Towards the Softwarization of Content Delivery Networks for Component and Service Provisioning

Narjes Tahghigh Jahromi

A

Thesis

In

The Concordia Institute

For

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Information and Systems Engineering) at

Concordia University

Montreal, Quebec, Canada

October 2018

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Narjes Tahghigh Jahromi

Entitled: Towards Softwarization of CDNs for Component and Service Provisioning

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

Doctor Of Philosophy (Information and Systems Engineering)

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

Signed by the final examining committee:

Dr. Hua Ge	Chair
Dr. Abdelhakim Senhaji Hafid	External Examiner
Dr. Juergen Rilling	External to Program
Dr. Chadi Assi	Examiner
Dr. Jamal Bentahar	Examiner
Dr. Roch Glitho	Thesis Supervisor (s)

Approved by

Dr. Chadi Assi — Chair of Department or Graduate Program Director

Nov. 28, 2018
Date of Defence

Dr. Amir Asif — Dean, Gina Cody School of Engineering and Computer Science

ABSTRACT

Towards the Softwarization of Content Delivery Networks for Component and Service Provisioning

Narjes Tahghigh Jahromi, Ph.D.
Concordia University, 2018

Content Delivery Networks (CDNs) are common systems nowadays to deliver content (e.g. Web pages, videos) to geographically distributed end-users over the Internet. Leveraging geographically distributed replica servers, CDNs can easily help to meet the required Quality of Service (QoS) in terms of content quality and delivery time. Recently, the dominating surge in demand for rich and premium content has encouraged CDN providers to provision value-added services (VAS) in addition to the basic services. While video streaming is an example of basic CDN services, VASs cover more advanced services such as media management.

Network softwarization relies on programmability properties to facilitate the deployment and management of network functionalities. It brings about several benefits such as scalability, adaptability, and flexibility in the provisioning of network components and services. Technologies, such as Network Functions Virtualization (NFV) and Software Defined Networking (SDN) are its key enablers.

There are several challenges related to the component and service provisioning in CDNs.

On the architectural front, a first challenge is the extension of the CDN coverage by on-the-fly deployment of components in new locations and another challenge is the upgrade of CDN components in a timely manner, because traditionally, they are deployed statically as physical building blocks. Yet, another architectural challenge is the dynamic composition of required middle-boxes for CDN VAS provisioning, because existing SDN frameworks lack features to support the dynamic chaining of the application-level middle-boxes that are essential building blocks of CDN VASs. On the algorithmic front, a challenge is the optimal placement of CDN VAS middle-boxes in a dynamic manner as CDN VASs have an unknown end-point prior to placement.

This thesis relies on network softwarization to address key architectural and algorithmic challenges related to component and service provisioning in CDNs. To tackle the first challenge, we propose an architecture based on NFV and microservices for an on-the-fly CDN component provisioning including deployment and upgrading. In order to address the second challenge, we propose an architecture for on-the-fly provisioning of VASs in CDNs using NFV and SDN technologies. The proposed architecture reduces the content delivery time by introducing features for in-network caching. For the algorithmic challenge, we study and model the problem of dynamic placement and chaining of middle-boxes (implemented as Virtual Network Function (VNF)) for CDN VASs as an Integer Linear Programming (ILP) problem with the objective of minimizing the cost while respecting the QoS. To increase the problem tractability, we propose and validate some heuristics.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Roch H. Glitho for the continuous support of my Ph.D. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I would like to gratefully thank my Ph.D. committee members: Prof. Juergen Rilling, Prof. Chadi Assi, and Prof. Jamal Bentahar, and my external examiner, Prof. Abdelhakim Senhaji Hafid for their insightful comments and time devoted to my thesis.

I am also thankful to Dr. Sami Yangui, Dr. Diala Naboulsi and other former colleagues for their assistance and collaborations. I would like to thank Ericsson MDN team, for their discussions, enlightening comments, and valuable feedback during my internship.

Furthermore, I am thankful to all of my colleagues in the Telecommunication Service Engineering lab at Concordia University. In particular, I would like to thank Mohammad Abu Lebdeh and Abbas Soltanian for their companionship, ideas, and fruitful discussions.

My special appreciation words go to my best colleague, Carla Mouradian, who became my best friend soon after I met her. I love you more than words can express not only for your warm heart, sharp mind and unbelievable support to help me overcome my life challenges but also for being my best companion in my joyful Ph.D. moments. You have a forever friend and forever has no end.

To all my friends, I express my gratitude who have always been a major. In particular, I would like to thank Ilnaz Mobayen, for her unconditional friendship, kindness, and patience. Thank you for always being there during my toughest moments of Ph.D. years.

Last but not least, I am forever thankful to my family who provided me through moral and emotional support. My sincere thanks go to my mom and dad. Without your sacrifices, love, and prayers this thesis would not have been possible. I thank my brothers for their support, encouragements, and advice. There are no words that can express my gratitude and love for you.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ABR	Adaptive Bit Rate
API	Application Programing Interface
CDN	Content Delivery Network
COTS	Commercial Off-The-Shelf
DPI	Deep Packet Inspection
ETSI	European Telecommunications Standards Institute
GUI	Graphical User Interface
HTTP	Hypertext Markup Language
IaaS	Infrastructure as a Service
IoT	Internet of Things
ISP	Internet Service Provider
MANO	Management and Orchestration
NAT	Network Address Translation
NF	Network Function
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NIST	National Institute of Standards and Technology
NS	Network Service
NSO	Network Service Orchestrator
O&M	Orchestration and Management
OSS/BSS	Operational Support System/Business Support System
PaaS	Platform as a Service
PoD	Point of Deployment
QoS	Quality of Service
REST	REpresentational State Transfer
RO	Resource Orchestration
SaaS	Software as a Service
SDN	Software Defined Networking

SFC	Service Function Chaining
SLA	Service Level Agreement
VM	Virtual Machine
VAS	Value-added Service
VNF	Virtualized Network Function
WAN	Wide Area Network

Chapter 1

1. Introduction

In this chapter we first provide an overview of the thesis, then we review the challenges that exist toward softwarization of CDNs followed by the thesis contributions. Afterward, we present the background information related to the thesis. Finally, the thesis outline is given.

1.1. Overview

Content Delivery Networks (CDNs) have emerged to improve the network performance and content distribution by offering mechanisms and infrastructure to deliver the content to a large number of end-users. CDNs are defined as a collaborative collection of replica servers (a.k.a. surrogate or edge servers) and a CDN controller [1] [2][3]. The replica servers are located close to end-users and offload the content providers' origin servers by hosting the replicated content and perform a transparent and efficient delivery of content to end-users. The CDN controller manages the CDN and redirects the end-users' requests to the appropriate replica server [1], according to their content availability, network condition, and end-user location.

According to a Cisco Virtual Networking Index forecast [4], [5], the video traffic will take up to 80% of Internet traffic by 2020 and around two-thirds of this traffic will be carried by CDNs. The significant importance of CDNs and also the end-users' demand for rich and premium content has motivated CDN providers to provision Value-Added Services (VASs), such as media management/video ad-insertion, in addition to their basic services. While CDN basic services include audio/video streaming and audio/video/software download, CDN VASs cover more advanced services. According to an article published in Streaming Media magazine (November 2012 issue) [6], VASs cover the CDN injected services such as "*application acceleration, dynamic site acceleration, front-end optimization, mobile content acceleration, media management (transcoding, ad-insertion, content protection), and a host of other services for the purposes of security and commerce*". Considering the case of media management, an example is advertisement insertions (ad-insertion). Another example of such a service is sign-language video insertion for hearing-impaired end-users.

Network softwarization is a concept that has emerged in recent years and has the potential to significantly help to facilitate the provisioning of CDN components and services. Network softwarization relies on programmability properties of the network functionalities (i.e. middle-boxes) and network resources (e.g. routers and switches) to facilitate the deployment and management of network functionalities. It brings about scalability, elasticity, adaptability, and flexibility in introducing new services and provisioning components. Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) are two complementary technologies that can realize network softwarization in CDNs. NFV and SDN can help re-architecting the traditional CDN components, with virtualization as their key feature. NFV is a novel way to virtualize network services and brings flexibility and efficiency by decoupling the network functions (NF), from the underlying hardware. It defines NFs as standalone pieces of software called Virtual Network Function (VNF). As a complementary technology to NFV, SDN enables the dynamic programming of network resources by decoupling the network control from the data plane. This leads to the dynamic orchestration and chaining of the already-deployed VNFs.

1.2. Challenges and Thesis Contributions

Network softwarization in CDNs can help to tackle several challenges in provisioning components and services in CDNs. Examples of such challenges are given below.

- **Flash Crowds:** In some events, some content attracts the attention of a large number of end-users and this might lead to flash crowds, which are the sudden and unpredicted surges in requests towards particular contents. Flash crowds eventually cause an outage in CDNs, e.g. a decrease in Quality of Service (QoS) and site slow-downs. Terrorist attacks of September 11, 2001 [7], is among examples that lead to flash crowds. In the case of flash crowds, traditional CDNs are challenged to promptly extend their coverages by provisioning additional CDN components in new locations. This is mainly because traditionally, they are provisioned statically on a proprietary hardware [8] and their deployment and management tasks require labor-intensive manual efforts.
- **Upgrading the CDN components:** In the CDN context, with the growth of video traffic, the viewer requirements are also evolving. This forces the CDN providers to frequently adapt their systems to the market needs. Traditionally,

CDN components are designed as physical building blocks. Hardware-based deployments require lengthy upgrade intervals when new features are needed, due to the fact that this process is done manually. Although recently some CDN providers have been offering virtualized components, they are designed as monolithic software, which imposes unavoidable complexity for the upgrade process. In such cases, the upgrade process requires to take the whole component off service temporarily and recover it after the upgrade process is completed. In order to deal with the challenges of CDN component upgrades with minimum service downtime, the CDN architectures need to be rethought.

- **On-the-fly VAS provisioning:** Another challenge is on-the-fly provisioning of VASs in CDNs, including the dynamic chaining of the required VAS middle-boxes. Based on the Internet Engineering Task Force (IETF) categorization [9], the building blocks of VASs are application-level middle-boxes such as video transcoder, mixer, and compressor. Traditionally, these middle-boxes are provisioned statically [10], at fixed network locations, as most of the middle-boxes nowadays. The reliance of network services to their underlying network topology brings some challenges in their deployment and chaining. Even though some technologies such as SDN can help to facilitate CDN VAS provisioning, it’s still challenging as the chaining of the application-level middle-boxes requires specific features that are overlooked in existing SDN frameworks.
- **The location of VAS middle-boxes:** In order to realize a VAS, the required middle-boxes need to be deployed and chained. However, the location where middle-boxes are deployed has a great impact on CDN provider expenses and end-user perceived QoS. Moreover, CDN VASs have a particular feature that is the fact that a chain end-point is unknown prior to the placement. This end-point corresponds to the replica server that is selected to serve the content to the end-

user. This makes the placement problem challenging as the assignment of a replica server to end-users impacts the optimal placement of middle-boxes in the chain.

These challenges must be addressed before the agile and flexible provisioning of CDN components and VASs can advance to reality. This thesis aims at addressing the fundamental architectural and algorithmic challenges related to the network softwarization in CDNs. These problems along with potential solutions are summarized next.

1.2.1. An Architecture for on-the-fly component provisioning in CDNs [11], [12]

As mentioned earlier, CDN components are traditionally provisioned as physical building blocks on proprietary hardware. The fact that those functionalities are hardly coupled to the underlying hardware brings about some unique challenges in the provisioning of basic services in CDNs. In particular, when a specific content gains popularity, flash crowds might occur that might lead to site slowdowns or service unavailability. In such cases, CDNs are challenged to react fast in extending their coverage by provisioning new components in locations where there is a high demand for the content. Furthermore, CDN components require to be upgraded frequently as the ever-changing end-user needs call for frequent upgrades to support new content formats, protocols, and content protection requirements. Currently, CDN components are provisioned as monolithic functionalities that makes the upgrade process complex and time-consuming. In this context, automation remains essential, especially to cope with sudden changes in service usage pattern and CDN provider requirements.

In this thesis, we propose an architecture to tackle the related challenges for on-the-fly provisioning of new CDN components when needed. the architecture leverages the microservice architectural style [13] and NFV technology [14] to enable the on-the-fly provisioning (including on-the-fly deployment and upgrading) of CDN components. The proposed modular design of CDN components using microservices architectural style facilitate the upgrade process with a minimum downtime. The ETSI MANO principles [15] brings automation in deployment and management procedure. In addition, a pre-deployment and a post-deployment procedure are considered to be covered by the architecture which are crucial procedures for automation of the whole process.

