PLACEMENT AND SCHEDULING OF NETWORK TRAFFIC ON VIRTUAL NETWORK FUNCTIONS

Nicolas El Khoury

A thesis

in

The Department

of

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science in Information Systems Security

Concordia University

Montréal, Québec, Canada

October 2016

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Nicolas El Khoury**

Entitled: **Placement and Scheduling of Network Traffic on Virtual Network Functions**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science in Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

C. Wang ______ Chair

A. Youssef ______ CIISE Examiner

W. Hamouda ______ External Examiner

C. Assi ______ Supervisor

Approved ______

Chair of Department or Graduate Program Director

______ 20 ______ ______

Dean of Faculty

Abstract

Placement and Scheduling of Network Traffic on Virtual Network Functions

Nicolas El Khoury

Hardware MiddleBoxes represent a vital part in today's networks. Despite their important roles, they are accompanied by several problems, namely, their lack of flexibility, high capital and operational expenditures, and power consumption. Network Function Virtualization is one promising solution to address these problems. This trend replaces the MiddleBoxes by software-based entities. Indeed, these Virtual Network Functions promise to alleviate the numerous disadvantages brought by their hardware counterparts. One of these most serious issues is the steadily increasing power consumption. Studies suggest that the Virtual Network Functions will reduce the electricity costs needed to turn on and operate the hardware functions. In order to further optimize the power consumption of the network, an efficient framework, capable of placing and scheduling traffic on these VNFs, is needed. Such a framework allows to optimally map and schedule the flows to be serviced, and place the unused servers in energy saving modes. In this thesis, we assume VNFs are already placed on physical machines. We consider traffic flows with deadlines. We aim at assigning and scheduling flows to VNFs in the most energy efficient manner. We formulate this problem mathematically and, owing to its complexity, present an efficient algorithmic method for solving the problem. We compare our heuristic with two other approaches, one of which aims to minimize the makespan, and the other to minimize number of servers used. We show that our heuristic combines the advantages of both approaches and generates better results by consuming up to 31.3% and 46.1% energy less than other two approaches respectively. Further, we extend the existing work in the literature, and solve the problem of placement of traffic flows on VNFs while taking into account the transmission delay between pairs of VNFs, and the routing of the virtual links on the underlying physical network.

Acknowledgments

I would like to express my sincere gratitude to everyone who made this achievement possible, by giving technical and personal advice and motivation. I would like to begin by thanking Professor Chadi Assi. Truly, Professor Assi is an outstanding person. Throughout these two years, he has never stopped believing in my efforts, and always pushed me to bring the best in me. More importantly, he has always been available for either a technical help or a personal advice. In summary, I would like to say that I was very lucky to have worked under the supervision of professor Assi, and I owe him a lot.

Second, I would like to thank my family (my father Nabil, my mother Amal, and my brother Christophe), whom without their support, I would have never been able to complete their work. I would like to acknowledge the amount of advice, personnal and emotional help that I have received from them, especially during my down moments. I would like to especially thank my father, who is also my best friend. My father made everything possible. He is always the first one by my side in the beautiful and ugly moments. Whenever I was weak, my father pushed me back on my feet. I hope to God I can be as great as my father is.

Third, I would like to express my love and gratitude to my girlfriend Josette El Kreidy. I am blessed to have such an amazing, strong, educated, and intelligent woman who stood by my side during this journey, accompanying me everyday, and encouraging me to bring the best in me. Despite the distance between us, she never quit on me. Rather, she made it all possible with her kindness, and constant support.

Fourth, no words can express the greatness of my roommate, my best friend, and my brother Adel Belmona Zeineddine. Adel proved many times, that he is my best man. During my happiest moments, Adel was happier than me. During my worst moments, Adel, not only had to endure me, but he never quit on me, and always supplied me with advice to keep moving forward. I will never forget how Adel took care of me after my accident, which left me with many broken bones. At that time, Adel took care of me, and made sure to cheer me up on a daily basis. Adel is not a friend. Adel is a brother, a best man, and a treasure to keep for life.

Fifth, I would like to thank my family in montreal: The Habib and Bitar families, Mr. Roland Dick, and Joe Kanaan. These people have been the biggest help to me in Montreal. While we are not blood related, they considered me as part of their families, and treated me as one of their own. Words cannot express how much I appreciate everything they have done to me during my stay in Montreal, especially in my early days where getting settled would have been extremely impossible without them.

Sixth, I would like to thank three special people that I have met in Montreal and became great friends: Sara Ayoubi, Ribal Atallah, and Elias Bou-Harb. These three friends, since day one, became special to me, because of their kind heart. They helped me a lot on both, the technical and personal level. I truly appreciate every moment I spent with them, and every piece of advice they have offered.

Seventh, I would like to sincerely thank the help and effort spent by Dr. Anna Lenkovskaya, especially during the last periods of my thesis. Dr. Lenkovskaya is an outstanding person with a kind heart, and a surreal determination to motivate people, and help them achieve their goals.

Finally, I would like to thank Professor Maurice El-Khabbaz for introducing me to Professor Chadi Assi, and every help he has offered to me during my stay in Canada. Professor El-Khabbaz is an outstanding person, and a wonderful friend.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

SN .... Substrate Network

PS .... Physical Server

PL .... Physical Link

VN .... Virtual Network

VL .... Virtual Link

VM .... Virtual Machine

VNF .... Virtual Network Function

MB .... MiddleBox

CAPEX .... Capital Expenditure

OPEX .... Operational Expenditure

SLA .... Service Level Agreement

NP-hard .... Non-deterministic Polynomial-time hard

IaaS .... Infrastructure as a Service

PaaS .... Platform as a Service

SaaS .... Software as a Service

ILP .... Integer Linear Programming

Chapter 1

Introduction

1.1 Background

1.1.1 Cloud Computing

When using many of the household utilities (e.g., electricity, water, gas, etc.), clients are unaware about how they are generated or how they are delivered to them. The utilities are made readily available for the users [1] who are billed according to their usages. This concept was applied in the Information Technology (IT) world, and led to the birth of *Cloud Computing*. The NIST defines cloud computing as *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and*

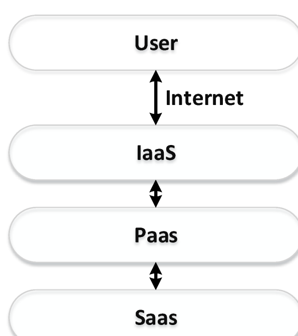


Figure 1: Cloud Computing Business Model.

released with minimal management effort or service provider interaction [2]. This computing model alleviates several burdens on both individual users and companies. Indeed, clients no longer need to worry about installing any hardware or software components, but rather, may just rent them. This concept is deemed feasible due to the virtualization of the hardware components in the data center. For instance, a Physical Server (PS) may host several Virtual Machines (VMs), each of which may run a different service.

In practice, the cloud architecture is divided into three groups (Figure 1), each of which offers a type of services:

- **Infrastructure as a Service (IaaS):** The provisioning of the physical resources (storage, computing, etc.). Examples of Iaas providers are Amazon EC2 [3], and GoGrid [4].

- **Platform as a Service (PaaS):** The provisioning of platform services, such as Operating Systems (OS) and development frameworks. Google App Engine [5], and Microsoft Windows Azure [6] are two main PaaS providers

- **Software as a Service (SaaS):** The provisioning of applications, used on demand, and accessed by the internet. Examples of SaaS Providers are RackSpace [7], and BroadSoft [8].

1.1.2 Network Function Virtualization

Cloud computing relies on virtualizing the general purpose PSs, and leasing them to clients. This concept paved the way into also virtualizing the network functions in the networks. Indeed, in addition to the general purpose PSs, the networks contain specialized servers who perform definitive functions in the networks. These specialized servers are called MiddleBoxes (MBs).

a MiddleBox is defined as *any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host* [9]. Examples of such NFs include, but are not limited to, Firewalls (restrict access to/from specific devices and applications), Network Address Translators (map IP addresses from one address realm to another), Proxies (relay messages between users and servers of an application) [10], etc. Indeed, MiddleBoxes have recently become a ubiquitous element in operators' networks; today's enterprises consist of a large number of hardware MBs in order to satisfy the diverse demands of the numerous flows travelling through the network. These MBs are physically placed at fixed locations in the network. Every traffic flow in need of a specific service must be routed to a MB offering that service itself. A flow may not necessarily request one network function only. Rather, it may require the traversal of an ordered set of service functions, a phenomenon called *Service Function Chaining*, also called *Policy Chain* [11]. Figure 2 illustrates two different SFCs. Evidently, the SFCs may differ in the number, and order of functions to be traversed by a flow. The IETF presents several SFCs use cases in data centers [12], and mobile [13].

In defiance of their increased popularity, MBs are also becoming more of a burden than a solution to network operators:

- The processing capacity of a hardware MB is fixed; such inflexibility limits their ability to optimally operate and concurrently being energy efficient. For example, in periods of low traffic, the MB will be running with full power, and therefore under-utilized. On the other hand, in cases of abundant traffic, it is not possible for MBs to serve more than their fixed capacities, causing potential congestion in the network. In order to handle these traffic surges, network operators tend to over-provision in the number and type of hardware NFs procured [14].

- Each function is deployed at a fixed physical location within the network. A request for a certain service chain might force the traffic to travel the network back and forth, thus occupying unnecessary bandwidth from some links.

- Hardware MBs are function-specific. It is not possible for one MB to provide more than one service, nor can it be reconfigured with another function. The lack of dynamic change renders the network vulnerable to excessive use of the resources to accommodate the constant changes in the traffic flowing.

- MBs come with high capital expenditure (CAPEX) and operational expenditure (OPEX). They are expensive to purchase, maintain, and operate. Furthermore, given the short-hardware lifecycle and the fast-pace technological advancements, adding new services and functionalities or replacing existing hardware requires the procurement of more hardware NFs, which is a tedious and cumbersome procedure [15], [14].

(a) Policy Chain 1 (b) Policy Chain 2

Figure 2: Service Function Chains.

Network Function Virtualization (NFV) is a promising new technology that aims to tackle the aforementioned limitations of hardware NFs [15]. The tremendous benefits that virtualization has offered to multi-tenant cloud networks [16] has inspired the virtualization of middleboxes by decoupling the NFs from the hardware. NFV consists of virtualizing the middleboxes into software-based functions which can run atop any commerical-off-the-shelf (COTS) hardware [17] (standard servers, or routers/switches [18]). This greatly reduces the CAPEX in terms of hardware provisioning and energy cost, since these Virtualized NFs (VNFs) can be consolidated, deployed/removed, and migrated anywhere in the network. Moreover, VNFs support targeted instantiation, multi-tenancy, and elasticity (scale-up and scale-out) to fit the unpredictable and fluctuating nature of traffic demands; thereby alleviating any network resource wastage due to over-provisioning. These software-based NFs can also be easily upgraded/modified [15], thus, reducing OPEX required to maintain/configure them, and greatly enhancing the time-to-market.

Now, despite the immense benefits that NFV brings, several technical challenges need to be addressed to enable NFV to fulfil these aforementioned promises. Indeed, standardization bodies such as IETF [11] and ETSI [15], as well as recent studies [19, 20] have pin-pointed these challenges; among which we find concerns of management and orchestration, resource allocation, automation, security, etc. Accordingly, NFV has received significant attention from the literature in response to these calls for actions. While some [20–25] aimed at solving the orchestration and management concerns (i.e., the problem of finding the optimal number and placement of VNFs in the network, along with routing the flows through them.), others aimed at facilitating the realization of NFV by developing virtualized software-based NFs platform (e.g. ClickOS [14], NetVM [17], and Stratos [26]). However, these above solutions did not account for the processing delays at the functions. The orchestration and management problem does not consider precedence in the processing of functions in a SFC. For clarity, a flow traversing a policy chain similar to that of (Fig. 2-(a)) may not begin its processing on the second function before it stops using the first one. Therefore, instead of keeping the second function idle, it may be able to serve another flow while the first one is being processed on the first function. Therefore, tackling the problem of placing VNFs and routing flows through them alone is not enough. Assigning the order of function requests on a typical VNF is a vital problem to be studied. This flow-to-VNF scheduling problem has received only very little attention [27], and [28].

1.2 Problem Definition

1. **The Substrate Network:**

   We represent the substrate network as an undirected graph, denoted by $G^s = (K, L)$, where K is the set of substrate nodes, and L is the set of substrate links. Each node $k \in K$ may host one or a set of VNFs, and each link $l \in L$, has a finite capacity b_l.

