[78] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds." in *Infocom 2012*. 21

[79] L. Hu, K. Ryu, and at el., "v-bundle: Flexible group resource offerings in clouds." in *ICDCS 2012*. 21

[80] F. Wuhib, R. Yanggratoke, and at el, "Allocating compute and network resources under management objectives in large-scale clouds," in *JNSM 2013*. 21

[81] N. Bobroff, A. Kochut, and at el., "Dynamic placement of virtual machines for managing sla violations." in *IEEE IM 2007*. 22

[82] U. Sharma, P. Shenoy, and at el., "Kingfisher: Cost-aware elasticity in the cloud." in *Infocom 2011*. 22

[83] S. Bazarbayev, M. Hiltunen, and at el., "Content-based scheduling of virtual machines (vms) in the cloud." in *ICDCS 2013*. 22

[84] M. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive prepaging and dynamic self-ballooning," in *VEE 2009*.

[85] H. Liu, H. Jin, and at el., "Live migration of virtual machine based on full system trace and replay." in *HPDC 2009*.

[86] V. Mann, A. Vishnoi, and at el., "Crossroads: seamless vm mobility across data centers through software defined networking." in *NOMS 2012*. 22

[87] N. Bobroff, A. Kochut, and at el., "Dynamic placement of virtual machines for managing sla violations." in *IEEE IM 2007*. 22

[88] U. Sharma, P. Shenoy, and at el., "A cost-aware elasticity provisioning system for the cloud." in *ICDCS 2011*. 22

[89] X. Zhang, Z. Shae, and at el., "Virtual machine migration in an over-committed cloud." in *NOMS 2012*. 22

[90] H. Xu and B. Li, "Egalitarian stable matching for vm migration in cloud computing." in *Infocom Workshop 2011*. 22

[91] D. Gale and L. Shapley, "College admissions and the stability of marriage," in *Am. Math.* 22

[92] J. Dejun, G. Pierre, and at el., "Resource provisioning of web applications in heterogeneous clouds." in *WebApps 2011*. 22

[93] J. Schad, J. Dittrich, and at el., "Runtime measurements in the cloud: observing, analyzing, and reducing variance." in *Proc. VLDB Endow.* 22

[94] T. Guo, U. Sharma, and at el., "Seagull: intelligent cloud bursting for enterprise applications." in *USENIX 2012*. 22

[95] M. Shifrin, R. Atar, and I. Cidon, "Optimal scheduling in the hybrid-cloud." in *IEEE IM 2013*. 22

[96] G. Foster, G. Keller, and at el., "The right tool for the job: switching data centre management strategies at runtime." in *IEEE IM 2013*. 22

[97] M. Bari, S. Chowdhury, and at el, "On orchestrating virtual network functions," in *CNSM 2015*. 23, 100

[98] R. Mijumbi, J. Serrat, and at el, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *NetSoft 2015*. 23

[99] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, and L. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IEEE IM '15*. 23, 140

[100] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE CloudNet 2015*. 23

[101] J. F. Riera, X. Hesselbach, and at el., "Modelling the nfv forwarding

graph for an optimal network service deployment," in *IEEE ICTON 2015.* 23

[102] R. Bruschi, A. Carrega, and F. Davoli, "A game for energy-aware allocation of virtualized network functions," in *Journal of Electronics and Computer Engineering 2016.* 24

[103] B. Martinia, F. Paganelli, P. Cappanera, and at el, "Latency-aware composition of virtual functions in 5g," in *IEEE NetSoft 2015.* 24

[104] J. Elias, F. Martignon, and at el, "Efficient orchestration mechanisms for congestion mitigation in nfv: Models and algorithms," in *IEEE Transaction on Services and Computing 2015.* 24

[105] A. Mohammadkhan and et al, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *IEEE LANMAN 2015.* 24

[106] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *SIROCCO 2015.* 24

[107] T. Lin, Z. Zhou, and at el, "Optimal network function virtualization realizing end-to-end requests," in *IEEE GLOBECOM 2015.* 24

[108] S. Sahhaf and at el, "Network service chaining with optimized network function embedding supporting service decompositions," in *Computer Networks 2015.* 24

[109] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vdpi functions in nfv infrastructures," in *IEEE NetSoft 2015.* 24