1.2.2. An Architecture for on-the-fly Provisioning of VAS CDN Services [16], [17]

The building blocks of CDN VASs are application-level middle-boxes such as mixer, transcoder, and compressor. To realize an end-to-end VAS, the above-mentioned middle-boxes need to be deployed and chained in the network. Traditionally, such middle-boxes are provisioned statically on proprietary hardware. Their substantial reliance on the underlying hardware and network topology makes VAS provisioning very challenging. In unpredicted events when many videos go viral, CDNs are challenged to react fast to provision VASs in various locations, in a timely manner, and with sufficient flexibility. SDN technology can facilitate service provisioning by on-the-fly chaining of required middle-boxes. However, the existing SDN frameworks currently support chaining of IP-level middle-boxes such as NAT and firewall. They need to be redesigned to enable on-the-fly chaining of application-level middle-boxes such as video mixer and compressor that are essential in CDN context.

In this thesis, an architecture is designed for on-the-fly provisioning of VASs for CDNs. The architecture relies on well-adopted technologies, such as NFV and SDN, to enable CDN VAS provisioning by dynamic chaining of application-level middle-boxes. The application-level middle-boxes, that are key components of CDN VASs, are packaged in-line with NFV technology in form of VNFs. The architecture includes extensions to the existing SDN switches and controllers, to support the dynamic chaining of application-level VNFs for VASs. Besides, the architecture provides in-network caching features to reduce the high latency overhead perceived by end-users when delivering content for VASs. It should be noted that also we have used our proposed method for chaining application-level VNFs in another domain i.e. Internet of Things (IoT) in [18].

1.2.3. An Algorithm for Dynamic Placement of VNFs for VAS [19], [20]

As mentioned earlier, to realize a VAS, a set of middle-boxes should be deployed and chained in the network. However, the location where middle-boxes are placed significantly impacts the total cost of CDN providers and the perceived QoS by end-users. Placement of middle-boxes for VASs is challenging as CDN VASs have an unknown chain endpoint prior to their placement. In NFV settings, such middle-boxes are defined and packages in form of VNFs that can be migrated over NFV Infrastructure (NFVI) from one server to another, as they are softwarized functionalities. Although VNF placement and chaining problems have received significant attention in the research community in recent years [21], to the best of our knowledge, very few research efforts (i.e. [22][23][24]) focus on the VNF-FGE for CDN VASs. Furthermore, they only focus on the static mode of placement in which VAS VNFs are placed and chained all together once. However, in real life, end-user requests for VAS vary in time and space and are rather difficult to forecast. Thus the necessity of dynamicity in the placement should be considered.

In this thesis, we propose an approach for dynamic VNF placement for VASs in CDNs (DVPVC). In the proposed approach, already deployed VNFs might be re-used when new VASs are introduced. Furthermore, the topology of the deployed VNFs is re-configured whenever necessary. The problem is modeled as an Integer Linear

Programming (ILP) which minimizes the reconfiguration cost while meeting the required QoS. Reconfiguration cost is defined as an aggregation of VNF instantiation, migration, hosting and routing costs. To increase the problem tractability in large-scale scenarios, we propose a Tabu-search algorithm to find a suboptimal solution. We evaluate our proposed algorithm against the greedy first fit algorithm and the optimal solution. The results show that the proposed algorithm reaches the optimality in several scenarios and significantly helps in reducing reconfiguration costs compared to the first fit algorithm.

Next, we provide some background information related to the topics in this thesis.

1.3. Background Information

This subsection presents the background information that is relevant to our research domain. The background information covers three topics: Content Delivery Networks, Network Function Virtualization, and Software Defined Networks.

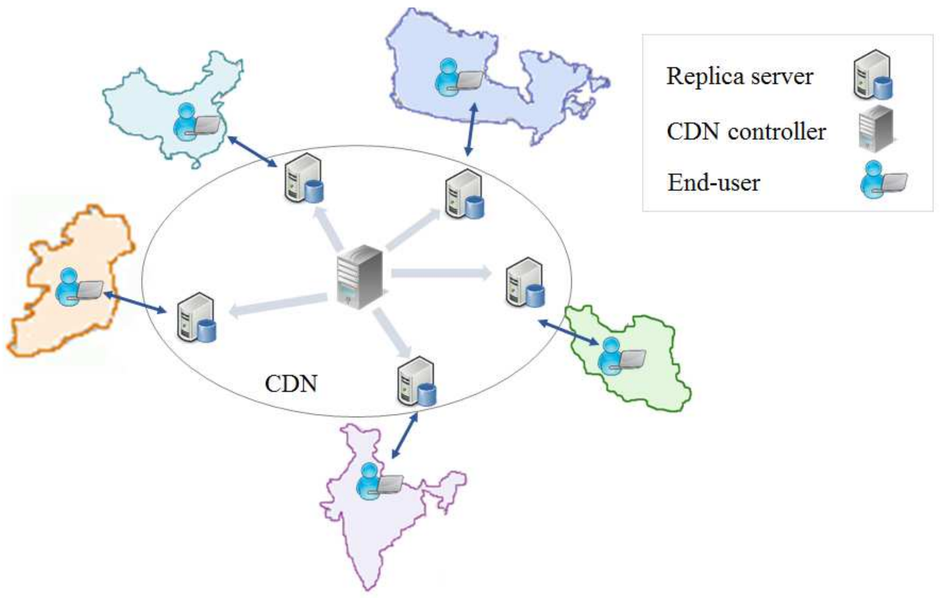


Figure 1.1. A high-level view of the CDN Architecture

1.3.1. Content Delivery Networks (CDN)

A Content Delivery Network or Content Distribution Network (CDN) is a distributed network of collaborative content delivery elements which enable the efficient delivery of content services to a large number of geographically distributed end-users. Based on the several definitions provided by the relevant literature (e.g. [1][2][3]), and as shown in Fig. 1.1, CDNs can be described as an overlay network that consists of replica servers (a.k.a. surrogate or edge servers) and a controller. CDN replica servers are geographically distributed in strategic network locations, close to end-users. They hold a copy of the original content which is initially located on the origin servers, owned and managed by content providers. The CDN controller is a key entity in CDNs, responsible for redirecting the end-user requests to the appropriate replica server and managing the whole CDN network. The replica server selection is based on a set of predefined criteria, such as content availability, physical distance, network conditions, replica server overhead, and etc. Content providers rely on CDNs to easily improve their service performance and meet the required QoS in terms of content availability and delivery time. CDNs can also help to localize the traffic to reduce the overhead in the core network and eventually to decrease the overall network bandwidth usage.

1.3.2. Network Function Virtualization (NFV)

NFV is an emerging technology which offers a novel way to virtualize network services[14][25][26]. It aims at decoupling the network functions from the underlying hardware [27] by defining the network functions and middle-boxes as stand-alone pieces of software called Virtual Network Functions (VNF). This eventually enables the dynamic provisioning of network functions and middle-boxes (e.g. NAT, firewall) on top of any generic, Commercial Off-The-Shelf (COTS) hardware, anytime and anywhere in the network. This makes it easy for service providers and enterprises to

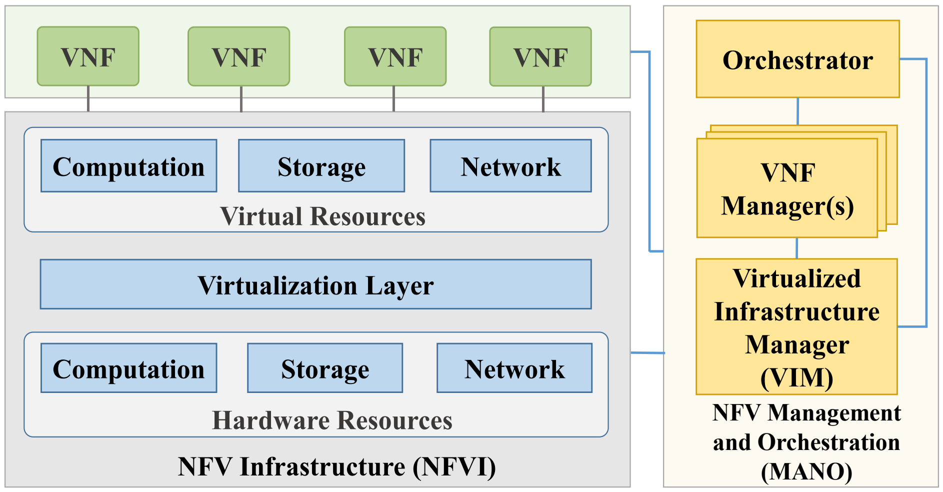


Figure 1.2. A high-level view of the NFV architecture.

deploy new services faster while maximizing their investments in the existing platforms. The European Telecommunications Standards Institute (ETSI) has defined a reference architectural framework for NFV [15]. As Fig. 1.2 shows, the NFV framework includes three main components: VNFs, NFV Infrastructure (NFVI), and NFV Management and Orchestration (MANO). A VNF is the software implementation of a given middle-box (i.e. network function). NFVI provides the virtualized and physical resources as services. NFVI is managed by MANO, providing the required environments for VNFs to be deployed and executed. NFV MANO is also responsible for VNF and network service life cycle provisioning including deployment, execution, and management.

One of the main NFV goals is to bring agility, dynamicity and cost reductions in network service deployments. In NFV context, in order to realize an end-to-end network service (NS), a set of VNFs should be composed, i.e. deployed in the network and chained. The chain, a.k.a. VNF Forwarding Graph (VNF-FG), is an ordered set of VNFs that the traffic steers among them. NFV exploits the virtualization technique to dynamically place the VNFs on the NFVI. Reference [28] defines three main stages for NFV Resource Allocation (NFV-RA): i) VNFs Chain Composition (VNFs-CC), i.e.

Service Function Chaining (SFC), ii) VNF Forwarding Graph Embedding (VNF-FGE), and iii) VNFs Scheduling (VNFs-SCH). The VNFs-CC stage defines the way that VNFs should be composed in a VNF-FG (their order in the chain) such that the service provider requirements are met. The VNF-FGE stage decides about the appropriate location to place the VNFs and chain them in a VNF-FG, in a way that resource costs are minimized and QoS is met. The VNFs-SCH focuses on the VNF scheduling on high-volume servers in a way that the total execution time of the network service is minimized and an improved performance is obtained.

1.3.3. Software Defined Networking (SDN)

SDN is a networking technology which aims at splitting the control plane and the data plane of the network elements. The goal is to provide a robust management of the forwarding behavior of the network elements [29][30]. The control plane can easily program the forwarding elements of the network, according to the emerging application requirements and policies [31].

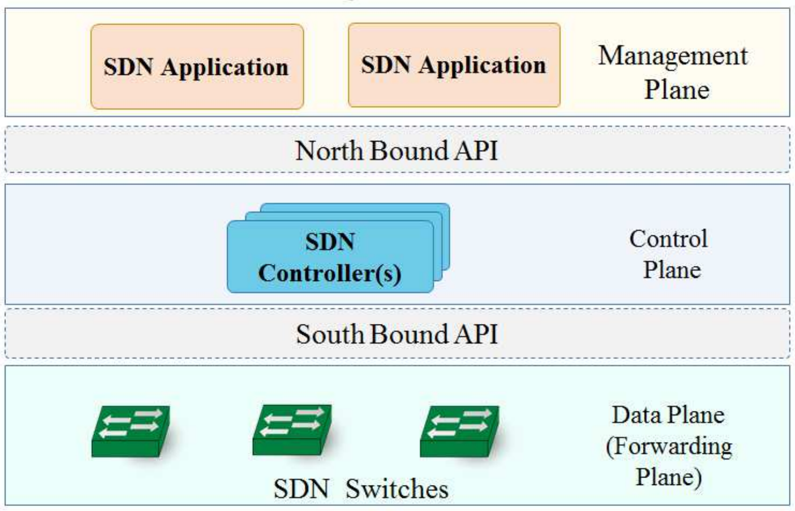


Figure 1.3. A high-level view of the SDN architecture.

As illustrated in Fig. 1.3, the SDN architecture is composed of three main planes: (i) management plane, (ii) control plane and (iii) forwarding plane [32][33]. The management plane includes the SDN applications that define the application policies and inject them to the control plane via appropriate Application Programming Interfaces (APIs). The control plane includes SDN controllers that program the forwarding plane by populating the SDN switches with forwarding rules. The forwarding plane consists of SDN elements i.e. switches and routers and steers the traffic based on the forwarding rules that reflect the application policies. As a complementary technology to NFV, SDN enables the dynamic orchestration and chaining of VNFs [34].