2. **Virtual Network Functions:**

   The VNFs represent the different types of functions in the network. In this work, we assume that the VNFs are already placed on the PSs. Therefore let w_f^k denote the type of VNFs placed on server k. Typically, $w_f^k = 1$ if VNF of type f is placed on server k. We assume that all the VNFs have the same processing power, and each one is able to serve only one traffic request at a given time; if two traffic flows are scheduled to use a particular VNF, one of them has to wait for the other to finish its processing before being serviced by that VNF.

3. **The Virtual Network:**

   A virtual network is represented as the set of VNFs placed on the physical servers. We assume that each pair of VNFs $(\bar{f}, \bar{f}')$ is directly connected one to another through a virtual link $e = (\bar{f}, \bar{f}')$. Let $o(e)$ and $d(e)$ denote respectively the origin and destination of virtual link e. We assume that each e has a bandwidth requirement to be ensured on the physical link, and a transmission rate r_e. Similar to the VNFs, each link e may only transmit the traffic of only one flow at a time.

4. **Traffic Flows:**

   Let S be a set of traffic flows. A flow s_i ($i \in S$) consists of a sequence of network functions F_i which need to be processed in a proper order, and without violating a deadline d_i. f_{ij} represents the j^{th} function of flow s_i. Similar to the VNFs, for every flow s_i, each f_{ij} belongs to a distinct type m. Therefore m_{ij} denotes the type of f_{ij}. As mentioned previously, the order of processing through the chain of a service must be respected. For instance, f_{ij} may not begin its processing before its previous $f_{i(j-1)}$ finishes its processing. Each s_i has a size c_i, and thus processing times p_{ij} and $\frac{c_i}{r}$ respectively, which depend on the size of f_{ij}.

   Problem Definition 1. *Given a substrate $G^s = (K, L)$ and a set of traffic flows S, each with a forwarding policy F_i, find the optimal schedule of the flows on the already placed VNFs while respecting the capacity constraints of the substrate network.*

1.3 Reasoning

Very little work has been done on the placement and scheduling of flows on VNFs problem. As the problem has been proven to be *NP-Hard*, most of the work focused on creating heuristic and meta-heuristic approaches to overcome the issue [28]. The main purpose of the existing work is to minimize the makespan (i.e. the minimum schedule to serve all the flows). However, two important factors remain not addressed, and they constitute the main contributions in the thesis:

- **Network power consumption:** The existing work does not address the issue of energy saving in their placement and scheduling frameworks. Recently, however, studies [29], [30] have shown that an idle server consumes power approximately 70% of that consumed by that same server running at full power. As a result, ETSI stresses on the need of a NFV framework able to optimize energy consumption on demand whether by scheduling and placing VNF instances on specific resources or placing unused resources in energy saving modes [31]. In the first part of the thesis, we present a mathematical model that places and schedules flows on VNFs in a way to optimize the network power consumption. We then present an algorithmic solution, and test it on large scale test cases.

- **Placement, Scheduling, and Routing:** There exists no frameworks that considers the bandwidth consumption on the physical links. Therefore, no work addresses the problem routing of the traffic in the network. In the second part of the thesis, we solve the problem of the placement and scheduling of traffic flows on VNFs, and propose to jointly address the problem of policy-aware traffic steering (while considering transmission delays) in the underlying physical network, with the aim of maximizing the number of flows admitted.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, related background knowledge are introduced; we scan the literature that focused on: NFV Frameworks and Architectures, VNF Orchestration and Management, and the placement and scheduling of flows on VNFs. Chapter 3 presents the network power consumption. In this chapter, we present an ILP model that places and schedules traffic flows on VNFs with the aim to minimize the power consumption of the network. We design a heuristic to overcome the non-scalability of the model, and compare it with the model itself, and two other benchmarks. Chapter 4 takes the placement and scheduling problem one step further, and addressed the problem of routing the traffic on physical links. We present an ILP model, and test its scalability. Finally, conclusion and future work are presented in Chapter 5.

Chapter 2

Related Work

2.1 ETSI NFV-MANO

Network Functions Virtualisation adds changed the behaviour of the control plane in the communication networks. In older networks, the MB implementations are often tightly coupled with the infrastructure they run on. NFV, on the other hand, is based on the decoupling of the VNFs from the hardware infrastructure, such as the computation, storage, and networking resources they use. The virtualisation creates a layer that seperates the hardware platform from the network functions. This decoupling exposes a new relationship between the VNFs, and the infrastructure. The virtualisation principle stimulates a multi-vendor system, which now require an enhanced platform able to cope with the changes, especially in terms of management and orchestration, administration, maintenance and provisioning. In what follows we present a management and orchestration framework required for the provisioning of virtualised network functions defined by ETSI [32], and serves as a core to numerous NFV projects and Implementations. The objectives of that draft are to define the framework, provide requirements for management and orchestration, identify issues and problems that may be further studied in the future, and suggest practices and provide guidance on how to solve the identified problems.

ETSI define three categories of virtualized components:

- Computing hardware such as CPU and memory.

- Storage spaces on either block or file-system level.
- Network components such as networks, subnets, ports, addresses, links and forwarding rules.

The virtualised resources are leveraged for providing VNFs with the resources they need. Resource allocation may be complex task because of the numerous requirements and constraints that may need to be met in parallel. As such, an efficient management and orchestration framework is needed. The proposed framework should be able to perform the following tasks:

- Instantiate VNF: create a VNF.
- Scale VNF: increase or reduce its capacity.
- Update and/or Upgrade VNF: Support VNF configuration change.
- Terminate VNF: Release the allocated physical resources to the VNF.

The NFV-MANO architectural framework has three functional blocks:

- *Virtualised infrastructure manager (VIM):* responsible for controlling and managing the compute, storage and network resources of the network infrastructure.
- *NFV Orchestrator (NFVO):* responsible for orchestrating the infrastructure resources across all VIMs and managing the lifecycles of the network resources.
- NFV Orchestrator (NFVO): responsible for managing the lifecycles of the VNFs (e.g., Instantiation, Upgrade, Deletion, Destruction of the VNFs).

2.2 NFV Projects and Implemetations

In this section, we present some VNF Projects and implementations that exist in the literature, some of which are heavily based on the ETSI NFV-MANO:

- *Open Platform for NFV (OPNFV) [33]:* an open source project founded and hosted by the Linux foundation. It aims to be a platform used for the evolution of the NFV technology, allowing consistent interaction and ensuring a good performance among multiple components. The project follows the ETSI architecture, and simulations the VNF Infrastructure, and can therefore be used to develop applications, and evaluate their performances on it. OPNFV allows developers to deploy and test third party applications on it as well.

- *Mobile Cloud Networking (MCN) [34]:* is an association of different network operators, cloud operators, and research institutes interested in the field of NFV, with a clear objective of switching all the components of the mobile networks into the cloud. Examples of such components are the Radio Access Network (RAN), IP Multimedia Subsystem (IMS), Digital Signage (DSS), Operational Support Systems (OSS), and Business Support Systems (BSS)

- *T-NOVA: [35]* is a management and orchestration platform for the automatic provision, configuarion, and monitorying the VNFs based on the changing demands of the network (based on the traffic flowing in and out).T-NOVA allows operators to deploy VNFs based on their needs and to offer them to their customers as a value added service.

- *Zoom: [36]* is a project aimed at enabling the delivery and management of VNFs along with security features that will protect them and their infrastructure. To achieve their goals, the project conducts demos with different network operators, and vendors, producing real life simulations. The project runs simulations on different aspects, namely automated management, security, and service modeling.

- *OpenMANO: [37]* is an open source project that implements the NFV MANO framework designed by ETSI. It applies Enhanced Platform Awareness (EPA) [38] principles to address the performance and portability aspects. The project has an architecture that can be decomposed into three componts: first, a graphical user interface that offers the different services, such as the creation/deletion of VNF templates and instances. Second, an NFV-specific infrastructure manager with an openflow controller that creates network topologies. Third, the engine itself, openmano that runs the functions asked by the user on the created network topology.

- *UNIFY: [39]* develops and evaluates the different means of orchestrating and observing the service delivery through core home and small enterprise networks, to data centers, through core networks. The project aims at creating a service abstraction model that enables automatic placement of networking, computing, and storage components of the network, with the help of a orchestrators that monitors the networks and optimizes the placement of the different service components.

- *CONTENT: [40]* is a project aimed at offering a network and infrastructure architectures to allow the deployment of the mobile network on the cloud. They propose a virtualization solution that allows to create infrastructure subsets of the network and its different physical components, allows dynamic service provisioning across the network, and offers reliable QoS guarantees.

- *HP OpenNFV: [41]* is a platform upon which services and networks are dynamically built. It provides solutions to the blocks defined by ETSI. The infrastructure and VNFs are based on the HP servers and products. The management and orchestration platform is based on three components: the VNF Director, which orchestrates and manages the service and the VNFS, while also managing the resources of the network based on the network demand. The VNF manager is responsible for the VNFs themselves (i.e., their lifecycles). Finally, the Helion OpenStack platform that run the VNFs.

- *Huawei NFV Open Lab: [42]* aims at developing an NFV solutions environment and infrastructure. The lab is an open and collaborative place for expanding innovations and developing an NFV system that helps customers achieve business sucess. They also contribute to provide use-cases for multi-vendors interoperability around the infrastructure and services.

- *Intel Open Network Platform (Intel ONP): [43]* is a system to develop solutions for NFV and SDN. It is focused on Intel products (e.g., Intel Servers), and collaborative participation in open source development efforts with the industry. The Intel ONP managed to develop the Intel ONP Server, an architecture that integrates software and hardware components to optimize SDN and NFV. It allows managing the network by exposing the resource availablity, for workload balancing. Their software advancements is owed to the efforts done in community projects, in addition to the contributions provided by Intel. Examples of Software contributions include OpenStack, OpenDaylight, DPDK, Open vSwitch, and Linux KVM.

- *CloudNFV: [44]* is a platform that runs NFV, SDN and cloud computing environments. Their proposed architecture is made up of three components: active virtualization, NFV orchestrator, and NFV Manager. The active virtualization is a model that reporesents the services, functions and resources. The orchestrator contains a set of policy rules, which determines the locations of the functions as well as the connections between them. The VNF manager uses a data model and is used to manage the VNFs.

- *Alcatel-Lucent CloudBand: [45]* is a platform that implements NFV. It is made up of two levels. First, it contains the nodes that contain resources, such as storage and VMs. Second, a management system which manages the whole environment. The system behaves as a load balancer, making all the decision based on policy rules. Moreover, VNFs are deployed following also a set of policy rules, that specify the number, position and connections of the deployed VNFs.

- *Broadcom Open NFV: [46]* is a platform that accelerates the creation of NFV-based aoolication across processors, allowing system vendors to migrate VNFs between platforms regardless of the different vendor solutions. The platform supports open API standars to access the network components and their functionalities.

- *Cisco Open Network Strategy: [47]* includes an Evolved Services Platform (ESP) and an Evolved Programmable Network (EPN).The ESP and EPN contain an orchestrator, VNF manager, and SDN controller, which, altogether provide implementations for the blocks defined by ETSI in their framework. The orchestrator provides the management at the network service level. The manager provides scalability and VNF lifecycle management (e.g., creation, provisioning, and monitoring VNFs). The manager scales the VNFs (up and down) based on the traffic in the network. The SDN controller connects all the virtual resources (i.e., VNFs, Virtual Links, etc.) to the internet. It is designed around open standards and is capable to function in multi-tenant environments.

- *F5 Software Defined Application Services: [48]* provides Layer 4-7 capabilities to compute initiatives such as SDN. is is based on three components: The service platform, which has programmable control and data paths, and allows service creation. The services fabric, which provides core services (e.g., scalability, service isolation, multi-tenancy, etc). Finally, a wide range of application services.

,

2.3 Virtual Network Function Placement and Scheduling

Only little work focused on the scheduling of VNFs to process service chains. [27] provide a brief overview of the SDN and NFV technologies over optical networks. Moreover, they provide a basic formalisation model for the VNF scheduling problem, with the hopes to be used as starting point to optimally solve the scheduling problem. [28] formulate the online virtual function mapping and scheduling problem and propose a set of algorithms for solving it. they develop three greedy, and one Tabu search algorithms, and compare one with another. They consider three objectives in their work: Minimizing the flowtime, defined as the time needed to fully service a flow minus the time it arrived, the revenue, which is a function of the income generated by total amount of physical network resources that are utilized, and finally the cost, which is calculated to the total of network resources used. While [28] takes the optimization of network resources into consideration, none of these works actually account the power consumption of the network, which nowadays is major factor to be focused on.