[110] T. Lin and Z. Zhou, "Demand-aware network function placement," in *Journal of Lightweight Technologies 2016.* 24

[111] M. Ghaznavi and at el., "Elastic virtual network function placement," in *IEEE CloudNet 2015.* 25

[112] F. Wang, R. Ling, and at el., "Bandwidth guaranteed virtual network function placement and scaling in datacenter networks," in *IPCCC 2015.* 25

[113] N. Handigol, S. Seetharaman, and at el, "Plug-n-serve: load-balancing web traffic using openflow," in *ACM SIGCOMM Demo 2009.* 26

[114] R. Wang, D. Butnariu, J. Rexford, and at el, "Openflow-based server load balancing gone wild," in *Hot-ICE 2011.* 26

[115] L. A. Batista, L. Campos, and at el, "Flow-based conflict detection in openflow networks using first-order logic," in *ISCC 2014.* 26

[116] Y. Li and D. Pan, "Openflow based load balancing for fat-tree networks with multipath support," in *ICC 2013.* 26

[117] M. Koerner and O. Kao, "Multiple service load-balancing with openflow," in *IEEE HPSR 2012.* 26

[118] R. Sherwood, G. Gibb, and at el, "Flowvisor:a network virtualization layer," in *OpenFlow Switch Consortium: Technical Report.* 26

[119] S. Rajagopalan, D. Williams, and at el, "A. split/merge: System support for elastic execution in virtual middleboxes," in *USENIX*

NSDI 2013. 27, 111

[120] P. Shoumik, L. Chang, H. Sangjin, and at el, "E2: a framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015. 27, 112

[121] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, and V. Jain, "An intent-based approach for network virtualization," in *IM 2013*. 33, 38

[122] "Openflow 1.4 specifications," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow. 34, 126, 133

[123] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *HotSDN'13*. 39

[124] "Test data," https://github.com/windyswsw/DataForNFVSDNExperiments. 41, 136, 147

[125] S. Gebert, R. Pries, D. Schlosser, and K. Heck, "Internet access traffic measurement and analysis," in *ICTMA '12*. 42, 148

[126] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: New insights," in *Structural and Multidisciplinary Optimization 2010*. 52, 57

[127] I. Giagkiozis and P. Fleming, "Methods for multi-objective optimization: An analysis," in *Information Sciences Journal 2014*. 52, 57

[128] C. Clark, K. Fraser, S. Hand, J. G. Hansen, and at el., "Live migration of virtual machines," in *NSDI*, 2005. 68, 83

[129] M. Melanie, *An Introduction to Genetic Algorithms*, 1999. 74

[130] A. Verma, G. Kumar, and R. Koller, "The cost of reconfiguration in a cloud," in *Middleware (Industrial Track) 2010*. 97

[131] Q. Zhang, L. Cheng, and at el., "Cloud computing: state-of-the-art and research challenges," in *Journal of Internet Services and Applications 2010*. 97

[132] D. Dietrich, A. Abujoda, and at el., "Network service embedding across multiple providers with nestor," in *IFIP Networking 2015*. 100

[133] S. Palkar, C. Lan, and at el., "E2: a framework for nfv applications," in *OSP 2015*. 101

[134] "Design best practices for latency optimization," https://www.cisco.com. 101

[135] D. Karger, E. Lehman, and at el, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *STOC 1997*. 112, 113, 114

[136] "Consistent hash ring," https://infinitescript.com/2014/10/consistent-hash-ring/. 114

[137] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid

prototyping for software-defined networks," in *ACM HotNets'10*, 2010. 124, 133

[138] "Ryu sdn controller," http://osrg.github.io/ryu/. 124, 133

[139] "Pcapplusplus," https://github.com/seladb/PcapPlusPlus. 125, 126

[140] "iperf - the ultimate speed test tool for tcp, udp and sctp," https://iperf.fr/. 128, 129

[141] "httperf(1) - linux man page," https://linux.die.net/man/1/httperf. 130

[142] "All about pox," http://www.noxrepo.org/pox/about-pox/. 133

[143] "Project floodlight," http://www.projectfloodlight.org/floodlight/. 133

[144] "The bro network security monitor," https://www.bro.org/. 133, 140

[145] "Netfilter: iptables," http://www.netfilter.org/. 133, 140

– Notes –