1.4. Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 introduces the motivating scenario and requirements, followed by the critical review of the state-of-the-art. Chapter 3 presents an architecture for on-the-fly provisioning and upgrade of CDN components. Chapter 4 is devoted to the proposed architecture for provisioning of CDN VASs. Chapter 5 studies the VNF placement and chaining problem for CDN VASs and presents the proposed ILP model and heuristics for that. We conclude this manuscript in chapter 6 and provide future directions for this research work.

Chapter 2

2. Related Work

In this section, we first present some motivating scenarios from which we derive some architectural and algorithmic requirements. Next, we review the related work according to the derived requirements.

2.1. Motivating Scenarios

In this subsection, we provide a scenario that motivates the need for on-the-fly deployment of CDN components followed by a motivating scenario for their on-the-fly upgrade. Next, we present a motivating scenario for on-the-fly provisioning of CDN VASs and finally, we present motivations behind dynamic placement of VAS middleboxes in CDNs.

2.1.1. On-the-fly CDN component provisioning

Let us assume a business model with a content provider, a CDN provider, and an Internet Service Provider (ISP). The content provider, e.g. Canadian Broadcast Corporation (CBC) news, provides contents to end-users. CDN provider operates and manages a CDN and provides content delivery services to content providers. ISP provides Internet connectivity services to end-users and additionally, we assume that it provides Points of Deployment (PoDs) to CDN providers. PoDs provide the infrastructure and resources needed for hosting the CDN replica servers.

In a first scenario, let us assume that initially, the contents of the content provider are replicated on a replica server in Toronto and as shown in Fig. 2.1.a., all end users in Canada are redirected to this replica server. Let us now assume that an unpredictable event (e.g. British Columbia floods in 2017) happens and hence some contents gain popularity in a short period of time. The CDN provider might notice a surge in traffic from British Columbia, which might lead to a flash crowd as shown in Fig. 2.1.b.

In this situation, the CDN provider might decide to deploy (in a fully automated manner) a new CDN component, e.g. a replica server, at ISP premises in British Columbia. As part of the pre-deployment phase, it will request the deployment at specific ISP premises and will retrieve from the ISP the information related to the specific PoD the server will be deployed on. The deployment per se will take place. During the post-deployment phase, the CDN provider will fully integrate the newly deployed replica server in its content delivery network and will fill it with the popular contents. As illustrated in Fig. 2.1.c, the end-users in British Columbia will then be automatically

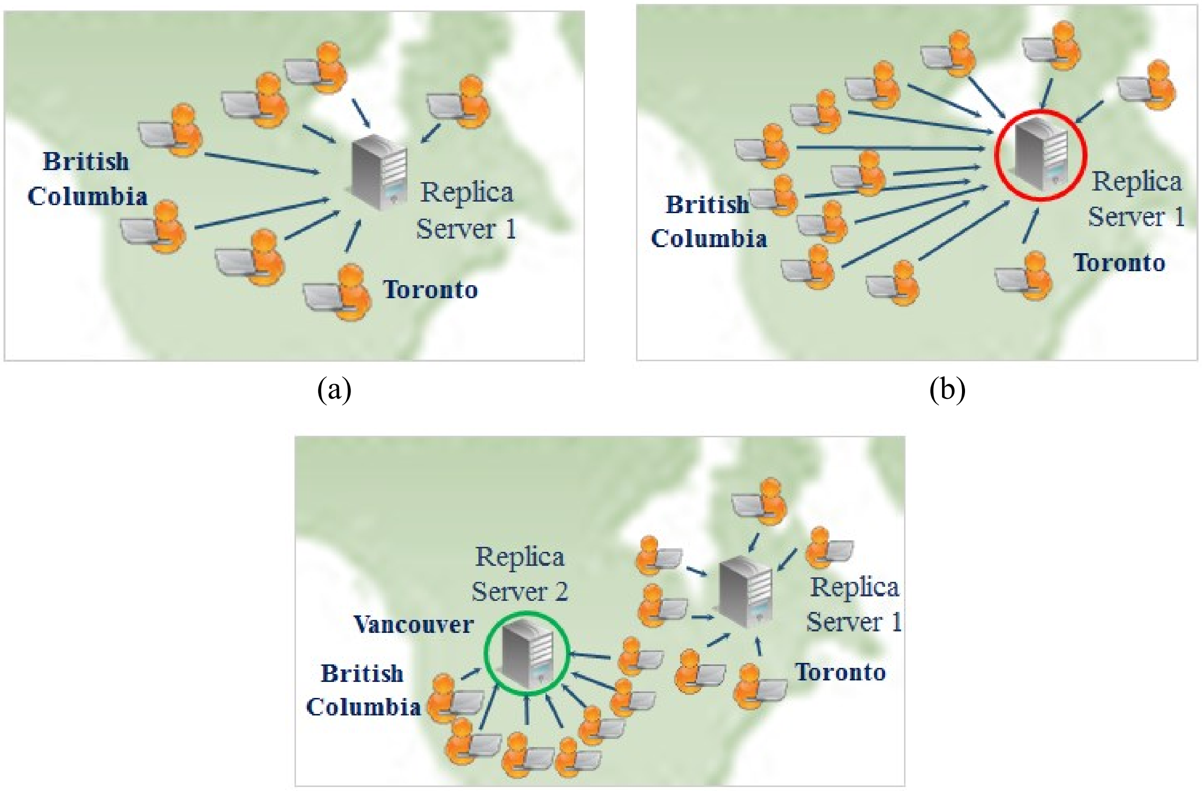


Figure 2.1. An illustration of flash crowds use case

a) Normal situation; end-users are redirected to replica server 1 in Toronto, b) Flash crowds; a content gains sudden popularity, c) Automatic provisioning of a new replica server 2 in Vancouver and the redirection of end-user requests.

redirected to the new replica server. The CDN provider might decide to take the replica server off service (in a fully automated manner) when the number of requests decreases.

Let us assume a second scenario in which, after a while, the content provider decides to stream the contents with a feature for adapting the content quality to end-users' devices and network conditions in real time. Using this feature, the end-users with lower bandwidth situations and poor Internet connectivity smoothly receive a lower quality of the content stream. In order to satisfy these requirements, the CDN provider needs to upgrade their service, to support Adaptive Bit Rate (ABR) streaming. The CDN

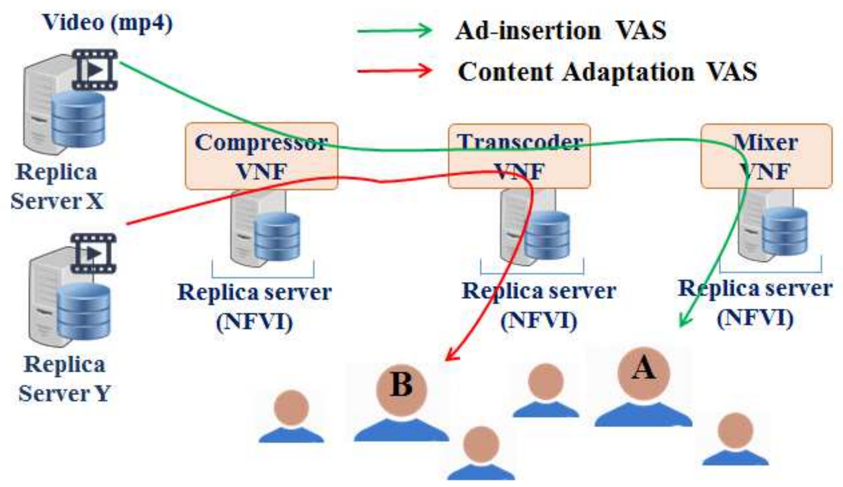


Figure 2.2. Ad-insertion CDN VAS use case.

provider should be able to upgrade their existing replica servers and equip them with the ABR streaming functionality on-the-fly.

2.1.2. On-the-fly provisioning of CDN VASs

Let us assume a content provider decides to provide value-added contents to end-users in addition to their basic services. While CDN basic services include audio/video streaming and audio/video/software download, CDN VASs cover more advanced services such as media management and advertisement insertions (ad-insertions). For an ad-insertion VAS, the CDN provider might insert advertisements for guest log-ins and offer ad-free videos to the registered end-users. Moreover, some end-users may select higher quality streaming for a higher price while others may select lower quality streaming for a lower price. Also, the CDN provider might need to convert the video encoding to be playable by the end-user device based on its capabilities and/or configuration. As shown in Fig. 2.2, the application-level middle-boxes these VASs could rely on are: (i) a mixer for advertisement insertion, (ii) a compressor for decreasing the video size and/or quality, and (iii) a transcoder for video encoding conversions [16]. As shown in Fig. 2.2, for a guest end-user A, who has requested a low-quality video in AVI encoding, a raw MP4-encoded video should then pass through three middle-boxes, i.e. a compressor, a transcoder, and a mixer before being delivered to the end-user. Such middle-boxes are the building blocks for CDN VASs and should be deployed in the network and chained in a specific order to realize the VAS.

2.1.3. Dynamic VAS middle-box placement

The location where the VAS middle-boxes are deployed brings unique challenges in service provisioning as it impacts the CDN provider expenses and the end-user perceived QoS [20].

In NFV settings, such middle-boxes are provisioned in form of VNFs. The VNF-FG for this VAS is formed such that the main video passes through the required VNFs before being delivered to end-users. In concrete CDN scenarios, multiple replica servers

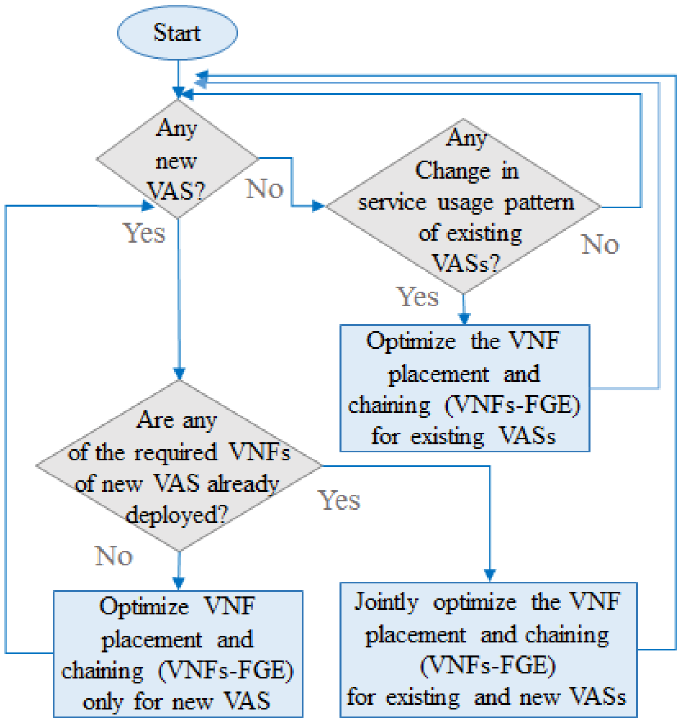

Start
Any new VAS?
No
Yes
Any Change in service usage pattern of existing VASs?
No
Yes
Optimize the VNF placement and chaining (VNFs-FGE) for existing VASs
Are any of the required VNFs of new VAS already deployed?
Yes
No
Optimize VNF placement and chaining (VNFs-FGE) only for new VAS
Jointly optimize the VNF placement and chaining (VNFs-FGE) for existing and new VASs

Figure 2.3. Triggers for VNF topology reconfiguration.

may be capable of serving the end-user requested content according to their geographic location and content availability. For example, in Fig.2.2 both replica servers X and Y can be assigned to end-user A. This leads to a particular feature for CDN VASs: one VNF-FG end-point is unknown prior to the placement [22]. This end-point corresponds to the replica server that serves the content to the end-user a.k.a. content server. The assignment of a replica server to the end-users is critical as it impacts the optimal placement of VNFs in the VNF-FG.

After VNFs are placed and chained to realize a specific VAS, there could be a variety of motivations for modifying the deployment topology of the VNFs in the network. Fig. 2.3 shows the triggers that indicate the need for VNF topology reconfiguration.