2.4 NFV Frameworks and Architectures

ClickOS [14] is a Xen-based software platform optimized for MB processing. NetVM [17] is a virtual server platform optimized for running complex network functionality at line-speed. [25] describe a control plane architecture capable of controlling both internal NF and network forwarding states allowing an efficient reallocation of flows across NF instances. Stratos [26] is a framework that addresses the issues of scaling and composition required by the application and middleboxes. The framework comprises a network-aware placement algorithm to place and scale NFs according to the changing demands. Gember et al. [49] realized a software-defined MB networking framework that supports NFs scaling and live migration. The authors concentrate on the manipulation and controllability of the MB state and the appropriate abstractions to address the issue. [50] proposes a distributed paradigm for NFV through the integration of software defined network. Flowtags [51] is an extended SDN architecture that organizes the traffic in the network through adding Tags to outgoing packets.

2.5 VNF Orchestration and Management

Several works present mathematical models and heuristics to solve the problem of placing VNFs and routing flows through them. Lukovszki and Shmid [52] present a deterministic algorithm for Online Service Chain Embedding Problem. Bari et al. [53] deploy and orchestrate VNFs with the aim to minimize the deployment, energy and traffic forwarding costs. They used a heuristic to model the problem as a multi-stage graph, and find the optimal placement by running the Viterbi algorithm. Mehraghdam et al. [54] develop a MIQCP model for finding the best placements for the chained VNFs based on resource constraints, and tenants and operators demands. The authors in [55] model the VNF placement and routing problem as a MILP function with the aim to minimize the use of network resources and minimize the flow's end to end delay. They also develop heuristics to embed a large number of flows on large networks. [56] provide a mathematical model to determine the placement of VNFs while minimizing bandwidth consumption in the network. [57] formulate the problem as an ILP model and propose a heuristic based approach for initial NFs placement and chaining problem with the goal of minimizing number of NFs instances used in the cloud. [58] propose a heuristic that performs sequential embedding of VNFs. Their approach is straight forward with no optimality guarantees. Moens and De Turck [59] assume a hybrid network comprising both physical MBs and VNFs. [60] address the placement and routing disjointly rather in only one.

Chapter 3

Energy-Aware Placement And Scheduling of Traffic Flows on VNFs

Today, 40% of the world population is connected to the Internet as opposed to less than 1% in 1995 [61]. This exponential growth in data traffic, and the need to accommodate this pervasive access, calls for the need of network infrastructures expansion, leading to a fast escalation in power consumption [62]. Indeed, [63] presents a proof of the alarming growth in the yearly power consumption for some of the major telecom operators worldwide. Since MBs have fixed processing capacities, are deployed at specific physical locations, are function specific and come with high capital expenditure (CAPEX) and operational expenditure (OPEX), such limitations reduce the MBs' ability to efficiently serve the increasing volume of traffic flowing into the network, without an immense power consumption. NFV promises to overcome the problems caused by their hardware counterparts, by virtualizing them. This technology allows the network operators to provision VNFs dynamically, with the change of the traffic in the network, and with minimal overhead. Indeed, the number and places of the VNFs may be upgraded/degraded almost instantly, in order to avoid any over-provisioning or bottlenecks as the shape of the traffic varies in the network.

One of the important challenges brought by NFV is the placement and scheduling of the traffic flows on these VNFs. This flow-to-VNF scheduling problem has received only very little attention [27], and [28]. The main purpose of the existing work is to minimize the makespan (i.e. the minimum schedule to serve all the

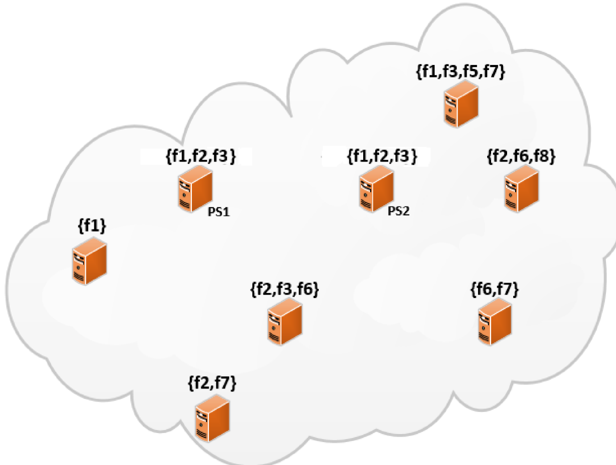


Figure 3: Cloud Network Representation.

flows). Recently, however, studies [29], [30] have shown that an idle server consumes power approximately 70% of that consumed by that same server running at full power. As a result, ETSI stresses on the need of a NFV framework able to optimize energy consumption on demand whether by scheduling and placing VNF instances on specific resources or placing unused resources in energy saving modes [31].

In this chapter, we formulate the problem of placing and scheduling traffic flows on VNFs. Placing flows to VNFs consists of assigning a function of a flow to a specific VNF. Scheduling flows on the other hand allocates a time slot for each of the flows enqueued on a VNF in order to be processed. We aim to provide a trade-off between minimizing the makespan of the schedule and reducing the power consumed. We also formulate the problem as an ILP model. Given that the service function chain scheduling problem resembles the job-shop problem [27], and thus no polynomial-time solution exists, we develop a heuristic to overcome the complexity of the original problem.

3.1 System Model

3.1.1 Network Model

Consider a Substrate Network (SN) consisting of a set K of Physical Servers (PS). Each PS has a power consumption, which can be estimated as follows [64]: let e_k^δ be the total power consumption of PS $k (k \in K)$ during one timeslot δ. e_k^δ is a function of a baseline power e_b (The power required to turn on the server k

without any load on it), an energy proportion factor e_p, and finally a CPU load u_k^δ on that server. u_k^δ is a function of the total CPU capacity taken by the Virtual Machines (VMs) that are hosted on a PS k. Finally, we divide the timeslot T into timeslots δ of equal length. The total power consumption of a server k can be estimated by the following:

$$e_k = \sum_{\delta} e_b + (e_p * u_k^\delta) \quad \forall \quad k \in K. \tag{1}$$

Due to the variation of the workload, the total power consumption of a PS can be defined as an integral of the power consumption over a period of time [29]. Therefore, the total power consumption of a PS is the summation of its total power consumption at each time slot:

$$e_k = \sum_{\delta} e_k^\delta \quad \forall \quad k \in K. \tag{2}$$

Denote by F_k the set of VNFs, hosted on PS k. Each of the VNFs on PS k provides a distinct functionality (i.e., IDS, NAT, etc). We assume that a PS may host at most one VNF of each type. Thus, let w_f^k denote the type of a VNF hosted on PS k. Typically, $w_f^k = 1$ if a VNF of type f is hosted on k, and 0 otherwise.

Let S be a set of traffic flows. A flow $s_i (i \in S)$ must be processed by a sequence of VNFs F_i in proper order, and without violating a deadline d_i. f_{ij} represents the j^{th} function in the sequence F_i of network flow s_i. Similar to the VNFs, for every flow s_i, each f_{ij} belongs to a distinct type t_{ij}. As mentioned previously, the order of processing through the chain of a flow must be respected. For instance, f_{ij} may not begin its processing before its previous $f_{i(j-1)}$ finishes its processing. Each f_{ij} is characterized by a processing time p_{ij}. A VNF hosted on k may only service one traffic flow at a time. Therefore, a VNF is considered to be in an active state if and only if it is servicing at least one service s_i. Similar to [53], a PS is in an active state if and only if at least it is hosting a VNF in the active state. Otherwise, it is assumed to be in an idle state. A PS, which does not have (or no longer has) any flow scheduled to be serviced on it may be switched to a sleep mode (i.e. a low power state that can quickly return into the active state [65]). we assume that a PS in sleep mode does not consume power at all.

In this work, we focus our attention on the problem of mapping and scheduling a set of network flows on VNFs, with the aim of minimizing the power consumption in the network. Similar to [28], and owing to the complexity of the problem, we do not solve the routing sub-problem. We rather assume that the underlying network has enough capacity to route the traffic of the corresponding network services.

3.1.2 Motivation

Consider the network represented in Figure (3). Out of all the PSs in the Figure, this example is concerned in two PSs PS_1, and PS_2, each hosting the following set of VNFs: $<f_1, f_2, f_3>$ (Each number in the set represents the type of the VNF). For the sake of illustration, assume that each PS has a baseline power $e_b = 7$ Units of power (U) per timeslot, and an energy proportion factor $e_p = 1\ U$ per timeslot per VM. For simplicity reasons, the values of e_b and e_p are not realistic. However, considering that a PS can host at most three functions, the values respect the fact that an idle servers consumes power 70% of its power consumption running at full power [29].

Consider two traffic flows s_1, and s_2 arriving at the same time and need to be placed and scheduled on the available servers. As mentioned previously, each flow $s_i (i \in S)$ has a sequence of network functions F_i, and a deadline d_i. Each function f_{ij} has a type t_{ij}, a processing time p_{ij}. The flows are characterized as follows:

- $s_1 : F_1 = <f_{11}, f_{12}, f_{13}>$, $t_{1j} = <1, 2, 3>$, $p_{1j} = <1, 1, 1>$, $d_1 = 8$.
- $s_2 : F_2 = <f_{21}, f_{22}, f_{23}>$, $t_{2j} = <1, 2, 3>$, $p_{2j} = <2, 2, 2>$, $d_2 = 6$.

There exists two direct approaches to place and schedule these flows in the network; the first one aims at minimizing the makespan[1] of each PS. Indeed, splitting the flows on all the available resources in the network permits the processing of the maximum number at the same time, thus reducing both, the time spent by a service in the network, and the time a server is turned on. Figure 4(a) presents the schedule resulting from such a placement. s_1 alone is processed on PS_1, s_2 is processed on PS_2. As a result, the

[1]The total duration during which a Physical Server is running in active or idle modes.

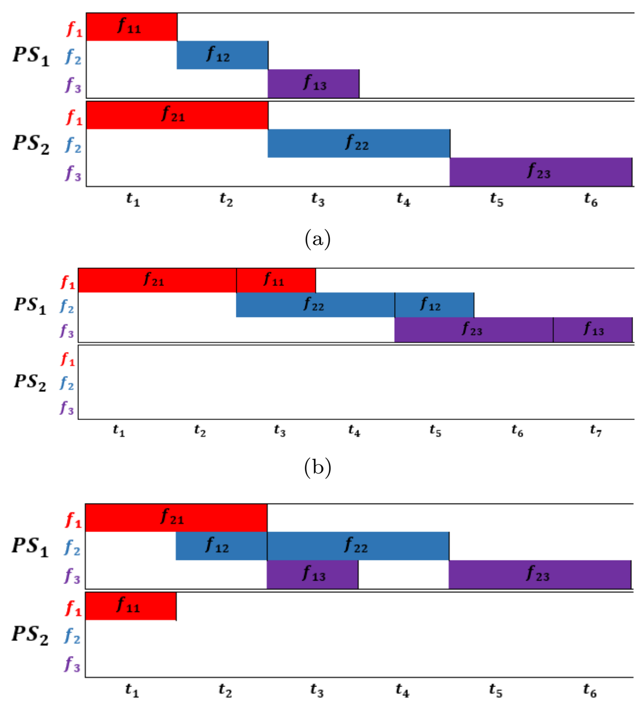
makespan of PS_1 is 3 timeslots, and that of PS_2 is only 6 timeslots. The overall power consumption of this network is calculated (using Equation (1)) to be **72** U.

Alternatively, another approach would be to consolidate the flows as much as possible on the minimum number of servers. This permits fewer servers to be turned on, and thus save energy. However, the flows might suffer from a longer waiting time, before being admitted into service. Figure 4(b) depicts the resulting schedule. Indeed, both flows are placed on PS_1 and PS_2 is switched to sleep mode. Moreover, the total power consumption of the network is reduced from **72** U to only **58** U, a reduction of **19.4%**. However, one can notice that the makespan of PS_2 in this case is one timeslot larger than that in the first scenario. More importantly, since both functions of type 1 were scheduled on the same VNF, and as mentioned previously, a VNF can only service one request at a time, s_1 had to wait two timeslots before being serviced. This waiting time for s_1 resulted in extending the makespan. While in this example, the deadline of s_1 is met, such a waiting time may not always be tolerated, especially in larger networks with a larger number of flows, some of which may have stricter deadlines.

In summary, consolidation seems to be a great method to minimize the total energy consumption, by activating the minimum number of server needed. However, this reduction in the number of used servers may not be able to accommodate all the flows, and thus violate the Service Level Agreement (SLA) [66], resulting in penalties for the network operator. Nonetheless, allocating the services on all available resources will reduce the chances of SLA violations, as well as the waiting time of the services, but at the expense of increased power consumption. In light of the above, we argue that a tradeoff between these two extremes exist.