A first motivation is the introduction of new VASs. In some cases, new VASs that might share a subset of VNFs with already-deployed VASs are introduced. For example, as shown in Fig. 2.2, let us assume a content adaptation VAS is already deployed in the network and used by end-users such as end-user B. A set of transcoder and compressor VNFs are deployed for content adaptation and chained to customize the content format and size in order to make the video playable by small devices. Next, we assume a new VAS is going to be introduced for ad-insertion. As mentioned earlier, the new ad-insertion VAS requires a compressor to decrease the content size for end-users with poor Internet connectivity, a transcoder VNF for codec conversions, and a mixer VNF to insert an advertisement on top of the main video. In order to provision the new ad-insertion VAS, some of the already-deployed VNFs of the content adaptation VAS, i.e. the compressor and transcoder VNFs, can be reused to reduce the costs. Thereafter, a new VNF type, i.e. a mixer VNF, must be deployed. However, the eventual multiple reuses of compressor and transcoder VNFs may require a reshuffle in their deployment topology so that the QoS of all of the VASs, both the existing and the new ones, are jointly satisfied. As depicted in Fig. 2.3, if any required VNFs of the ad-insertion VAS are not already deployed, the CDN provider needs to optimally place all the required VNFs from scratch, while taking into account the required QoS of only the new VAS end-users.

A second motivation is a change in the service usage pattern as discussed in [21]. For example, a group of end-users might subscribe to an existing VAS. As end-users could be located in various geographical locations, a VNF topology reconfiguration may be needed to jointly satisfy the QoS for existing and newly subscribed end-users.

2.2. Requirements

According to the above-mentioned motivating scenarios, the following requirements are derived. Some requirements are architectural and some of them are specific to algorithms.

2.2.1. Architectural Requirements

In this subsection, first, we define the general requirements on the architectures. Afterward, we define specific requirements for component provisioning and specific requirements for VAS provisioning in CDNs. We define an architecture here as a set of modules and interfaces that enable provisioning of CDN components and VASs.

A) General architectural requirements

Following requirements are defined as general architectural requirements.

Automation: The architecture should guarantee that the whole service provisioning process will be fully automated to enable the dynamic incorporation of required changes. It is critical for the CDNs to be able to automatically provision services especially to cope with flash crowds where sudden changes happen in end-user usage pattern. In such cases, the traditional provisioning of basic services and VASs cannot be acceptable since it is performed manually.

Heterogeneity: The heterogeneity should be taken into account in terms of PoDs resources and infrastructures offered by providers. The architecture should support CDN replica servers and middle-box provisioning on heterogeneous resources. The main reason is that such components should be deployed on PoDs in different locations that might be covered by various providers such as ISPs and homogeneity cannot be expected across different providers.

Using widely-deployed technologies: The architecture should be designed according to the well-adopted and widely-deployed technologies. This is necessary and helps to ensure the interoperability at the level of architectural modules, CDN components, and business entities.

B) Requirements specific to on-the-fly provisioning of CDN components

Several architectural requirements can be naturally derived from the first and second motivating scenarios which are identified as requirements specific to the on-the-fly provisioning of CDN components for basic CDN services.

Supporting the whole provisioning process: The architecture should be able to cover the whole CDN component provisioning phase. This includes pre-deployment and post-deployment procedures in addition to the actual deployment procedure. For example, as part of the pre-deployment procedure, the appropriate PoDs should be selected and as part of the post-deployment procedure, the newly deployed components should be integrated into existing CDNs and filled with content if needed.

On-the-fly upgradability of CDN components: The architecture should support on-the-fly upgradability of the already deployed CDN components with as little complexity as possible. This requirement is essential due to the ever-changing end-user requirements which encourage the CDN providers to adapt their systems to volatile market needs.

C) Requirements specific to on-the-fly provisioning of CDN VASs

The following requirements are considered to be important for provisioning VASs in CDNs.

Flexibility: The architecture should be flexible enough to support chaining of various type of middle-boxes required for provisioning a wide range of VASs. Some types of VASs require specific middle-boxes to be deployed and chained in the network. Application-level middle-boxes are among examples which are required to realize an end-to-end VAS in CDNs.

Minimizing the content customization overhead: Application-level middle-boxes impose some overhead in terms of latency due to the fact that content customization is a time-consuming task per se. Therefore the architecture should introduce features that reduce the end-to-end content delivery latency as much as possible.

2.2.2. Algorithmic Requirements

The following requirements are defined according to the placement challenges in the motivating scenario.

Dynamicity: The algorithm should adapt the system to the new changes to improve the placement performance. It should support the dynamic mode of placement in which the current VNF topology is efficiently reconfigured to accommodate the existing and newly arrived VNF-FG requests.

Supporting a wide range of VNF-FGs: The algorithm should provide VNF placement solutions for wide range of VNF-FGs including the ones with an unknown VNF-FG end-point such as CDN VASs.

VNF chaining: The algorithm should offer features for optimal placement of multiple VNFs in a VNF-FG and chaining them in the specific required order, in addition to supporting the placement of individual VNFs.

2.3. Related Work

2.3.1. Architectures for on-the-fly CDN Component Provisioning

The component provisioning process (including on-the-fly deployment and upgrading) in CDNs remains essential manual nowadays (e.g. Netflix OpenConnect [35]) even if some CDN providers provide CDN component as software packages (Akamai Aura licensed CDN [36]). In this context, the proposals so far rely on generic frameworks, such as ETSI MANO and active networks for on-the-fly functionality deployment. In this subsection, we first discuss approaches based on ETSI MANO, and then we review architectures built based on active networking.

A) ETSI MANO-based approaches

A related work is the ETSI NFV use case architecture for CDNs [37]. It proposes the design and implementation of the CDN components in form of VNFs and motivates the need for dynamic deployment and management of them in a telecommunication network operator domain. Another approach is CDN as a service (CDNaaS) [38], which facilitates the deployment of telco-CDN replica servers. It proposes an architecture to allow content providers to automatically deploy the CDN replica servers in the ISP domain. The ISP receives the content provider requests for replica server deployment, orchestrates the resources, and deploys the replica server as a monolithic functionality. The case study on the replica servers designed as VNFs is another example [39]. Veitch *et al.* in [39] focus on testing the CDN replica servers as VNFs before their deployment, to assess the effect of various factors (e.g. the use of analytics and the level of granularity) on the baseline performance. This work is based on well-adopted NFV technology and tackles the VNF deployment issue from a testing perspective. Although references [37], [38], and [39] satisfy the general architectural requirements on automation, heterogeneity, and using widely-deployed technologies, they do not provide any feature for upgrading CDN components in general. Moreover, they do not support the whole provisioning process.

B) Active network-based approaches

Active networking [40] is an approach that enables the deployment of network functions on network elements such as routers and switches. Examples of such functions include firewalls and video mixers. Srinivasan *et al.* in [41] propose an approach called ActiveCDN that is powered by the active network. It enables the dynamic extension of content delivery services by virtualizing the storage and computation resources of network elements. The ActiveCDN architecture uses NetServ [42] as a network virtualization framework that focuses on the automatic deployment of CDN replica servers. It overlooks the full provisioning phases. ActiveCDN follows active networking technology, which is not a widely-deployed technology and furthermore, it offers no mechanism for the on-the-fly upgrade of replica servers.

2.3.2. Architectures based on NFV and SDN for VAS Provisioning

Several works in the relevant literature investigate the on-the-fly VAS provisioning. Overall, they mainly focus on the dynamic orchestration of IP-level VNFs (e.g. NAT, Load Balancer and Firewall). In this section, we focus on the works done so far on on-the-fly VAS provisioning using NFV and SDN technologies. We first review related works in the CDN space. Then, we discuss the works with a focus on the use of NFV and SDN at large in other domains.

A) NFV and SDN in the CDN domain

Most of the reviewed literature contributes to IP-level routing rather than the application-level chaining required for our approach. For example, Giotis *et al.* in [43] propose an architecture for CDN traffic management that provides a privileged service for premium applications by interacting with an SDN controller. This work is based on NFV and SDN technologies and satisfies automation and heterogeneity requirements by automatic deployment of VNFs on commercial-off-the-shelf (COTS) infrastructure and their dynamic composition. However, the architecture focuses on an IP-level NFV system using Traffic Monitoring VNFs. In what follows, the works on NFV for the CDN domain and those on SDN for the CDN domain are discussed.

Liu *et al.* in [44] propose the use of programmable storage routers instead of replica servers. The programmable storage routers are controlled by an SDN controller augmented by functionalities, such as request re-routing and load balancing. Although this work satisfies the general architectural requirements, the focus is on the automatic deployment of basic services, and composition of the VNFs for CDN VASs are out of the scope of this research.

To the best of our knowledge, research works on the use of SDN in the CDN space have not so far tackled basic video service provisioning. Neither have they discussed the VAS provisioning. As an example, reference [45] only proposes an architecture that enables the CDN providers and Internet Service Providers (ISP) to manage high volume flows.

B) NFV and SDN in domains other than CDN

Several studies investigate the use of NFV and SDN in domains other than CDN. However, their focus remains on the IP-level middle-boxes and network services.

Mouradian *et al.* in [46] propose an NFV-based architecture for provisioning gateways for Virtualized Wireless Sensor and Actuator Network (VWSAN). The VNFs implement protocol converters and information model processors between the VWSAN and end-user applications. This work satisfies our general requirements and also addresses flexibility since it focuses on the application-level VNF development, deployment, and migration. However, VNF chaining is done statically and the dynamic chaining of VNFs is not addressed. Callegati *et al.* in [47] focus on dynamic/automatic chaining, implementing two topology alternatives (Network and Data Link layer), using SDN controller to illustrate the feasibility of dynamic VNF chaining. Their VNFs include IP-level VNFs such as Deep Packet Inspection (DPI), NAT, and firewall. This architecture mainly focuses on a dynamic chaining of IP-level VNFs, therefore it is not flexible in supporting all type of middle-boxes. Ding *et al.* in [48] propose a solution called OpenSCaaS to enable the VNF dynamic chaining. It satisfies the heterogeneity requirement as it proposes an architecture of a flexible and adaptable VNF deployment and chaining and it packages VNFs into lightweight ClickOS VM images [49]. OpenSCaaS focuses on generic IP-level VNFs, such as NAT and Firewall.

Several works have so far focused on SDN in domains other than CDN. In our literature study, we have focused on the relevant works to our contributions, i.e. those that concern the application layer context. Paul *et al.* in [50] propose OpenADN. OpenADN provides a platform for flow management based on the application-level context, such as end-user device type, link conditions, and mobility. OpenADN is a technology-based work, however, it does not focus on dynamic chaining of application-level middle-boxes. Another example is Tegueu *et al.* [51], that proposes an approach called ADN. This work offers SDN-based architecture and algorithms for optimal network resource allocation based on the application QoS requirements. In particular, ADN translates the application QoS needs (e.g. bandwidth and delay) into OpenFlow rules and then employs the SDN controller to inject IP-level OpenFlow routing rules in the SDN switches. Dolberg *et al.* in [52] propose another example of an SDN-based framework for mapping application-level policies to IP-level flow rules, leading to a differentiated network service per application. Although the above-mentioned works employ well-adopted SDN technology for application-level context, they lack features for flexible support of all VNF-types including application-level ones.

2.3.3. Algorithms proposed for VNF placement

Here we review the relevant literature on VNF placement. The first subsection is focused on the works done to date on VNF-FG embedding for CDN VASs, and in the second subsection, we review the related work on VNF placement in CDNs at large. The third subsection reviews the dynamic VNF placement proposals in other domains.

A) VNF-FG embedding for CDN VASs

Although VNF-FG embedding (VNF-FGE) has recently attracted the attention of researchers [53][54] [55], very few research efforts have focused on VNF placement for CDN VASs. One example is Ahvar *et al.* [23], which models the problem of VAS VNF placement as an ILP and proposes a static VNF-FGE algorithm for the placement of CDN VAS VNFs. Their solution focuses on minimizing cost while respecting the delay thresholds of VNF-FG requests. The solution has two main phases. In the first phase a minimum number of VNF instances are placed and chained in a network so that they cover all end-users; in the second phase, the placement is modified and improved to meet the constraints of server capacity, VNF processing capacity, and the end-user perceived QoS in terms of content delivery latency.