Figure 4(c) illustrates that, while consolidation is a great approach to save energy, there exists better alternatives. Observe that by placing f_{11} on PS_2, and alternating the schedule of the remaining functions on PS_1, the original schedule shown in Figure 4(b) is reduced by one timeslot, and the power consumption remained **58** U (The same power consumed in scenario 2). More importantly, s_1 was admitted and processed immediately, and did not suffer from a waiting time that could have violated any SLA. Evidently, this tradeoff combined the advantages of the two aforementioned approaches, and reduced their disadvantages.

Scaling these examples into larger and more realistic networks, comprising large numbers of PSs, hosting



(a) (b) (c)

Figure 4: Schedules of Different Placement and Scheduling Scenarios

hundreds or thousands of VNFs, and serving numerous flows, highlights further the need of such an energy aware traffic placement and scheduling framework. As a conclusion, a placement and scheduling of flows in the network, without taking into consideration the power consumption of the servers may lead to an excessive power consumption, which can dramatically increase the cost for the network operator. Therefore, a framework capable of providing an efficient placement and scheduling of the flows, while ensuring minimal power consumption at the same time is a necessity.

3.1.3 Problem Statement

Assume a set of PS, each of which is hosting several VMs, and has a power consumption computed in Equation (2). Every VM runs a single VNF functionality. Given, a set of network traffic flows, where each one possesses a series/sequence of functions to be processed on (in order), find the most power efficient schedule to service all these flows, without exceeding their deadlines, while respecting the network resource constraints (i.e., VM capacity, processing speed, etc.). Indeed, scheduling the requests on the servers, while minimizing the power consumption in the network, and shutting the unused servers will significantly decrease the total cost of the network, and thus generate a higher revenue for the operator.

3.1.4 Problem Formulation

The schedule in this paper is defined as a series of decisions, placing and scheduling flows on the already placed VNFs, in order to process them, while minimizing the power consumption. Let x_{ij}^{δ} and v_{ij}^{δ} be two variables representing the timeslots at which the j^{th} function of traffic flow s_i began, and finished its processing respectively. Finally, denote by $y_{ij}^{k\delta}$ be a binary variable describing the placement of function f_{ij} on PS k at timeslot δ. The main objective is to minimize the power consumption of the PSs. The power consumption of each PS k depends on its makespan C_k, that is the total number of timeslots the server is on. The objective function can be mathematically formulated as follows:

- **Parameters:**

δ : Timeslot.

S : Set of traffic flows.

d_i : Deadline of flow s_i.

F_i : Set of functions of flow s_i.

p_{ij} : Processing time of the j^{th} function of flow s_i.

K : Set of Physical Servers.

e_b : Baseline power.

e_p : Energy proportion factor.

F_k : Set of VNFs placed on PS k.

t_{ij} : Type of the j^{th} function of flow s_i.

$$w_f^k = \begin{cases} 1, & \text{if PS } k \text{ hosts a VNF of type } f, \\ 0, & \text{otherwise.} \end{cases}$$

- **Decision Variables:**

v_{ij} : The processing ending time of the j^{th} function of flow s_i.

C_k : Makespan of PS k.

u_k^δ: The load on server k at timeslot δ.

$$x_{ij}^\delta = \begin{cases} 1, & \text{if function } f_{ij} \text{ began processing at timeslot } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

$$r_{ij}^{k\delta} = \begin{cases} 1, & \text{if function } f_{ij} \text{ began processing on server } k \text{ at timeslot } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{ij}^{k\delta} = \begin{cases} 1, & \text{if PS } k \text{ is servicing function } f_{ij} \text{ at timeslot } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

$$z_{ij}^{k} = \begin{cases} 1, & \text{if PS } k \text{ is servicing function } f_{ij}, \\ 0, & \text{otherwise.} \end{cases}$$

- **Mathematical Model:**

$$Min \quad \sum_k \left(C_k * e_b + \sum_\delta e_p * u_k^\delta \right) \tag{3}$$

subject to

$$C_k \geq \sum_{\delta \in T} r_{ij}^{k\delta} * (p_{ij} - 1 + \delta) \quad \forall\ i \in S, j \in F_i, k \in K \tag{4}$$

Constraint (4) calculates the makespan of each server.

$$r_{ij}^{k\delta} = x_{ij}^\delta * z_{ij}^k \quad \forall\ i \in S, j \in F_i, k \in K, \delta \in T \tag{5}$$

Constraint (5) is the product of two binary variables. Clearly, it is nonlinear. In order to translate it into an ILP format, we replace it with the following three constraints:

$$r_{ij}^{k\delta} \leq x_{ij}^\delta \quad \forall\ i \in S, j \in F_i, k \in K, \delta \in T \tag{6}$$

$$r_{ij}^{k\delta} \leq z_{ij}^{k} \quad \forall\, i \in S, j \in F_i, k \in K, \delta \in T \tag{7}$$

$$r_{ij}^{k\delta} \geq x_{ij}^{\delta} + z_{ij}^{k} - 1 \quad \forall\, i \in S, j \in F_i, k \in K, \delta \in T \tag{8}$$

Indeed, Constraints (6), (7), and (8) altogether represent the linearized version of constraint (5), confining the ILP format.

$$y_{ij}^{k\delta} \leq z_{ij}^{k} \quad \forall\, i \in S, j \in F_i, k \in K, \delta \in T \tag{9}$$

$$\sum_{k} z_{ij}^{k} = 1 \quad \forall\, i \in S, j \in F_i \tag{10}$$

$$z_{ij}^{k} \leq w_{t_{ij}}^{k} \quad \forall\, i \in S, j \in F_i, k \in K \tag{11}$$

$$\begin{gathered} y_{ij}^{k\delta} + y_{i'j'}^{k\delta} \leq 1 \;\; if\; t_{ij} = t_{i'j'} \\ \forall\, (i, i') \in S, j \in F_i, j' \in F_{i'}, k \in K, \delta \in T \end{gathered} \tag{12}$$

$$\begin{gathered} \sum_{k \in K} y_{ij}^{k\delta'} \geq x_{ij}^{\delta} \forall\, i \in S, j \in F_i, k \in K, \\ \delta' \in T \mid \delta \leq \delta' \leq \delta + p_{ij} - 1 \end{gathered} \tag{13}$$

The constraints above deal with the placement of the flows in the network. Constraint (9), and (10) ensure that a function f_{ij} is only placed on one PS and one PS only. Further, Constraint (11) dictates that a function f_{ij} may be placed on a PS if and only if that PS has a VNF that is able to process the function (i.e., of the same type). Constraint (12) allows only one function, during a particular timeslot δ, to be placed on a VNF. That is, no two functions of the same type may be placed on the same VNF during the same timeslot. Constraint (13) guarantees that a function is placed on a VNF during its whole processing time.

$$v_{ij} = \sum_{\delta \in T} (\delta + p_{ij}) * x_{ij}^{\delta} \quad \forall\, i \in S, j \in F_i, \tag{14}$$

$$v_{iJ} \leq d_i \;\; where \; J = |F_i| \quad \forall\, i \in S \tag{15}$$

$$v_{ij} > v_{i(j-1)} + p_{ij} \quad \forall\, i \in S, j \in F_i \tag{16}$$

$$\sum_{\delta \in T} x_{ij}^{\delta} = 1 \quad \forall\, i \in S, j \in F_i \tag{17}$$

The constraints above handle the scheduling part of the problem. Indeed, Constraint (14) determines the timeslot at which a function has finished its processing. Constraint (15) ensures that the placement and scheduling of the functions does not exceed the deadline of the service. Constraint (16) denies a function to begin its processing before its preceding function in the chain finishes its own processing. Constraint (17) guarantees that each function is placed and started its processing.

$$u_k^{\delta} = \sum_{i \in S} \sum_{j \in F_i} y_{ij}^{k\delta} \quad \forall\, k \in K, \delta \in T \tag{18}$$

Finally, Constraint (18) calculates the load on the server at each timeslot (i.e., the number of active VNFs).

3.1.5 Model Scalability

In this section, we test the scalability of the ILP model. We first run it on small scale networks, and with a small number of services. Table 1 shows that the model generates optimal solutions, without consuming too much time. However, as we increase the scale of the network, and the number of the services, the model's ability to find optimal solutions in polynomial time decreases dramatically. Therefore, we can conclude that

Input	2 PS, 3 Flows	4 PS, 5 Flows	4 PS, 8 Flows	6 PS, 12 Flows	8 PS, 16 Flows
runtime	$\approxeq 0s$	4.87 seconds	9.78 seconds	16.47 minutes	>23 hours
Solution	optimal	optimal	optimal	optimal	34.12% Gap

Table 1: Scalability Results of the ILP Model.

the ILP model is not scalable, and therefore, we need to come up with algorithmic solutions to bypass this issue.

3.2 Algorithmic Solution

In Section 1, we show that the ILP Model is non-scalable. Therefore, we resort to an algorithm to overcome this problem. The algorithm aims at placing and scheduling as many services as possible, while minimizing the power consumption of the network. In this regard, the algorithm (presented in Algorithm 1) is a tradeoff between two extremes: minimizing the makespan, and consolidation. The former ensures the smallest schedule, at the expense of a considerable power consumption. The latter, on the other hand, uses the least amount of servers at the expense of not meeting the services' deadlines. Both approaches have great advantages, yet more serious disadvantages. The algorithm combines their advantages, and avoids their disadvantages.

Algorithm 1 Energy Aware Consolidation(K, S).

Sort flows according to the deadline
for $s_i \in S$ **do**
    for $f_{ij} \in F_i$ **do**
        initialize candidate node set $K' = \phi$
        $K' = getCandidateNodes()$;
        if $(K' == null)$ **then**
            Reject s_i
        else
            Select the best PS k from K'
            Place f_{ij} on k
        end if
    end for
    checkDeadline();
    if (solution exceeds deadline) **then**
        reduceGap(s_i, K);
    end if
end for

Algorithm 2 Reduce Gap(s_i, K).

Sort flows according to the deadline
while !(solution meets deadline) **do**
  sort f_{ij} based on longest waiting time
  for $f_{ij} \in F_i$ **do**
    find best candidate PS k in sleep mode
    if ($k ==$ null) **then**
      reject s_i
    else
      Place f_{ij} on k
    end if
  end for
  checkDeadline();
end while

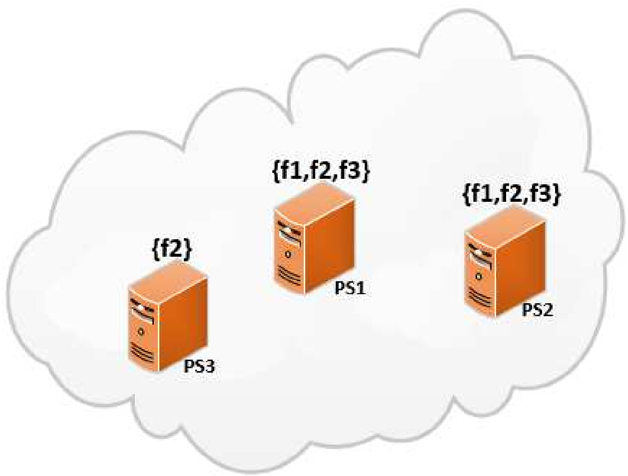


Figure 5: Small Network State Illustration.

3.2.1 Algorithm Explanation

In this section, we explain the functionality of our algorithm (also presented in Algorithm 1), which is given the name of *Energy Aware Consolidation*.

For each function j of flow s_i (denoted by f_{ij}), the algorithm searches for all the PSs, that are in active mode, and that can serve this function (i.e., the PS is hosting a VNF of the same type as that of f_{ij}). The PSs found are sorted in an increasing order, based on the availability time a_k of this function of the PS (e.g., the time at which this VNF becomes available). If no PSs are found, the algorithm selects a PS that is in sleeping mode, provided it is hosting a VNF of this type. A preference is given to the PS capable of hosting more functions; that is, if two PSs in sleep mode, are found to be able to serve a particular function, the one who has the largest number of VNFs hosted on it is picked first. if no PS in sleep mode, that is hosting a VNF of this type, is found, the flow is rejected. Once the last function of a service is successfully placed and scheduled, the algorithm checks if the deadline is met or not. If it was not, an alternative placement should be made. In order to do so, the algorithm calculates the delay between each pair of functions (i.e., the finishing time of f_{ij} and the starting time of $f_{i(j+1)}$). Indeed, a delay, if present, is a result of a busy PS. Therefore, a new PS must be found for this function. The algorithm picks the function with the largest delay, finds a new PS, from the pool of PSs in sleep mode and that have the appropriate VNF, and schedules the function on it. Moreover, the functions following that with the largest delay are also replaced and scheduled. The algorithm keeps on re-checking the deadline, until either the deadline is met, or no new server is found for a function that needs to be re-placed. In the former case, the flow is admitted, and rejected in the latter.

3.2.2 Benchmarks

In order to properly evaluate the efficiency of our proposed solution, we compare it two greedy, energy oblivious algorithms for the placement of the service functions.

Minimize Flowtime

Algorithm 3 Minimize Flowtime(K, S).

```
for s_i ∈ S do
    for f_ij ∈ F_i do
        initialize candidate node set K' = φ
        K' = getCandidateNodes();
        if (K' == null) then
            Reject s_i
        else
            Select the best PS k from K'
            Place f_ij on k
        end if
    end for
    checkDeadline();
    if (solution exceeds deadline) then
        reject s_i
    end if
end for
```

Algorithm 3 aims at minimizing the makespan of the PSs. In order to do so, for each service s_i, we try to place every f_{ij} on the PS whose function has the best availability time a_k. We do not distinguish between servers in active or sleeping modes. If, for any s_i, the deadline is not met, the service is directly dropped.

Evidently, with this method, the number of flows admitted will be maximized, and the makespan of each PS will be minimized. However, this method will generate a high power consumption, due to the utilization of the biggest number of PSs possible.

Consolidation

Our algorithm is effectively the same as consolidation. The only difference is that our algorithm, when searching for a PS in sleeping mode, gives priority to PSs with the largest number of VNFs placed on them. The regular consolidation method doesn't perform this step.

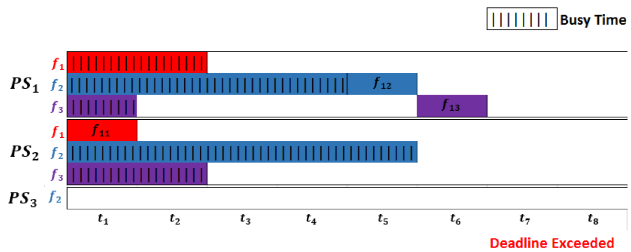
Indeed, our algorithm mainly relies on consolidation. However, consolidation alone proves that it can have disastrous results, especially in networks where the servers do not have a large number of VNFs placed on them. In fact, the only difference between our algorithm and the consolidation method, is that, when searching for a PS in a sleeping mode, preference is given to that with the biggest number of VNFs placed on them. This step alone reduces, as can be seen in Section 3.3, both the average number of servers used, and the average power consumption of the network. Indeed, this small, yet important step will allow to take advantage of the PSs with the highest numbers of VNFs placed on them, thus accommodating more service functions on a less number of servers.

3.2.3 Algorithms in Action

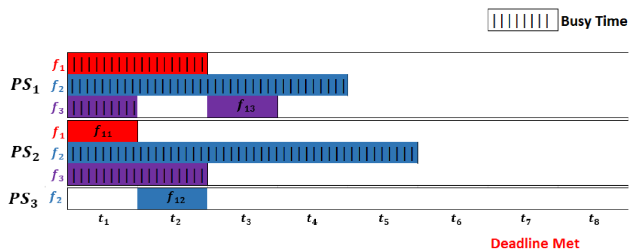
In order to better understand how the algorithm and the benchmarks work, we provide, an example of a small network (Figure 5), in which we will use these algorithm to place and schedule a flow. In this example, we assume a number of PSs, each of which has a set of VNFs placed on them, some of which are in active mode, with flows already scheduled on them, and others in sleeping mode.

Consider a network with the following PSs:

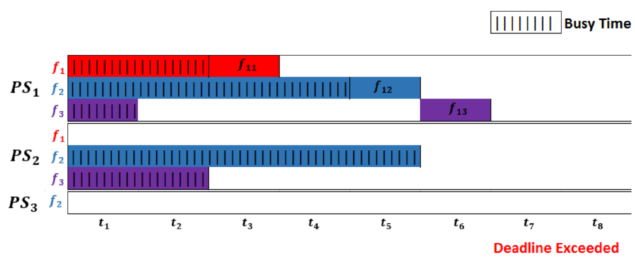
- PS_1: hosts functions $F_1 = <f1, f2, f3>$, with availability times $a_k = <2, 4, 1>$.
- PS_2: hosts functions $F_2 = <f1, f2, f3>$, with availability times $a_k = <0, 5, 2>$.
- PS_3: hosts functions $F_3 = <f_2>$. This PS is in sleeping mode.



(a)



(b)



(c)

Figure 6: Schedules Produced by Different Algorithms.

Assume a new flow s_1 with the following characteristics:

- s_1: $F_1 = <f_{11}, f_{12}, f_{13}>$, $t_{1j} = <1, 2, 3>$, $p_{1j} = <1, 1, 1>$, $d_1 = 4$.

We begin by explaining how our algorithm works. As explained in Section 3.2.1, we iterate on every function of the flow. We begin by searching for the set of PSs (in active mode) that can service each function, sort them based on their availability times, and place and schedule the function on it. By doing so, we place f_{11} on PS_2, and both f_{12} and f_{13} on PS_1. While this placement is legitimate, one can notice, in Figure 6(a), that the deadline is not met. With this placement, it takes 6 timeslots to fully process s_1. This delay is caused by function f_2 of PS_1. In fact, f_{12} has to wait 3 timeslots, after the finish time of f_{11}, before being processed. In order to overcome this problem, we switch PS_3 into active mode, and place f_{12} on it. The new schedule, shown in Figure 6(b), surely meets the deadline, since now the total processing time of s_1 decreased to 3 timeslots, which is less than d_1. Note that if the network did not contain another PS able to service f_{12}, for example PS_3, the service would have been rejected.

Next, we describe the functionality of the *Minimize Flowtime* algorithm (Alg. 3). For each function, the algorithm finds the PS with the corresponding VNF, and that has the best availability time. The algorithm does not care, nor does it give any priority between PSs in active or sleep mode. Figure 6(b) presents the schedule resulted from this algorithm. note that the schedule is the same as that of our algorithm in this example. However, this is not always the case, since our algorithm is a tradeoff between Consolidation, and minimizing the makespan.

Finally, we present the consolidation algorithm. For each function, the algorithm searches for any server in active mode, and that can host the function. Beginning by PS_1, the algorithm notices that the PS can host all three functions of s_1. In this scenario, the deadline is not met (Figure 6(c)). Therefore, the flow is directly rejected. Even though there are other PSs that can accommodate the functions of s_1, the algorithm, with its greedy nature tries to find the first PS that can host each function. If at the end, the deadline of the flow is not met, it is directly rejected.

	Model		Minimize Flowtime		Consolidation		Enhanced Consolidation	
Input	Runtime	Power	Runtime	Power	Runtime	Power	Runtime	Power
2 PS, 5 Flows	0.62s	424	$\approxeq 0s$	599	$\approxeq 0s$	459	$\approxeq 0s$	459
3 PS, 6 Flows	0.52s	627	$\approxeq 0s$	860	$\approxeq 0s$	947	$\approxeq 0s$	714
4 PS, 8 Flows	84s	705	$\approxeq 0s$	1165	$\approxeq 0s$	1230	$\approxeq 0s$	990

Table 2: Comparison Between the Model and the Algorithms

3.3 Numerical Results

3.3.1 Model vs algorithms

Table 2 evaluates the results of the mathematical model, our proposed solution, and the two benchmarks, on small scale networks. The comparison consists of the runtime, and the average power consumption. Clearly, none of the three algorithms provide the optimal solution, that is generated by the mathematical model. However, several important observations can be extracted from the table:

- Minimizing the makespan leads to an excessive power consumption of the network. Indeed, as explained previously, the algorithm tends to use as many resources available as possible, in order to minimize the service time of the flows. the advantage of such algorithm is that it accelerates the servicing of the flows, and thus decreasing the chance of any SLA violation.

- Energy Oblivious Consolidation is not enough. Indeed, the results show that the power consumed by the consolidation is higher than that produced by the algorithm that minimizes the makespan. The only case where regular consolidation saves energy is when all the PSs host all the VNFs. However, in the other case, regular consolidation performs poorly. A logical explanation to these results exists. As explained, consolidation involves using the smallest number of servers needed. this means that even though a smaller number of servers is used, the makespan of each server is increased, and this also increases the power consumption.

- Energy Aware Consolidation is needed. This method also tries to minimize the number of PSs needed. It does so, by consolidating the functions on the smallest number of PSs. However, this method prioritizes the PSs with the largest number of functions on them. This reduces the number of PSs needed even more than the energy oblivious consolidation. Therefore, even though the makespans of the PSs here are also long, the reduced number of PSs used compensates for these long makespans.

The results in Table 2 support these observations. Indeed, the power consumption generated by the energy aware consolidation algorithm is the closest to that of the mathematical model, and it is much less than that generated by both the energy oblivious consolidation and makespan minimization algorithms.

Next, we test our proposed solution on large scale networks:

3.3.2 Compared Algorithms

Simulation Environment

In order to test our proposed solution, we have implemented a discrete event simulator using JAVA. The simulation was ran on a Windows 7 64-bit operating system machine with an 2.67GHz Intel Xeon CPU and 4GB Installed RAM.

The results prove that a framework that places and schedules services with the sole purpose of minimizing the makespan may induce a higher cost on the network operators. In fact, our solution, as will be seen, is very efficient in saving energy. In our simulations, we analyse the power consumption, admission rate, average server utilization, and runtime. We test these metrics by varying the number of services to be admitted, the average number of VNFs on a PS,the network size, and the average deadline of a fixed number of services.

Simulation Parameters

We define a total of 10 different network functions. Each flow may hold between 5 to 10 functions. Each function has a processing time between 1 and 5 timeslots. The deadline of each service is set to be between 25 and 35 units of time, in addition to the total processing times of the service functions. The aforementioned parameters are generated using a uniform distribution. Each point on the graph is the average of 10 runs. Table 3 summarizes the different values considered for each parameter. Similar to [29], we choose the maximum power of a PS to be 250W per timeslot. Therefore, the baseline power, which represents 70% of the PS running at maximum load, has a value 175W per timeslot. Considering the fact that a PS may hold at most 10 functions, we then assume that the energy consumed of each function running at a particular

Parameter	Values
Nodes	100
Number of Flows	10-1500
Functions	10
Number of VNFs on a PS	2-9
Functions per service	5-10
Function processing time	1-5
Service deadline	25-35
Baseline power	250
Energy proportion power	7.5

Table 3: Parameters and Values.

timeslot to be 7.5W. Since we are solving the offline problem, we assume that all of the services that need to be placed and scheduled arrive together at the same time.

Performance Metrics

We evaluate our proposed solution (*EACons*) and compare it with the one that minimizes the makespan (*mintime*) and the regular consolidation algorithm (*EOCons*) based on four performance metrics: The average power consumption, the average server utilization, the admission rate, and the runtime.

- **Average Power Consumption:** The goal of this paper is to minimize the network's power consumption. Evidently, neither minimizing the makespan, nor minimizing the number of servers used contributes alone in minimizing the power consumption. The power consumption is affected by several factors, namely, the number of PSs turned on, the amount of time each is turned on, the placement and schedule of service functions on PSs, etc. The average power consumption is the total power consumption divided by the total number of admitted flows.

- **Average Server Utilization:** We evaluate the average number of servers used to admit a certain batch of flows. typically, mintime is expected to be using the largest number of servers. However, this metric serves to test the difference in the number of servers used by EOCons and EACons, since only one of them accounts for the number of VNFs on each PS. The average server utilization is the total number of used server divided by the total number of admitted flows.

- **Admission Rate:** Maximizing the number of services admitted into the network is an important factor in determining the effectiveness of the solution. It is important to maximize the number of flows accepted. Moreover, a maximized admission rate reflects the ability of the solution in optimize the network resource consumption. The admission rate is the total number of admitted flows divided by the total number of flows.