Dieye *et al.* in [22] propose an ILP and an algorithm for VNF-FGE for CDN VASs. They considered the specific requirement of CDN VAS VNF-FGs that have an unknown end-point and accordingly modeled the problem as an ILP formulation. They propose a static VNF placement algorithm based on the PageRank algorithm in which the replica servers that will host the VNFs are sorted according to their importance; i.e. their available hosting capacity and the quality of their links. They sort the VNF-FG requests according to their QoS requirements and try to accommodate the VNF-FG requests with strict requirements first while taking into account the VNF and server capacity and QoS constraints. Accordingly, they minimize the total cost by finding a trade-off between the optimal number of VNF instances and servers used and the QoS guarantee. Both [22] and [23] propose static VNF placement solutions with cost objectives containing multiple components, including hosting, routing, and VNF instantiations costs.

B) VNF-FG embedding for CDNs at large

Herbaut *et al.* propose a solution for the placement of replica servers in CDNs [24]. They define CDN replica servers as VNFs and focus on placing them on NFVIs provided by ISPs. They model the collaboration between ISPs and CDNs as an ILP. To increase the problem's tractability, they propose a heuristic for optimal VNF placement that is defined in three phases. In the first phase, the algorithm places CDN controllers and assigns them to client groups. In the second phase, a set of virtual CDNs, defined as VNFs, are placed and assigned to CDN controllers. In the third and last step, the network links are mapped to the service edges, taking into account the delay and bandwidth between nodes. This work proposes a solution that targets the minimization of the routing and VNF hosting costs and focuses on static VNF placement.

Another example is Benkacem *et al.* [56] that proposes a solution for the deployment and management of virtual CDN slices over resources offered by cloud providers. Each slice contains a set of VNFs such as video caches, streamers, and transcoders. The optimal placement of VNFs improves the slice performance. Each slice defines the location of end-users, the minimum QoS that the end-users of that slice should experience, and the required VNFs for the slice. This work proposes three optimization models. The first one aims at minimizing the cost of resources (e.g. vCPU and memory) for hosting VNFs, the second one targets QoS maximization in terms of the quality of content that end-user receives, and the third solution uses an approach based on game theory to offer a trade-off between the QoS and the cost.

Another example is Ibn-Khedher *et al.* [57] that focuses on the optimal placement of CDN replica servers defined as VNFs. This work proposes an ILP for dynamic VNF placement and their optimal assignment to the end-users while considering the server and link capacity constraints. It targets minimizing the VNF migration cost in the ILP model; however, new VNF instantiation, hosting and routing costs are overlooked.

C) Dynamic VNF placement in other domains

Here we explore the dynamic solutions for VNF placement in domains other than CDNs. Some works focus on the placement of individual VNFs and others propose mechanisms for VNF placement and chaining.

The studies in [58] [59] focus on the dynamic placement of individual VNFs. Abu-Lebdeh *et al.* in [58] define the VNF Managers (VNFMs) in the form of VNFs and model the problem of the dynamic placement of VNFMs on NFV Infrastructure (NFVI) and their optimal assignment to demands, i.e. VNFs. They consider the migration, routing, and hosting cost in this model and propose a Tabu-based heuristics to solve this problem. Ghaznavi *et al.* in [59] propose a solution for the dynamic placement of VNFs in data centers in response to changing workloads. They present an ILP formulation and heuristics and aim to jointly minimize the migration, hosting and routing costs. They propose heuristics for handling demand arrivals and departures while considering the reuse of existing VNFs, migrating them, or instantiating new VNF instances. The works in [58] [59] focus on the placement of individual VNFs and overlook features for the handling of VNF chains.

The studies in [60] and [61] target the dynamic placement of chains of VNFs. In [60], Xia *et al*. investigate the placement of VNF chains and address the problem of the migration of VNFs in data centers to meet computing and network resource constraints. Their objective is to minimize the VNF migration cost while satisfying the server capacity and delay constraints. They propose a three-step heuristic that in the first step finds the VNFs that need to be migrated due to resource deficiency, in the second step identifies the target nodes that can host those VNFs and finally migrates the VNFs in the third step. This work targets the optimal migration of the VNFs of chains in response to workload changes. Similarly, Eramo *et al.* in [61] consider the placement of multiple VNFs in VNF-FGs. They model the problem as an ILP and propose a heuristic to decide about when and where to migrate the VNFs, utilizing server consolidation to reduce energy consumption while minimizing the migration cost. The approaches proposed in Refs. [60] and [61] are not applicable to VAS chain placement since they do not consider endpoint unknowingness.

2.4. Conclusion

This chapter presented motivating scenarios for provisioning of CDN components and VASs. Accordingly, some requirements are derived followed by evaluation of the related work against the requirements. As discussed, none of the research proposals done to date can satisfy all the defined requirements. Table 2.1 summarizes the related work evaluations with respect to the derived requirements.

Table 2.1. Related work evaluation

Requirements / Related Works	Architectural Requirements							Algorithmic Requirements		
	General Architectural Requirements			Requirements for Component Provisioning		Requirements for VAS Provisioning				
	Automation	Heterogeneity	Widely-deployed technologies	Support the whole provisioning	upgradability	Flexibility in supporting all middle-boxes	Minimize content customization overhead	Dynamicity	Support all VNF-FG Types	VNF Chaining
ETSI Use cases [37]	Yes	Yes	Yes	No	No	-	-	-	-	-
Frangoudis *et al.* [38]	Yes	Yes	Yes	No	No	-	-	-	-	-
Veitch *et al.* [39]	Yes	Yes	Yes	No	No	-	-	-	-	-
Srinivasan *et al.* [41]	Yes	No	No	No	No	-	-	-	-	-
Giotis *et al.* [43]	Yes	Yes	Yes	-	-	No	No	-	-	-
Liu *et al.* [44]	Yes	Yes	Yes	-	-	No	No	-	-	-
Mouradian *et al.* [46]	Yes	Yes	Yes	-	-	No	No	-	-	-
Callegati *et al.* [47]	Yes	No	Yes	-	-	No	No	-	-	-
Ding *et al.* [48]	Yes	Yes	Yes	-	-	No	No	-	-	-
Paul *et al.* [50]	Yes	No	Yes	-	-	No	No	-	-	-
Tegueu *et al.* [51]	Yes	No	Yes	-	-	No	No	-	-	-
Dolberg *et al.* [52]	Yes	No	Yes	-	-	No	No	-	-	-
Ahvar *et al.* [23]	-	-	-	-	-	-	-	No	No	Yes
Dieye *et al.* [22]	-	-	-	-	-	-	-	No	Yes	Yes
Herbaut *et al.* [24]	-	-	-	-	-	-	-	No	No	No
Benkacem *et al.* [56]	-	-	-	-	-	-	-	No	No	Yes
Ibn-Khedher *et al.* [57]	-	-	-	-	-	-	-	Yes	No	No
Abu-Lebdeh *et al.* in [58]	-	-	-	-	-	-	-	Yes	No	No
Ghaznavi *et al.* [59]	-	-	-	-	-	-	-	Yes	No	No
Xia *et al.* [60]	-	-	-	-	-	-	-	Yes	No	Yes
Eramo *et al.* [61]	-	-	-	-	-	-	-	Yes	No	Yes

Chapter 3

3. An Architecture for on-the-fly Provisioning of CDN Components

3.1. Introduction

As mentioned earlier, during some events, specific contents might get a sudden and unpredicted popularity in a short period of time which might lead to flash crowds that eventually cause an outage in CDNs, i.e. site slowdowns, service unavailability, and a decrease in QoS. World Cup 1998 [62] or terrorist attacks of September 11, 2001 [7], are among flash crowds examples. In the case of flash crowds, traditional CDNs are challenged to extend their coverages by provisioning additional CDN components e.g. replica servers in new locations. Beyond flash crowds, CDN components need to be upgraded frequently, due to the fact that content delivery is a volatile market driven by new formats, protocols, content protection requirements. Introduction of new Adaptive

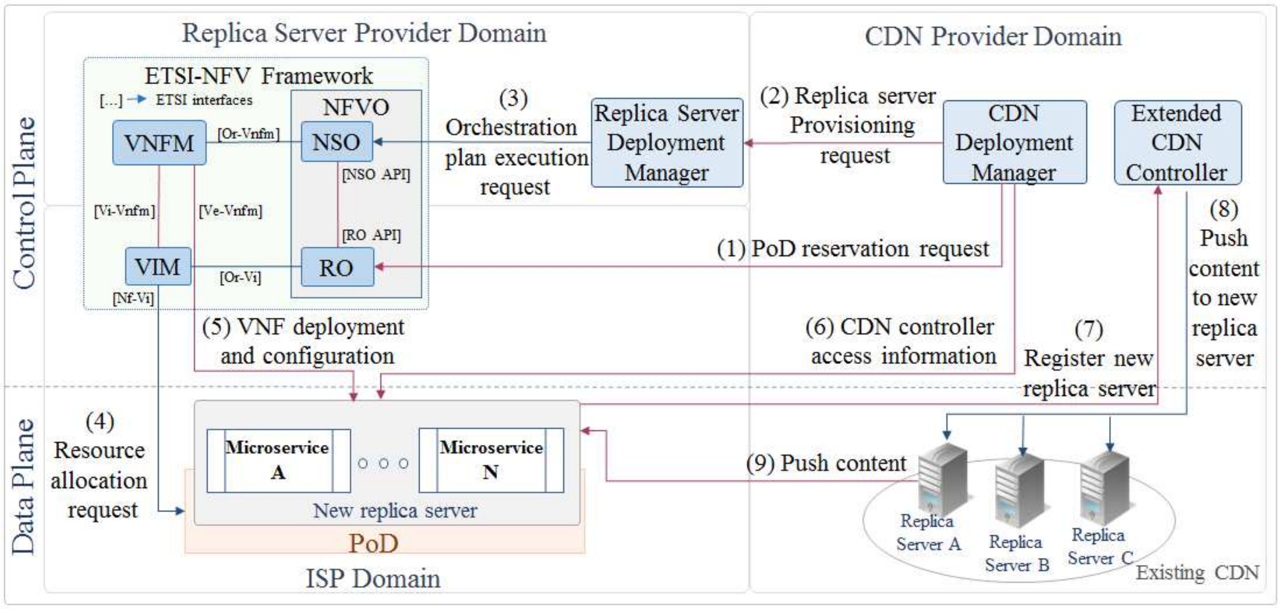

Replica Server Provider Domain
CDN Provider Domain
Control Plane
Data Plane
ETSI-NFV Framework
[...] → ETSI interfaces
NFVO
VNFM
[Or-Vnfm]
NSO
[NSO API]
[Vi-Vnfm]
[Ve-Vnfm]
[RO API]
VIM
[Or-Vi]
RO
[Nf-Vi]
(3) Orchestration plan execution request
Replica Server Deployment Manager
(2) Replica server Provisioning request
CDN Deployment Manager
Extended CDN Controller
(1) PoD reservation request
(8) Push content to new replica server
(5) VNF deployment and configuration
(6) CDN controller access information
(7) Register new replica server
(4) Resource allocation request
Microservice A
Microservice N
New replica server
PoD
ISP Domain
(9) Push content
Replica Server A
Replica Server B
Replica Server C
Existing CDN

Figure 3.1. High-level architecture for CDN component provisioning

Bitrate (ABR) streaming approaches such as MPEG-DASH in 2012 [63] and HTTP Live Streaming (HLS)-v7, in 2017 [64], are among examples that require an upgrade in CDNs. In conventional CDNs, the process of replica server provisioning requires labor-intensive manual efforts [65]. In this context, automation remains essential, especially to cope with sudden changes in service usage pattern and CDN provider requirements.

This chapter proposes an architecture that uses microservices architectural style [13] and NFV technology [15] to enable the on-the-fly provisioning (including deployment and upgrading) of CDN components, e.g. replica servers. Microservices architectural style is a paradigm in which several small loosely-coupled modules, called microservices, are composed to realize the functionality of an application. The modular design of microservices brings flexibility, scalability and enables the reuse of modules by multiple applications. Moreover, the microservice design facilitates the replica server upgrades, as microservices can be deployed, upgraded, and disposed independently.

NFV is an emerging technology that targets the decoupling network functions from their underlying hardware. NFV is used to eliminate the dependency between the network function software and their underlying hardware and implements the network function in form of stand-alone software functionalities called Virtualized Network Function (VNF) that are executable on any generic commercial hardware. NFV enables on-the-fly deployment and upgrade of CDN components on any generic hardware, anytime and anywhere in the network.