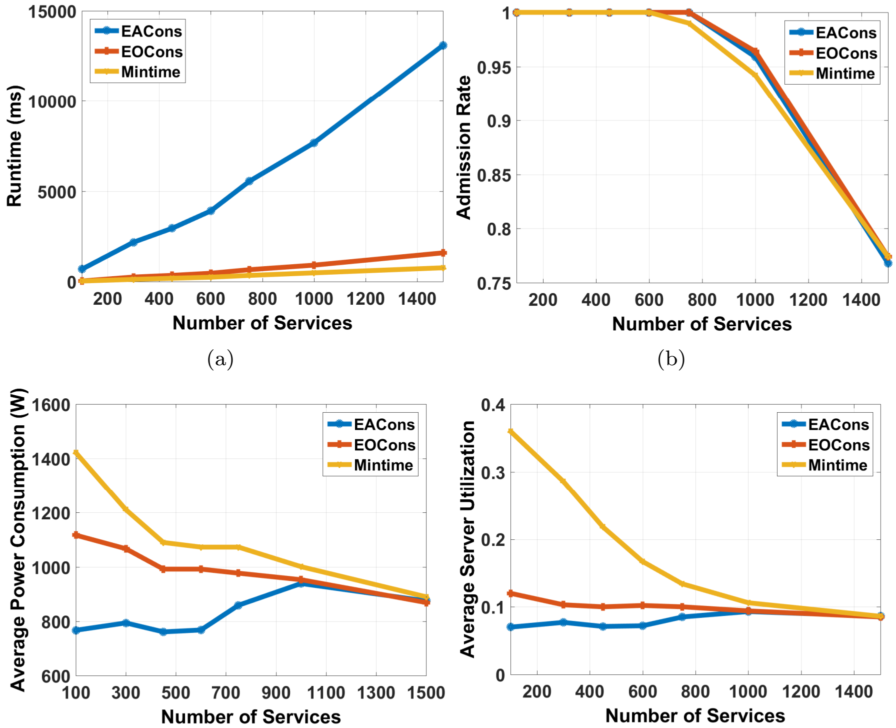
- **Runtime:** Placing and scheduling service flows in a datacenter comprising a large number of PSs, each of which has a different set of VNFs, while optimizing network resources, and respecting the service flows deadlines, is not an easy task to do. However, an efficient framework must be able to cope with completing such a task, for a large number of flows in a matter of seconds.

Simulation Results

We test the algorithms in two different scenarios: varying the number of flows, and varying the average deadline for the network flows. Each of these scenarios will highlight the advantages of our proposed solution.

Varying the number of services allows us to monitor the network's behaviour and performance in managing different loads of services. In fact, varying the load on the network tests the ability of each algorithm to place and schedule the flows, in a way to optimize the server utilization, network power consumption, and admission rate.

Figure (7) depicts the results of the three algorithms while varying the workload. Our proposed solution outperforms the other benchmarks in every aspect except for the runtime. Indeed, *mintime* handles 1500 flows in less than a second, while *EOCons* does it in 1.5 seconds. On the other hand, our solution, *EACons* requires around 13 seconds to perform the same task (Figure (7(a))). This extra time is due to the fact that *EACons* is not as greedy as the other two algorithms. In fact, not only does it search for the PSs with the best availability, but it also searches for the PSs with the most VNFs on them, in order to optimize the network resources. Moreover, handling 1500 flows in 13 seconds is considered rapid in such a large datacenter and large amount of flows. All three algorithms have the same admission rate Figure (7(b)), proving that all three admit as many flows as possible. The rate begins to drop as the number of flows increases, leaving no place to schedule any other functions on servers and meet the deadlines of the flows. Figures (7(c)) and (7(d)) show the results of the average power consumption and average server utilization respectively. The results highlight the importance of our proposed solution. Indeed, the results of all three algorithms



(a) (b) (c) (d)

Figure 7: Network Performance in Function of a Variable Workload.

come closer to one another when the network becomes congested. This is essentially logical since when the network is congested, all the network resources are used. However, one can notice the importance of our solution when the network is not congested. Logically, *mintime*, which aims to minimize the makespan, uses the largest number of servers and consumes energy the most. More importantly, however, is the comparison between *EACons* and *EOCons*. The former, by prioritizing servers with the largest number of VNFs proved its effectiveness by reducing the number of servers used and the average power consumption in the network. Indeed, consolidation alone may reduce the energy consumption, and definitely reduces the number of servers used, by stacking as many functions on PSs as possible, however, distinguishing the functions on the PSs allowed to greatly reduce the number of PSs used, and therefore, the average power consumption of the network.

To further test our solution, we fix the number of flows to 1500, and vary the minimum deadline of a flow. This allows us to analyze the advantages of our solution on flows with loose or no deadlines, and test its effects on the network.

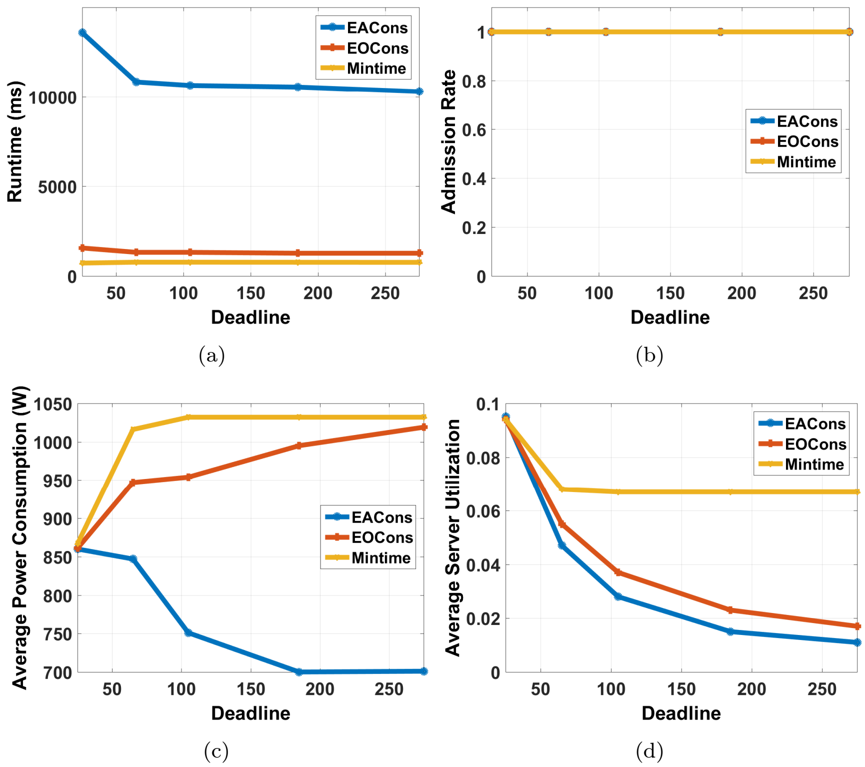


Figure 8: Network Performance in Function of a Variable Deadline.

Figure (8) shows the performance of the network as the deadline of the services changes. As explained above, the acceptance rate is the same for all three algorithms, and the *EACons* still requires more time to admit the flows. Moreover, the results of the average power consumption and average server utilization shown in Figures (8(c)) and (8(d)) respectively further strengthen the importance of our solution. When the deadline is strict, both the average server utilization and the average power consumption is almost similar for all three algorithms. This is essentially due to the congestion of the network and the unavailability of better placements to schedule the numerous functions on. However, as the deadline becomes more loose, one can notice that the average power consumption remains almost steady and only drops a bit with the *EACons*, and the average server utilization drops down. On the other hand even though the average server utilization with the *EACons* also drops, with the increase of the deadline, it is still higher than that of *EOCons*. This proves that, not only is the *EACons* using less PSs, but also saving more energy.

3.4 Conclusion

Network Function Virtualization is regarded as a promising technology to overcome the problems imposed by the hardware MiddleBoxes. Despite their numerous advantages, the virtual network functions come with quite a few challenges that need to be addressed, in order to better benefit from them. One important issue is finding the optimal placement and scheduling of the network services on these functions.

In this chapter, we solve this problem, by formulating an ILP model that tries to optimally place and schedule network services on virtual network functions, while also optimizing the total network power consumption. Due to the non-scalability of the ILP model, we design an algorithm to solve the problem on large scale networks.

Chapter 4

Placement, Scheduling, and Routing of Traffic Flows on VNFs

As mentioned in previous sections in the thesis, the main goal of the existing solutions to the placement and scheduling of flows on VNFs problem is to minimize the makespan while admitting as many flows as possible. These solutions aim to schedule and service the arriving traffic flow, in a way to optimize the usage of the available, and limited, network resources. However, the authors do not consider the bandwidth consumption on the physical links, nor do they consider transmission delays. In fact, all the previous work assumed that each pair of VNF is interconnected by a virtual link, and that a traffic flow is transmitted from one VNF to another without any transmission delay. Moreover, the previous work completely disregarded the underlying physical network (i.e., the physical path connecting a pair of VNFs located on different servers).

This chapter is concerned with the placement and scheduling of traffic flows on VNFs problem, and proposes to jointly address the problem of policy-aware traffic steering (while considering transmission delays) in the underlying physical network, with the aim of maximizing the number of flows admitted. We solve the problem by formulating an ILP model. We propose a Cut-and-Solve based approach to overcome the non-scalability of the problem. This approach consists of decomposing the ILP into two subproblems: a master and subproblem. The former handles the placement and scheduling of the traffic flow on the corresponding VNFs, while the latter performs the policy-aware routing. At every iteration, consecuting piercing cuts

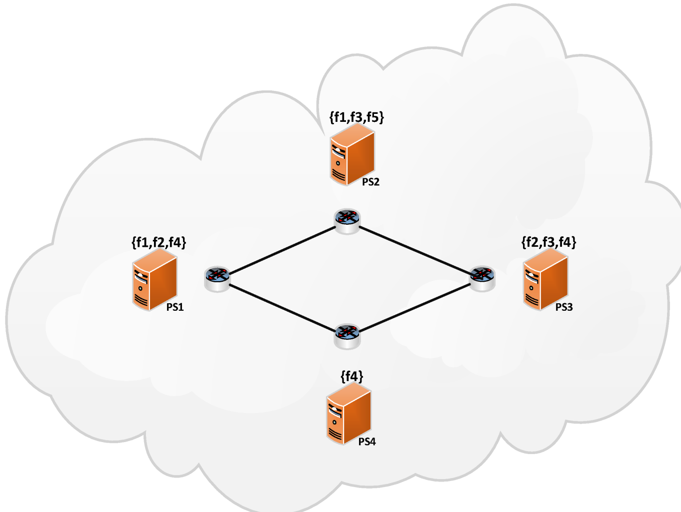


Figure 9: Small Network Example.

are generated and sent to the master in order to approach further to the optimal solution. However, the Cut-and-Solve approach is out of the scope of this thesis.

4.1 The VNF Scheduling Problem

4.1.1 Network Model

In this section, we describe the different components of the in order to formally define the VNF scheduling problem.

1. **The Substrate Network:**

   We represent the substrate network as an undirected graph, denoted by $G^s = (K, L)$, where K is the set of substrate nodes, and L is the set of substrate links. Each node $k \in K$ may host one or a set of VNFs, and each link $l \in L$, has a finite capacity b_l.

2. **Virtual Network Functions:**

   The VNFs represent the different types of functions in the network. In this work, we assume that the VNFs are already placed on the PSs. Therefore let w_f^k denote the type of VNFs placed on server k. Typically, $w_f^k = 1$ if VNF of type f is placed on server k. We assume that all the VNFs have the same processing power, and each one is able to serve only one traffic request at a given time; if two traffic flows are scheduled to use a particular VNF, one of them has to wait for the other to finish its processing before being serviced by that VNF.

3. **The Virtual Network:**

   A virtual network is represented as the set of VNFs placed on the physical servers. We assume that each pair of VNFs $(\bar{f}, \bar{f}')$ is directly connected one to another through a virtual link $e = (\bar{f}, \bar{f}')$. Let $o(e)$ and $d(e)$ denote respectively the origin and destination of virtual link e. We assume that each e has a bandwidth requirement to be ensured on the physical link, and a transmission rate r_e. Similar to the VNFs, each link e may only transmit the traffic of only one flow at a time.

4. **Traffic Flows:**

   Let S be a set of network services. A network service s_i $(i \in S)$ consists of a sequence of network functions F_i which need to be processed in a proper order, and without violating a deadline d_i. f_{ij} represents the j^{th} function of network service s_i. Similar to the VNFs, for every service s_i, each f_{ij} belongs to a distinct type m. Therefore m_{ij} denotes the type of f_{ij}. As mentioned previously, the order of processing through the chain of a service must be respected. For instance, f_{ij} may not begin its processing before its previous $f_{i(j-1)}$ finishes its processing. Each s_i has a size c_i, and thus processing times p_{ij} and $\frac{c_i}{r}$ respectively, which depend on the size of f_{ij}.