3.2. The Proposed Architecture

Fig. 3.1 shows a high-level view of our proposed architecture. To be specific, in this chapter we consider replica servers as main CDN components that need to be deployed or upgraded. The reader should note that the business model in this chapter includes *replica server provider* as a new actor, in addition to traditional CDN business actors. The *replica server provider* provisions (including deployment and upgrade) replica servers to *CDN providers*. The *replica server provider* provides replica servers as a set of microservices packaged as VNFs and deploys them on the selected PoDs. The PoDs provide the infrastructure and resources needed for replica server hosting and might be possessed and managed by either *CDN providers* or *ISPs*. This chapter focuses on the second alternative.

The proposed architecture reuses ETSI MANO framework and consists of a control and a data plane. The control plane handles the flow of control messages between modules in the *CDN provider*, *ISP*, and the *replica server provider* domains to enable PoD selections and replica server provisioning. The data plane allows the flow of content between replica servers and end-users.

In this section, we provide our core architectural principles, then we present the proposed microservice-based replica server, and afterward, the main modules of our architecture, their functionalities, and interactions will be discussed. Then a sequence diagram is presented to illustrate the interactions between modules.

3.2.1. Architectural Principles

The following design principles are considered:

- The first architectural principle is the design of replica server as a set of microservices. As mentioned earlier the microservice architectural style can be used to define replica servers in form of small and loosely-coupled functionalities. This will facilitate the replica server upgrades, as microservices can be deployed, updated, and disposed independently.
- The second architectural principle is the use of REpresentational State Transfer (REST) architectural style for designing the interactions between the microservices. REST is used because it is a lightweight, technology-neutral, and standards-based design style for data representation, allowing us to describe APIs in a generic and abstract way.
- The third architectural principle is the use of NFV technology for automatic deployment of microservices that are packaged as VNFs. NFV makes functionalities deployable on any generic hardware and facilitate their deployment and management.

3.2.2. Design of microservice-based replica server

As noted in [66], a microservice architecture can be defined as "*an approach to developing an application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery*." In the CDN context, a replica server can be the application mentioned in the definition.

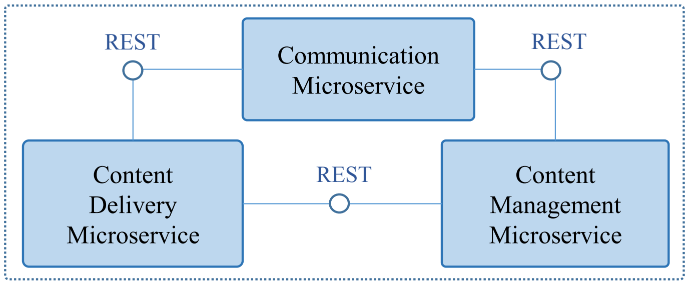

REST
Communication Microservice
REST
Content Delivery Microservice
REST
Content Management Microservice

Figure 3.2. The microservice design of a generic replica server.

The microservices independency is an essential characteristic of microservice architecture. Accordingly, in order to design a microservice-based replica server, we divide the functionality of the replica server into logical independent microservices, such that each microservice focuses on completing an independent task. The division is done in a way that each microservice operates in a fully autonomous manner with process and data isolation.

In this chapter, we propose a potential splitting of a generic replica server into the following three fundamental microservices: *communication microservice, content delivery microservice,* and *content management microservice*. It should be noted that upgrading the generic replica server may require the deployment of additional microservices. For instance, if this generic replica server were to be upgraded for the support of ABR streaming functionality, an additional microservice, the *ABR video generator microservice*, will be required. These three fundamental microservices are briefly discussed in the next paragraphs.

The *communication microservice* is designed to allow the newly deployed replica server to communicate with external entities. It essentially encompasses the protocol stacks required to enable the replica server to communicate with the end-users and also with the other entities of the CDN network. For instance, in the case of upgrading from HTTP v1 to HTTP v2, it will be this very microservice which will be upgraded. If the other microservices which previously used HTTP v1 are going to use the new features brought by HTTP v2, they will also need to be upgraded accordingly.

The *content management microservice* manages the local content repository and makes the requested content available. It interacts with other replica servers to get the requested content in case this content is not available in the local repository. These interactions are done via the *communication microservice*. It should be noted that this microservice will need to be upgraded if the ABR functionality is introduced.

The *content delivery microservice* manages end-user profiles including tracking their behavior and usage patterns. It handles the actual delivery of the content made available by the *content management microservice*. This delivery is done using the *communication microservice*.

Fig. 3.2 illustrates the interactions between these three fundamental microservices. It should be noted that the proposed design considers the core functionalities of replica servers and can be further extended by considering additional microservices to perform functionalities such as authentication, billing, and etc. The reader should also note that the above-mentioned microservices could be further broken into smaller components, although not done in this chapter.

3.2.3. High-Level Architecture

Some architectural modules are reused from ETSI MANO (as they stand) while some others are newly introduced.

A) Modules for replica server and associated microservice lifecycle and resource management

We have reused the following ETSI MANO modules for management of replica servers and their associated microservices: NFVO, VNFM, and VIM. We separate the NFVO functionality into Network Service Orchestrator (NSO) and Resource

Orchestrator (RO) as proposed as an option in the ETSI specification on architectural options for MANO [67].

1. NFV Modules in replica server provider domain

The *NSO* and the *VNFM* are located in the replica server provider domain. The *NSO* is in charge of the replica server lifecycle management, including deployment, configuration, upgrade, and disposal of the replica servers. The *VNFM* handles the microservice VNF lifecycle management, including instantiation, upgrade, and termination of microservices.

2. NFV Modules in ISP domain

The *RO* and the *VIM* are located in the ISP domain. The *RO* is in charge of ISP resource orchestrations and reservations. It receives the resource reservation requests from NSO and interacts with *VIM* for resource allocations and hardware configurations. The *VIM* is responsible for allocating resources to microservices to be deployed and executed, and also is in charge of releasing the virtualized resources when microservices are disposed.

B) Modules for replica server provisioning

The *Replica Server Deployment Manager* is located in the replica server provider domain. It is responsible for receiving and analyzing the replica server provisioning requests, selecting the required microservices, and requesting the execution of required orchestration plans. After a successful provisioning process, it sends the information of the newly deployed replica server back to the CDN provider domain.

The *CDN Deployment Manager* and the *Extended CDN Controller* are located in the CDN provider domain. The *CDN Deployment Manager* handles control message exchanges with the ISP domain for the PoD assignment purpose. It also communicates with the replica server provider domain to request the provisioning (i.e. deployment,

Table 3.1. Examples of REST modeling for CDN component provisioning

Interaction	Operation	REST Resource	HTTP Action and Resource URI	Request Body Parameters	Most Important Info in the Response
CDN Deployment Manager to Replica Server Deployment Manager {Interaction# 2 in Fig. 2}	Deploy a replica server on a PoD	Replica server	POST: /ReplicaServers/{ReplicaServerTypeID}	PoD access information	New replica server IP
	Upgrade a replica server	Replica server	PUT: /ReplicaServer/{ReplicaServersID}	New replica server type	New replica server IP
	Dispose a replica server	Replica server	DELETE: /ReplicaServers/{ReplicaServersID}	Replica server ID	Success/ failure
New Replica Server to Extended CDN Controller {Interaction# 7 in Fig. 2}	Register	Replica server	POST: /ReplicaServers	Replica server info, e.g. IP, location	Success/ failure
Extended CDN Controller to existing Replica servers {Interaction# 8 in Fig. 2}	Replicate contents to a replica server	Content	POST: /Contents	List of contents and access info of new replica server	Success/ failure

upgrade, and disposal) of replica servers. An example of REST modeling related to this interaction is provided in Table 3.1. In addition, as part of the post-deployment procedure, it enables the integration of the newly deployed replica servers into the CDN by providing them the CDN controller access information.

The *Extended CDN Controller* extends the functionality of the traditional CDN controller, i.e. redirecting the end-user requests. The extensions enable the automatic integration of the newly deployed replica server in the existing CDN, while in the traditional CDN, the integration is manual. This module is in charge of registering and integrating the newly deployed replica server into the existing CDN. It also has the responsibility of replicating the appropriate content in the newly deployed replica server. An example of REST modeling related to the interactions with the *Extended CDN Controller* is provided in Table 3.1. For example for interaction #7 in Fig. 3.2, the replica server is defined as a REST resource and for this resource, the POST operation is defined to register the replica server in the *Extended CDN Controller*.

3.2.4. Illustrative Sequence Diagrams

In this subsection, we illustrate some sequence diagrams. The diagrams focus on scenarios discussed in subsection 2.1.1. In the illustrated scenarios, a CDN provider notices a surge of traffic from a given area and decides to extend their coverage by provisioning a generic replica server in a new location. After some time, it decides to upgrade the generic replica server to support ABR streaming capabilities.

A) CDN component pre-deployment and deployment processes

The pre-deployment and deployment procedures are shown in Figure 3.3. During the pre-deployment procedure, CDN deployment manager sends requests to ISP to access their available PoDs in a specific location (Figure 3.3, action 1). The PoD access information is retrieved from the ISP domain (action 2). After that, the orchestration process, including the microservice deployment and configuration procedures takes place (actions 3-18). The orchestration starts with resource reservations in the specified PoD (actions 5-8). After resources are reserved successfully, the deployment procedure is done in line with the ETSI NFV standard. We assume the required microservices for a generic replica server are a communication microservice, a content management microservice, and a content delivery microservice, according to the discussion in subsection 3.3.2. In line with ETSI NFV MANO, the VNFM deploys the microservices that are packaged in form of VNFs prior to the deployment. After deployment of microservices, some configurations in microservices are done (actions 14-16) by VNFM as well. The configuration is needed to enable microservices to interact with each other and hence operate as a full-fledged replica server.

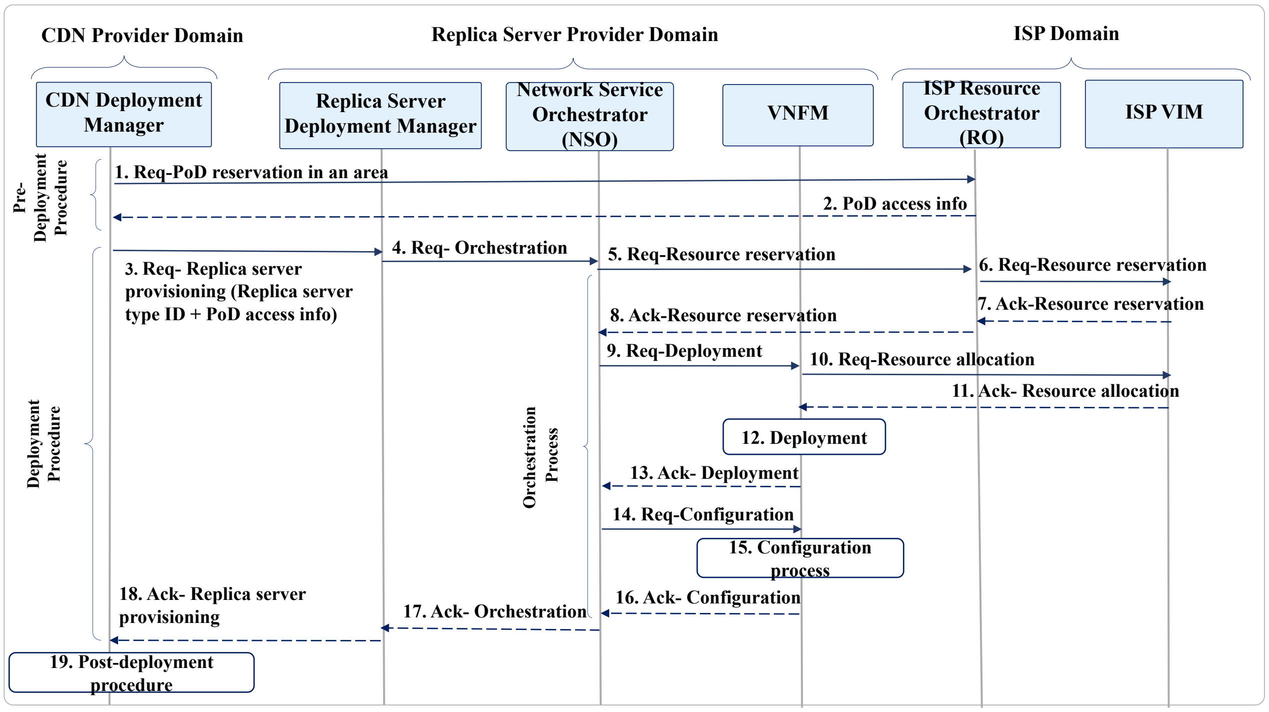
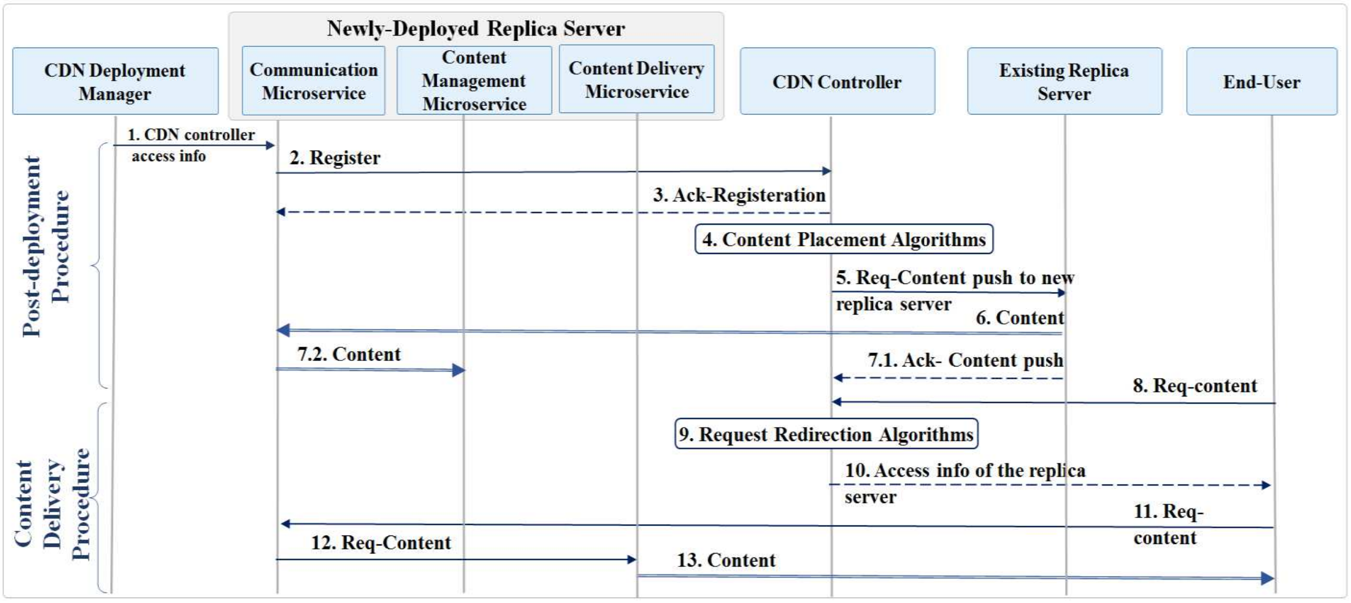