4.2 The VNF Scheduling Problem

5. **The VNF Scheduling Problem:**

   The VNF scheduling problem consists of finding the optimal schedule of the traffic flows on the already placed VNFs in a way that maximizes the amount of flows admitted into the network. A flow is only admitted if and only all of its functions are placed and scheduled on VNFs, and it is successfully routed

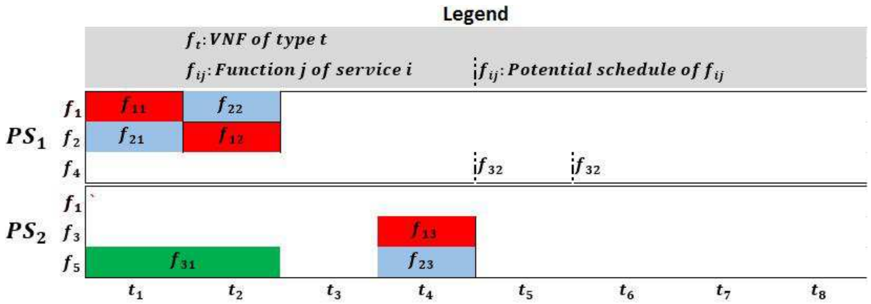


Figure 10: Schedule of Services on VNFs.

through these instances in the correct order.

The VNF Assignment problem can be formally defined as follows:

Problem Definition 1. *Given a substrate $G^s = (K, L)$ and a set of traffic flows S, each with a forwarding policy F_i, find the optimal schedule of the flows on the VNFs with the goal of maximizing the number of admitted traffic flows, while respecting the capacity constraints of the substrate network.*

An efficient schedule of the traffic flows on the VNFs with an optimized routing is highly correlated with the number of traffic flows admitted to the network, and it consists of, not only finding the best placement for each function, but also the best schedule to process it. Both the location and the processing time schedule for a particular function affect the amount of bandwidth consumed on the underlying physical links.
We highlight the importance of such a framework by the following example. Figure 9 represents a small substrate network of four PSs. Each of the PSs hosts a set of VNFs. Each substrate link has a capacity b_{uv} equivalent to the demand of two virtual links. That is each substrate link can support two virtual links at a time (We assume all virtual links have the same bandwidth requirement). We divide the time into timeslots δ of equal length. Consider three services s_1, s_2, and s_3 arriving at the same time and need to be placed and scheduled on the available servers. Each service s_i has a sequence of network functions F_i.. Each function f_{ij} has a type t_{ij}, and a size c_i. For simplicity purposes, we express the size of the functions in terms of units. Therefore, each virtual link is assumed to have a transmission rate of 1 unit per timeslot. Likewise, each VNF has a processing speed of 1 unit per timeslot. The services are characterized as follows:

- $s_1 : F_1 = <1, 2, 3>$, $c_1 = 1$.

- $s_2 : F_2 = <2, 1, 5>$, $c_2 = 1$.

- $s_3 : F_3 = <5, 4>$, $c_3 = 2$.

One possible solution would be to place all the functions on PS_1 and PS_2. The resulting schedule can be observed in Figure 10. Clearly, s_3 is suffering from a large delay. In this scenario, s_1, s_2, and s_3 need to transmit at timeslot 3. Assuming s_1, and s_2 occupied $l_{1,2}$, s_3 is no longer able to use it. As such, s_3 either has to wait one timeslot before the link is available again, and risk not meeting its deadline (if s_3 had a strict deadline, which cannot suffer from any delay), or it has to be re-routed through a longer path ($l_{1,4}$, $l_{4,3}$, and $l_{3,1}$), and occupy a lot of unnecessary bandwidth. In both cases, the network operator is suffering from a high cost: In the first case, violating the Service Level Agreement (SLA) [66], resulting in penalties for the network operator. In the second one, occupying excessive amounts of bandwidth can create bottlenecks in these links, forcing the network to reject a larger number of traffic flows due to the inability to route them.

Another valid solution would then be to place f_{32} on PS_3. With such a placement, s_3 does not have to suffer from any waiting time, and it will only occupy $l_{3,1}$ while transmitting its data. In this scenario, the network operator will have more bandwidth available in order to accept a larger number of traffic flows, and thus increase the revenue, and will have less chances of violating any SLA.

Scaling this example into real life networks, comprising large numbers of PSs, hosting hundreds or thousands of VNFs, and serving numerous traffic flows, highlights further the need of such a VNF placement and scheduling framework. As a conclusion, a placement and scheduling of functions in the network, without taking into consideration the bandwidth consumption in the underlying physical network may generate higher costs and less revenue to the network operator. Therefore, a framework capable of providing an efficient placement and scheduling of the flows, while ensuring minimal bandwidth consumption at the same time is a necessity for nowadays networks.

4.2.1 Problem Formulation

In this section, we mathematically formulate the VNF scheduling problem with the objective of maximizing the total number of flows admitted.

Parameters:

δ : Value in time.

K : Set of Physical Servers.

b_{uv} : Bandwidth capacity of link $(u,v) \in L$. u and v denote the origin and destination physical nodes of link l respectively

S : Set of services.

d_i : Deadline of service s_i.

F_i : Set of functions of service s_i.

E : Set of virtual links interconnecting pairs of VNFs.

r_e: Transmission rate of virtual.

c_i : Size of packet of service s_i.

$$w_f^k = \begin{cases} 1, & \text{if PS } k \text{ hosts a VNF of type } f, \\ 0, & \text{otherwise.} \end{cases}$$

Decision Variables:

v_{ij} : The processing ending time of function f_{ij}.

$\hat{b}^{\delta}_{u,v}$: Bandwidth consumed on physical link (u,v) at timeslot δ.

$$y_{ij}^{k\delta} = \begin{cases} 1, & \text{if function } f_{ij} \text{ began processing on server } k \text{ at } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

$$a_i = \begin{cases} 1, & \text{if service } s_i \text{ is admitted,} \\ 0, & \text{otherwise.} \end{cases}$$

$$h_{ij} = \begin{cases} 1, & \text{if } f_{i(j-1)} \text{ and } f_{ij} \text{ are on different servers,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\hat{\phi}_{ij}^{e\delta} = \begin{cases} 1, & \text{if } f_{i(j-1)} \text{ and } f_{ij} \text{ are using virtual link } e \text{ at } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

$$\phi_{ij}^{e\delta} = \begin{cases} 1, & \text{if } f_{i(j-1)} \text{ began transmission on virtual link } e \text{ at } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

$$l_{u,v}^{e\delta} = \begin{cases} 1, & \text{if virtual link } e \text{ is routed through (u,v) at } \delta, \\ 0, & \text{otherwise.} \end{cases}$$

Mathematical Model:

$$Max \quad \sum_{i \in S} a_i \tag{19}$$

subject to

VNF Scheduling

$$\sum_{k \in K : w_{t_{ij}}^{k} = 1} \sum_{\delta \in T} y_{ij}^{k\delta} = a_i \quad \forall \ i \in S, j \in F_i \tag{20}$$

$$\begin{gathered} y_{ij}^{k\delta} + y_{i'j'}^{k\delta} \leq 1 \ \ if \ t_{ij} = t_{i'j'} \\ \forall \ (i, i') \in S, j \in F_i, j' \in F_{i'}, k \in K, \delta \in T \end{gathered} \tag{21}$$

$$v_{ij} = \sum_{\delta \in T} \left(q^{\delta} + p_{ij}\right) * \sum_{k \in K} y_{ij}^{k\delta} \quad \forall i \in S, j \in F_i \tag{22}$$

$$v_{iJ} \leq d_i \quad \forall i \in S, J = |F_i| \tag{23}$$

$$\sum_{k \in K} y_{ij}^{k\delta'} < 1 - \sum_{k \in K} y_{i(j-1)}^{k(\delta + p_{i(j-1)})} \quad \forall i \in S, j \in F_i, \delta \in T, \delta' < \delta + p_{i(j-1)} \tag{24}$$

$$\sum_{k \in K} y_{ij}^{k\delta'} < 1 - \sum_{e \in E} \phi_{ij}^{e\delta} \quad \forall i \in S, j \in F_i, \delta \in T, \delta' < \delta + \frac{c_i}{r} \tag{25}$$

Constraints (20-25) handle the VNF scheduling part. Constraints (20) ensure that a service's functions cannot be scheduled unless the flow is admitted, and it can only be placed on a PS has a VNF that is able to process the function (i.e., of the same type). Constraint (21) allows only one function, during a particular timeslot δ, to be placed on a VNF. That is, no two functions of the same type may be placed on the same VNF during the same timeslot. Constraints (22) and (23) ensure that s_i is only admitted if all of its functions are processed within its deadline. Constraints (24) and (25) make sure that a function cannot begin processing if its preceding function has not finished being processed and transmitted (if needed).

Virtual Link Scheduling

$$\sum_{i \in S} \sum_{j \in F_i} \phi_{ij}^{e\delta} \leq 1 \quad \forall e \in E, \delta \in T \tag{26}$$

$$\sum_{e \in E : t_{o(e)} != t_{(j-1)}, t_{d(e)} != t_j} \phi_{ij}^{e\delta} = 0 \quad \forall \delta \in T i \in S, j \in F_i \tag{27}$$

$$\begin{gathered} \phi_{ij}^{e\delta'} \leq 1 - \sum_{k \in K} y_{i(j-1)}^{k\delta} \\ \forall \delta' < \delta + p_{i(j-1)}, i \in S, j \in F_i, \\ e \in E : o(e) = t_{j-1}, d(e) = t_j \end{gathered} \tag{28}$$

$$\hat{\phi}_{ij}^{e\delta'} = \phi_{ij}^{e\delta} \quad \forall i \in S, j \in F_i, \delta \leq \delta' \leq \delta + \frac{c_i}{r} \tag{29}$$

$$\sum_{e \in E} \sum_{\delta \in T} \phi_{ij}^{e\delta} = h_{ij} \quad \forall i \in S, j \in F_i \tag{30}$$

$$h_{ij} = \frac{1}{2}\left(\sum_{k \in K} \sum_{\delta \in T} y_{ij}^{k\delta} + \sum_{k \in K} \sum_{\delta \in T} y_{i(j-1)}^{k\delta} - 2 * \sum_{k \in K} \sum_{\delta \in T} h_{ij}^{k}\right) \quad \forall i \in S, j \in F_i \tag{31}$$

$$h_{ij}^{k} = \sum_{\delta \in T} y_{ij}^{k\delta} * y_{i(j-1)}^{k\delta} \quad \forall k \in K, i \in S, j \in F_i \tag{32}$$

Constraints (26-32) handle the virtual link scheduling part. Constraint (26) denotes that at most one function can use a particular virtual link at any timeslot. Constraint (27) forces the flow to use the designated virtual link. Constraint (28) calculates the timeslot at which a function has began transmission. Constraint (29) ensures that a function occupies the virtual link as long as not finished transmitting. Constraints (30-32) specify that a function must be scheduled for transmission only if that function and the one after it are located on different PSs. Constraint (32) is nonlinear, and therefore does not confine the ILP format, and is then replaced by the following set of constraints:

$$h_{ij}^{k} \leq \sum_{\delta \in T} y_{ij}^{k\delta} \quad \forall k \in K, i \in S, j \in F_i \tag{33}$$

$$h_{ij}^{k} \leq \sum_{\delta \in T} y_{i(j-1)}^{k\delta} \quad \forall k \in K, i \in S, j \in F_i \tag{34}$$

$$h_{ij}^{k} \geq \sum_{\delta \in T} y_{ij}^{k\delta} + \sum_{\delta \in T} y_{i(j-1)}^{k\delta} - 1 \quad \forall k \in K, i \in S, j \in F_i \tag{35}$$

Virtual Link Routing

$$\sum_{v:(u,v)\in L} l_{u,v}^{e\delta} - \sum_{v:(v,u)\in L} l_{u,v}^{e\delta} = \begin{cases} 1, & \text{if } o(e) = u, \\ -1, & \text{if } d(e) = u, \\ 0, & \text{otherwise.} \end{cases} \quad (36)$$

$$\forall u \in N, e \in E$$

$$\hat{b}_{u,v}^{\delta} = \sum_{e\in E} \sum_{i\in S} \sum_{j\in F_i} l_{u,v}^{e\delta} * \hat{\phi}_{ij}^{e\delta} \quad \forall \delta \in T, (u,v) \in L \quad (37)$$

$$\hat{b}_{u,v}^{\delta} \leq b_{u,v} \quad \forall \delta \in T, (u,v) \in L \quad (38)$$

Constraint (36) is the flow conservation constraint. Constraints (37) and (38) ensures that the number of virtual links using a physical link do not exceed the capacity of that physical link.

However, Constraint (37) is a product of two variables, making the model non linear. In order to linearize the constraint, we define a new binary variable r such that:

$$r = l_{u,v}^{e\delta} * \hat{\phi}_{ij}^{e\delta} \quad \forall \delta \in T, (u,v) \in L, e \in E, i \in S, j \in F_i \quad (39)$$

Equation (39) becomes:

$$r \leq l_{u,v}^{e\delta} \quad \forall \delta \in T, (u,v) \in L, e \in E, i \in S, j \in F_i \quad (40)$$

$$r \leq \hat{\phi}_{ij}^{e\delta} \quad \forall \delta \in T, (u,v) \in L, e \in E, i \in S, j \in F_i \quad (41)$$

$$r \geq l_{u,v}^{e\delta} + \hat{\phi}_{ij}^{e\delta} - 1 \quad \forall \delta \in T, (u,v) \in L, e \in E, i \in S, j \in F_i \tag{42}$$

$$\hat{b}_{u,v}^{\delta} = \sum_{e \in E} \sum_{i \in S} \sum_{j \in F_i} r \quad \forall \delta \in T, (u,v) \in L \tag{43}$$

4.2.2 Model Scalability

Due to the large number of variables, and the complexity of the model, the search space of the ILP model proved to be quite large, making the ILP model non-scalable. In fact, while it was able to solve a network of 2 PS and 3 flows in a matter of seconds, the model was not able to solve a network of 6 PS and 20 flows. In fact, after running for 88 minutes, the machine generated a memory space exception, despite increasing the space allocated for the model as much as possible.

Thus, alternatives must be found in order to overcome the non-scalability of the model. Several methods exist, that serve to surpass the non-scalability. Most important of which are heuristics, column generation, and the cut-and-solve approach. However, this step is out of the scope of this paper.

4.3 Conclusion

In the literature, the problem of placement and scheduling of traffic flows on VNF concentrated on finding solutions that minimize the makespan. However, the physical, limited, resources of the network were not taken into consideration. The underlying physical nodes, and links have fixed capacities, that may alter the way a schedule is organized. Moreover, the existing work does not consider any transmission delay, which is not realistic. Data packets, when transmitted over virtual links suffer from transmission delays. It is realistic, that over one virtual link, a 1 mbps packet will take 10 times less time to be transmitted than a 10 mbps packet.

In this chapter, we solve the problem of placement and scheduling of traffic flows on VNF while taking into consideration transmission delay over virtual links. Moreover, we adress, jointly, the problem of steering the traffic over the underlying physical network. In order to solve this problem, we formulate an ILP model. However, due to the non-scalability of the model, we propose a Cut-and-Solve approach, which is out of the scope of this thesis.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Network Function Virtualization promises to alleviate numerous challenges that their hardware counterparts suffer from. Indeed NFV, through virtualizing the hardware MBs, reduces the CAPEX and OPEX, and provides network operators with the necessary flexibility to efficiently accommodate the varying shape of the traffic in the network. However, being in its infancy, NFV comes with a number of challenges, that need to be addressed and solved. One of these challenges is the problem of placement and scheduling of the traffic flows on VNFS, a problem carefully studied in this thesis.

In chapter 3, we solve the problem while taking into account the power consumption of the physical server. We formulate an ILP model that tries to place and schedule the traffic on VNFs, while trying to find the schedule that optimizes the power consumed in the network. To overcome the non-scalability of the model, we resort to a heuristic, that serves as a tradeoff between minimizing the makespan and consolidation. To test its efficiency, we create two benchmarks, that serve as the two extremes (Consolidation amd minimizing the makespan) and compare it with them. The results show that our algorithm outperforms the two benchmarks, by admitting more flows and reducing the energy consumption of the network.

In chapter 4, we also solve the problem of placement and scheduling of traffic flows on VNFs. However, we extend the existing work, and take into consideration the transmission delays between pairs of VNFs.

Moreover, we address the problem of routing these virtual links on the underlying physical network with finite capacity. We formulate the problem as an ILP model, and test its scalability.

5.2 Future Work

Network Function Virtualization is receiving a considerable amount of attention by the researchers lately. However, it is still in its early phases, and there is room for a lot of improvement, specifically, the problem of placement and scheduling of traffic flows on VNFs. In this regard, what follows is a list of potential problems to be considered in the future:

- In chapter 3, we formulated an ILP model, and a heuristic that places and schedules traffic flows on VNFs while minimizing the energy consumption. In this chapter, we considered homogeneous servers. Indeed, we assumed that all of the servers have the same power consumption, which is not quite realistic. In fact, a network may have hardware equipment from different generations, and vendors. Therefore, it would be more realistic to consider a heterogeneous network, and develop a solution for it.

- In chapter 3, we only considered the offline version of the problem. Considering the online version, where flows arrive and leave over time is an important problem to consider.

- In chapter 4, we formulated an ILP model to solve the problem of placement and scheduling of traffic flows on VNFs, while taking into consideration the transmission delays and the routing of the virtual links on the physical network. The ILP model is non-scalable. As future work, to overcome this non-scalability, would be to decompose the ILP model, and solve the problem using a Cut-and-Solve approach.

- Reliability is an important factor for any client. Therefore, a framework that is able to place and schedule traffic flows based on their reliability requirements should be studied.

Bibliography

[1] William Voorsluys, James Broberg, and Rajkumar Buyya. Introduction to cloud computing. *Cloud computing: Principles and paradigms*, pages 1–44, 2011.

[2] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[3] Amazon elastic computing cloud, aws.amazon.com/ec2.

[4] Cloud hosting, cloud computing and hybrid infrastructure from gogrid, http://www.gogrid.com.

[5] Google app engine, url http://code.google.com/appengine.

[6] Windows azure, www.microsoft.com/azure.

[7] Dedicated server, managed hosting, web hosting by rackspace hosting, http://www.rackspace.com.

[8] Broadsoft, http://www.broadsoft.com.

[9] Brian Carpenter and Scott Brim. Middleboxes: Taxonomy and issues. Technical report, RFC 3234, February, 2002.

[10] Pyda Srisuresh, Jiri Kuthan, Abdallah Rayhan, Jonathan Rosenberg, and Andrew Molitor. Middlebox communication architecture and framework. 2002.

[11] P Quinn and T Nadeau. Service function chaining problem statement. *draft-ietf-sfc-problem-statement-07 (work in progress)*, 2014.

[12] S Kumar, M Tufail, S Majee, C Captari, and S Homma. Service function chaining use cases in data centers. *IETF SFC WG*, 2015.

[13] W Haeffner, J Napper, M Stiemerling, D Lopez, and J Uttaro. Service function chaining use cases in mobile networks. *draft-ietf-sfc-use-case-mobility-01*, 2014.

[14] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 459–473. USENIX Association, 2014.

[15] Network Operators. Network functions virtualization, an introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow SDN and OpenFlow World Congress*, 2012.

[16] NM Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *Communications Magazine, IEEE*, 47(7):20–26, 2009.

[17] Jinho Hwang, KK Ramakrishnan, and Timothy Wood. Netvm: high performance and flexible networking using virtualization on commodity platforms. *Network and Service Management, IEEE Transactions on*, 12(1):34–47, 2015.

[18] Ye Yu, Qian Chen, and Xin Li. Distributed collaborative monitoring in software defined networks. *arXiv preprint arXiv:1403.8008*, 2014.

[19] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization: State-of-the-art and research challenges. 2015.

[20] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, and Xiang Zhang. Network function virtualization in the multi-tenant cloud. *Network, IEEE*, 29(3):42–47, 2015.

[21] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 323–336, 2012.

[22] Elisa Maini and Antonio Manzalini. Management and orchestration of virtualized network functions. In *Monitoring and Securing Virtualized Networks and Services*, pages 52–56. Springer, 2014.

[23] K Giotis, Y Kryftis, and V Maglaris. Policy-based orchestration of nfv services in software-defined networks. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5. IEEE, 2015.

[24] Stuart Clayman, Elisa Maini, Alex Galis, Antonio Manzalini, and Nicola Mazzocca. The dynamic placement of virtual network functions. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9. IEEE, 2014.

[25] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. Opennf: Enabling innovation in network function control. *ACM SIGCOMM Computer Communication Review*, 44(4):163–174, 2015.

[26] Aaron Gember, Robert Grandl, Ashok Anand, Theophilus Benson, and Aditya Akella. Stratos: Virtual middleboxes as first-class entities. *UW-Madison TR1771*, page 15, 2012.

[27] Jordi Ferrer Riera, Eduard Escalona, Josep Batalle, Eduard Grasa, and Joan Antoni Garcia-Espin. Virtual network function scheduling: Concept and challenges. In *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*, pages 1–5. IEEE, 2014.

[28] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Steven Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–9. IEEE, 2015.

[29] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[30] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[31] http://www.etsi.org/.

[32] ETSI Industry Specification Group (ISG) NFV. Etsi gs nfv-man001 v1.1.1: Network functions virtualisation (nfv) managementand orchestration. 2014.

[33] Open platform for nfv (opnfv). 2015.

[34] Mcn: The mobile cloud networking eu project. 2015.

[35] T-nova: Network functions-as-a-service (nfaas) over virtualized infrastructures. 2015.

[36] Tech. Rep. TeleManagement Forum. Zero-time orchestration, operations and management (zoom). 2014.

[37] D. R. Lopez. Openmano: The dataplane ready open source nfv mano stack. 2015.

[38] Intel. Openstack enhanced platform awareness. white paper. 2015.

[39] András Császár, Wolfgang John, Mario Kind, Catalin Meirosu, Gergely Pongrácz, Dimitri Staessens, Alexandru Takacs, and Fritz-Joachim Westphal. Unifying cloud and carrier network: Eu fp7 project unify. In *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, pages 452–457. IEEE, 2013.

[40] Convergence of wireless optical network and it resources in support of cloud services (content) eu project. 2015.

[41] Hp opennfv reference architecture. 2015.

[42] Huawei nfv open lab. 2015.

[43] Intel open network platform. 2015.

[44] Cloudnfv. 2015.

[45] Alcatel-lucentâĂŹs clouband. 2015.

[46] Broadcom open nfv. 2015.

[47] Cisco. Nfv management and orchestration: Enabling rapid service innovation in the era of virtualization.

[48] Nfv: Beyond virtualization. techinical white paper. 2014.

[49] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. Toward software-defined middlebox networking. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 7–12. ACM, 2012.

[50] Raffaele Bolla, Chiara Lombardo, Roberto Bruschi, and Sergio Mangialardi. Dropv2: energy efficiency through network function virtualization. *Network, IEEE*, 28(2):26–32, 2014.

[51] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 543–546, 2014.

[52] Tamás Lukovszki and Stefan Schmid. Online admission control and embedding of service chains. In *Structural Information and Communication Complexity*, pages 104–118. Springer, 2014.

[53] Md Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, Raouf Boutaba, et al. Orchestrating virtualized network functions. *arXiv preprint arXiv:1503.06377*, 2015.

[54] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[55] Ali Mohammadkhan, Sheida Ghapani, Guyue Liu, Wei Zhang, KK Ramakrishnan, and Timothy Wood. Virtual function placement and traffic steering in flexible and dynamic software defined networks. In *Local and Metropolitan Area Networks (LANMAN), 2015 IEEE International Workshop on*, pages 1–6. IEEE, 2015.

[56] Abhishek Gupta, M Farhan Habib, Pulak Chowdhury, Massimo Tornatore, and Biswanath Mukherjee. Joint virtual network function placement and routing of traffic in operator networks. 2015.

[57] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 98–106. IEEE, 2015.

[58] Windhya Rankothge, Jiefei Ma, Franck Le, Alessandra Russo, and Jorge Lobo. Towards making network function virtualization a cloud computing service. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 89–97. IEEE, 2015.

[59] Hendrik Moens and Filip De Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 418–423. IEEE, 2014.

[60] Li Erran Li, Vahid Liaghat, Hongze Zhao, MohammadTaghi Hajiaghay, Dan Li, Gordon Wilfong, Y Richard Yang, and Chuanxiong Guo. Pace: policy-aware application cloud embedding. In *INFOCOM, 2013 Proceedings IEEE*, pages 638–646. IEEE, 2013.

[61] Internet live stats, http://www.internetlivestats.com/internet-users/.

[62] Yan Chen, Shunqing Zhang, Shugong Xu, and Geoffrey Ye Li. Fundamental trade-offs on green wireless networks. *Communications Magazine, IEEE*, 49(6):30–37, 2011.

[63] Raffaele Bolla, Roberto Bruschi, Alessandro Carrega, Franco Davoli, Diego Suino, Constantinos Vassilakis, and Anastasios Zafeiropoulos. Cutting the energy bills of internet service providers and telecoms through power management: An impact analysis. *Computer Networks*, 56(10):2320–2342, 2012.

[64] Sen Su, Zhongbao Zhang, Alex X Liu, Xiang Cheng, Yiwen Wang, and Xinchao Zhao. Energy-aware virtual network embedding. *Networking, IEEE/ACM Transactions on*, 22(5):1607–1620, 2014.

[65] https://www.energystar.gov.

[66] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. 2009.