CDN Provider Domain
Replica Server Provider Domain
ISP Domain
CDN Deployment Manager
Replica Server Deployment Manager
Network Service Orchestrator (NSO)
VNFM
ISP Resource Orchestrator (RO)
ISP VIM
Pre-Deployment Procedure
Deployment Procedure
Orchestration Process
1. Req-PoD reservation in an area
2. PoD access info
3. Req- Replica server provisioning (Replica server type ID + PoD access info)
4. Req- Orchestration
5. Req-Resource reservation
6. Req-Resource reservation
7. Ack-Resource reservation
8. Ack-Resource reservation
9. Req-Deployment
10. Req-Resource allocation
11. Ack- Resource allocation
12. Deployment
13. Ack- Deployment
14. Req-Configuration
15. Configuration process
16. Ack- Configuration
17. Ack- Orchestration
18. Ack- Replica server provisioning
19. Post-deployment procedure

Figure 3.3. Sequence diagram of the on-the-fly CDN component provisioning.

B) Post-deployment and content delivery processes

The detailed post-deployment and content delivery phases are shown in Figure 3.4. After the deployment, the newly-deployed replica server should be integrated into the existing CDN. Therefore, it gets registered into the CDN controller that is in charge of the management of the whole CDN (Fig. 3.4, actions 1-3). Afterward, the newly-deployed replica server needs to be filled with popular content in that area. Therefore, during the post-deployment phase, some content placement algorithms might be re-executed by the CDN domain (Figure 3.4, action 4). This helps to decide about the contents that should be placed on the newly-deployed replica server and might lead to the migration of some content from the existing replica servers to save storage. It should be noted that for this scenario, we assume the CDN controller uses a cooperative push-based mechanism to proactively push the popular contents to the newly-deployed



Newly-Deployed Replica Server
CDN Deployment Manager
Communication Microservice
Content Management Microservice
Content Delivery Microservice
CDN Controller
Existing Replica Server
End-User
Post-deployment Procedure
Content Delivery Procedure
1. CDN controller access info
2. Register
3. Ack-Registeration
4. Content Placement Algorithms
5. Req-Content push to new replica server
6. Content
7.2. Content
7.1. Ack- Content push
8. Req-content
9. Request Redirection Algorithms
10. Access info of the replica server
11. Req-content
12. Req-Content
13. Content

Figure 3.4. Sequence diagram of post-deployment and content delivery procedures.

replica server (actions 5, 6, and 7); however, other practices are also applicable. The communication microservice receives and passes the content to the content management microservice to save the content in the local content repository.

The content delivery process is done similar to traditional CDNs as shown in Figure 3.4, actions 9 to 14. The end-user triggers a request for a specific content (action 8). The CDN controller redirects the end-user request to the appropriate replica server according to the content availability of replica servers, network conditions, and the geographic location of the end-user (actions 9-11). At the newly-deployed replica server, the request is received by the communication microservice and is passed to the content delivery microservice (action 12). The content delivery microservice retrieves the requested content from the local content repository and sends it back to the end user (action 13).

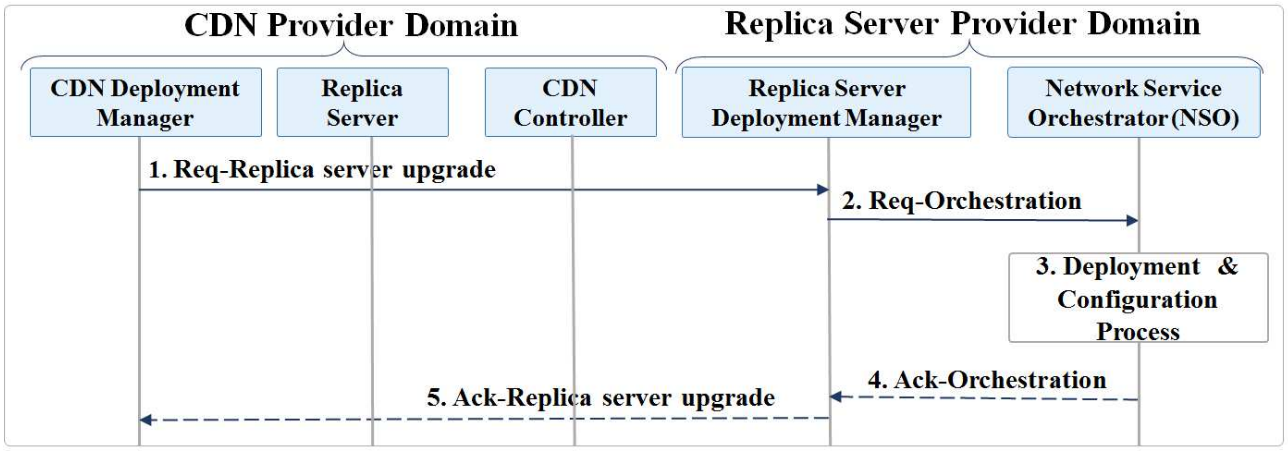

CDN Provider Domain
Replica Server Provider Domain
CDN Deployment Manager
Replica Server
CDN Controller
Replica Server Deployment Manager
Network Service Orchestrator (NSO)
1. Req-Replica server upgrade
2. Req-Orchestration
3. Deployment & Configuration Process
4. Ack-Orchestration
5. Ack-Replica server upgrade

Figure 3.5. Sequence diagram of CDN component upgrade procedure

C) CDN component upgrade process

As discussed in the scenarios in subsection 2.1.1, we assume after a while, the content provider decides to stream the contents with a feature for adapting the content quality to end-users' devices and network conditions in real time. The CDN provider needs to upgrade their replica servers to support ABR streaming features. The upgrade procedure of the CDN components is depicted in Figure 3.5.

The CDN deployment manager triggers a request for the upgrade of an existing replica server (Fig. 3.5, action 1). Replica server deployment manager receives and analyzes the replica server upgrade request, selects the required microservices, and requests the execution of required orchestration plans (action 2). It sends the orchestration request to the NSO and the upgrade procedure takes place by executing the appropriate orchestration plan (actions 2-4).

The upgrade procedure involves un-deploying some microservices if needed and choosing the new microservices (e.g. an ABR video generator microservice for ABR streaming) to be deployed and configured.

3.3. Implementation and Validation

In this Section, the design of the prototype architecture and settings are presented first. Second, the measurements performed on the prototype are presented followed by related discussions.

The implementation covers the illustrative scenarios discussed in the previous subsection including the replica server provisioning (i.e. the pre-deployment, deployment, post-deployment, and upgrade process) and the content delivery process.

We consider an implementation scenario in which a replica server in Toronto is in charge of delivering the contents of a content provider to end-users in Canada. Let us assume an unpredicted event occurs in British Columbia (e.g. British Columbia floods in 2017), and a specific content gains popularity in a short period of time. The CDN provider notices a surge in traffic from Vancouver. In order to prevent flash crowds, the CDN provider decides to extend their coverage by provisioning a new generic replica server close to the end-users in that area.

The provisioning process is implemented from the time the CDN provider requests the deployment of new replica server to the time that the replica server is fully operational. An upgrade scenario is also implemented where after a while the CDN provider decides to upgrade the already deployed generic replica server in order to support the ABR streaming.

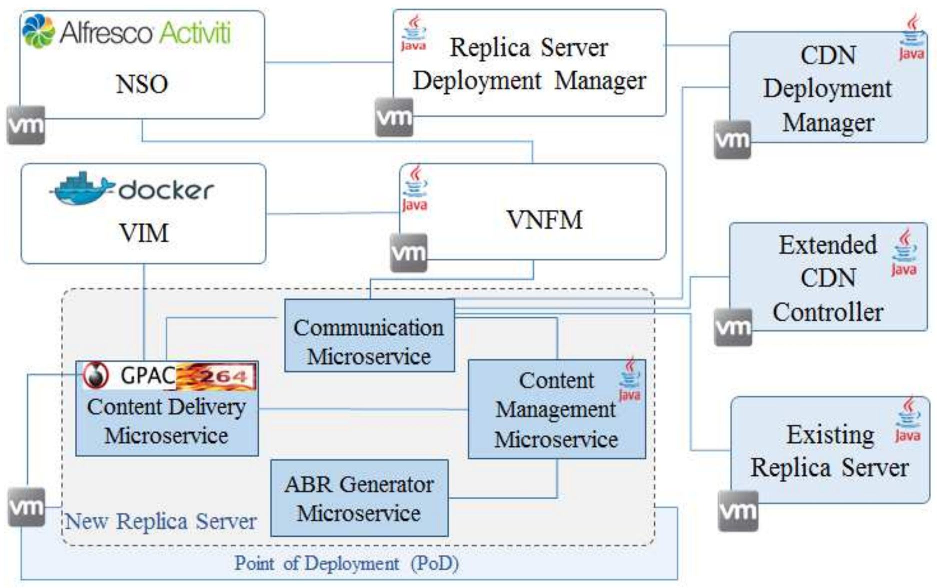

Alfresco Activiti
NSO
Replica Server Deployment Manager
CDN Deployment Manager
docker
VIM
VNFM
Extended CDN Controller
Communication Microservice
GPAC 264
Content Delivery Microservice
Content Management Microservice
Existing Replica Server
ABR Generator Microservice
New Replica Server
Point of Deployment (PoD)

Figure 3.6. The developed prototype architecture

3.3.1. Prototype Architecture

The prototype was implemented according to the architecture depicted in Fig. 3.6. All of the modules are deployed on Virtual Machines (VMs) provisioned by OpenStack IaaS Manager, provided by Smart Applications on Virtual Infrastructure (SAVI) testbed [68] that is a Canadian distributed platform for future Internet applications and has been used for the validation purpose.

For the *NSO*, Alfresco Activiti [69] Business Process Management (BPM) solution is used. It exposes REST API for execution of the orchestration plans and workflows that are defined in Business Process Model Notation (BPMN). The PoDs are equipped with Ubuntu operating system and have Docker platform [70] installed. We have used a container-based microservice orchestration because containers are lightweight, modular, and fast in deployments. The *RO* is not implemented as we assume all domains share a common understanding of replica servers and their required resources. All other architecture modules and microservices are modeled as Docker containers and pushed to the DockerHub [71] repository. For evaluating the upgrade process, an ABR video generator microservice is implemented in accordance with the MPEG-DASH [63] standard. The X264 video codec and GPAC MP4Box tools are used to enable the ABR video generation. To compare our microservice-based solution with those with a monolithic design, a monolithic replica server is also implemented and packaged in form of a VNF in a Docker container. The interaction between all modules and also microservices are enabled via the Wide Area Network (WAN) over the Internet. RESTful web services and developed using Java-based Restlet API. All microservices are packaged in form of VNFs.

All of the architecture modules, including the *CDN deployment manager*, the *extended CDN controller*, the *replica server deployment manager*, the *existing replica servers*, and *MANO* components are deployed in the SAVI-Toronto site. The PoD and end-users are located in the SAVI-Vancouver site. For each architecture module, a medium OpenStack VM is used with two vCPUs, a 40GB disk and 4GB of memory for execution.

3.3.2. Performance Evaluations

1) Performance Metrics

The performance metrics according to which we evaluate our system's performance are:

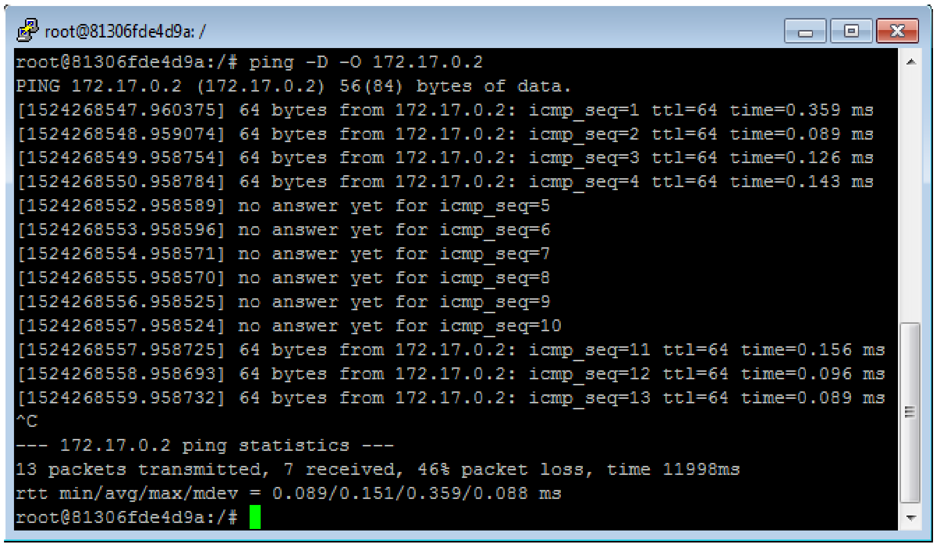
- *Deployment time:* The duration from when the *CDN deployment manager* requests the deployment of a new replica server and the moment that the newly-deployed replica server becomes fully operational. This consists of the required time for deployment and post-deployment procedures, including the actual deployment, the integration of the newly deployed replica server into the existing CDN, and filling it with the popular content.

- *Upgrade time:* The duration from the time the *CDN deployment manager* requests the upgrade of an existing generic replica server to support ABR streaming to the time that the upgraded replica server becomes fully operational. It should be noted that in the upgrade time for the monolithic case, the required time for on-the-fly disposal of the old replica server is also included.
- *Service downtime:* The duration in which the replica server is off-service as a result of the upgrade process. During this period, the replica server cannot be responsive to the end-user requests.
- *Content delivery time:* The duration between the time that an end-user in Vancouver sends a content request to a newly-deployed replica server in Vancouver site and the time he/she receives the requested content (a 1 MB video).

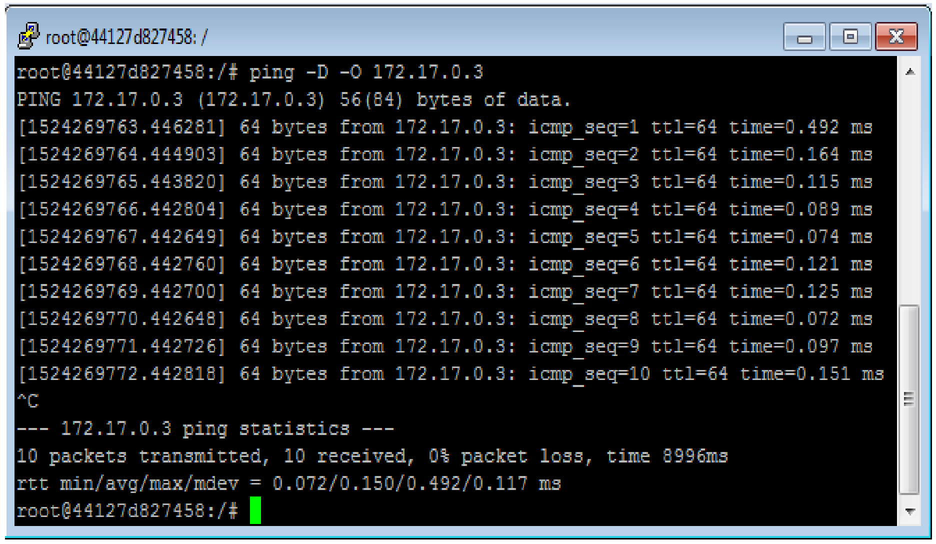
2) Measurements and discussions

We implemented four scenarios for performance measurements. Two of these scenarios are for the upgrade of a generic replica server to support ABR streaming. The replica server is implemented as a monolithic function in one and as a microservice-based function in the other. Another two scenarios are for the deployment of a generic replica server. Again, the replica server is implemented as a monolithic function in one and as a microservice-based function in the other. All the experimentations including the deployment and upgrade of both monolithic and microservice-based replica servers are enabled in the same environment using our developed prototype. Note that the post-deployment phase is also included in the deployment of the replica servers. In order to ensure an accurate comparison, all measurements are repeated 10 times and the average and standard deviation for each set of measurements are provided.

Fig. 3.7(a) and 3.7(b) depict the pinging results of upgrading a generic replica server to support ABR streaming. We started pinging the replica servers through ICMP before the upgrade process is requested by the CDN deployment manager, and it lasted until the upgrade process was completed. As shown in Fig. 3.8, it can be observed that the


root@81306fde4d9a: /
```
root@81306fde4d9a:/# ping -D -O 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
[1524268547.960375] 64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.359 ms
[1524268548.959074] 64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.089 ms
[1524268549.958754] 64 bytes from 172.17.0.2: icmp_seq=3 ttl=64 time=0.126 ms
[1524268550.958784] 64 bytes from 172.17.0.2: icmp_seq=4 ttl=64 time=0.143 ms
[1524268552.958589] no answer yet for icmp_seq=5
[1524268553.958596] no answer yet for icmp_seq=6
[1524268554.958571] no answer yet for icmp_seq=7
[1524268555.958570] no answer yet for icmp_seq=8
[1524268556.958525] no answer yet for icmp_seq=9
[1524268557.958524] no answer yet for icmp_seq=10
[1524268557.958725] 64 bytes from 172.17.0.2: icmp_seq=11 ttl=64 time=0.156 ms
[1524268558.958693] 64 bytes from 172.17.0.2: icmp_seq=12 ttl=64 time=0.096 ms
[1524268559.958732] 64 bytes from 172.17.0.2: icmp_seq=13 ttl=64 time=0.089 ms
^C
--- 172.17.0.2 ping statistics ---
13 packets transmitted, 7 received, 46% packet loss, time 11998ms
rtt min/avg/max/mdev = 0.089/0.151/0.359/0.088 ms
root@81306fde4d9a:/#
```

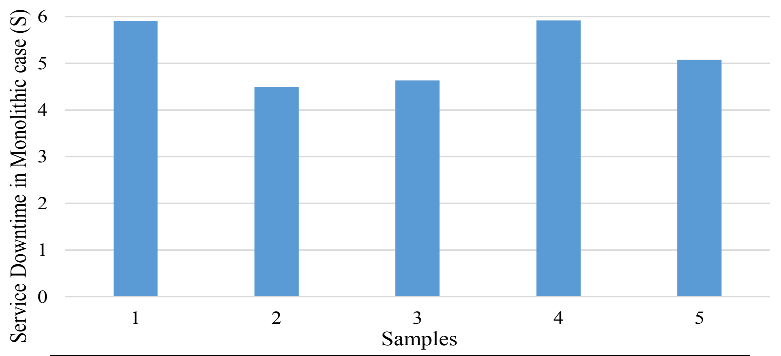
(a)


root@44127d827458: /
```
root@44127d827458:/# ping -D -O 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
[1524269763.446281] 64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.492 ms
[1524269764.444903] 64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.164 ms
[1524269765.443820] 64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.115 ms
[1524269766.442804] 64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.089 ms
[1524269767.442649] 64 bytes from 172.17.0.3: icmp_seq=5 ttl=64 time=0.074 ms
[1524269768.442760] 64 bytes from 172.17.0.3: icmp_seq=6 ttl=64 time=0.121 ms
[1524269769.442700] 64 bytes from 172.17.0.3: icmp_seq=7 ttl=64 time=0.125 ms
[1524269770.442648] 64 bytes from 172.17.0.3: icmp_seq=8 ttl=64 time=0.072 ms
[1524269771.442726] 64 bytes from 172.17.0.3: icmp_seq=9 ttl=64 time=0.097 ms
[1524269772.442818] 64 bytes from 172.17.0.3: icmp_seq=10 ttl=64 time=0.151 ms
^C
--- 172.17.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8996ms
rtt min/avg/max/mdev = 0.072/0.150/0.492/0.117 ms
root@44127d827458:/#
```

(b)

Figure 3.7. Pinging of replica server during upgrade.

a) The monolithic replica server. b) The microservice-based replica server.

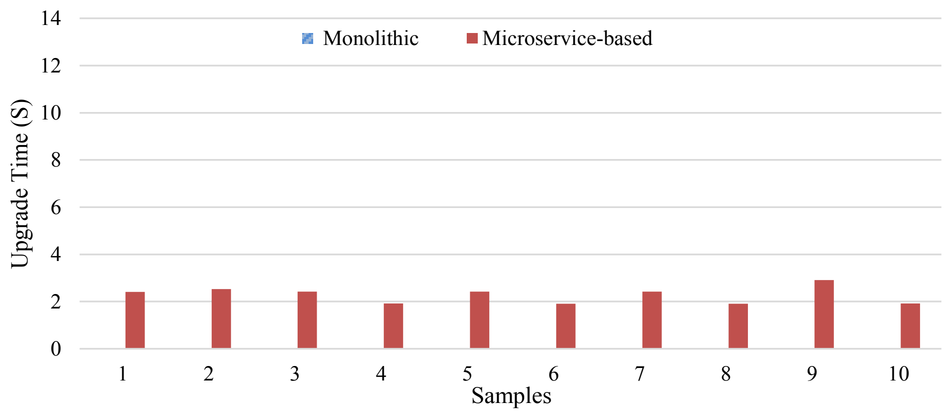


Service Downtime	Monolithic	Microservice-based
Average (S)	5.206	No downtime
Standard Deviation (S)	0.609	-

Figure 3.8. Service downtime during the upgrade of replica server.

average service downtime for the monolithic case is around 5.206 seconds, while the microservice-based replica server notably provides nonstop service during the upgrade period. The main reason is that microservices can be upgraded or configured independently, without affecting or interrupting the functionality and performance of other existing microservices. For example, in our design, the communication microservice will accept all incoming requests in a nonstop manner and might delay some requests until the upgrade procedure is done successfully. This can considerably improve the end-users' perceived Quality of Experience (QoE) compared to the monolithic case. Although there exist some methods that can decrease the service downtime in the monolithic case by temporarily redirecting end-user requests to another replica server until the upgrade process is done, they are subject to request routing costs and complexities [72].

Fig. 3.9 shows the required time for upgrading a generic replica server to support the ABR streaming. As it can be observed in this figure, the microservice case outperforms the monolithic case with a notable difference. The average upgrade time for monolithic replica server is about 10.302 seconds, while in the microservice-based design, this number is reduced by 77% to 2.277 seconds, which highlights the significant gain of



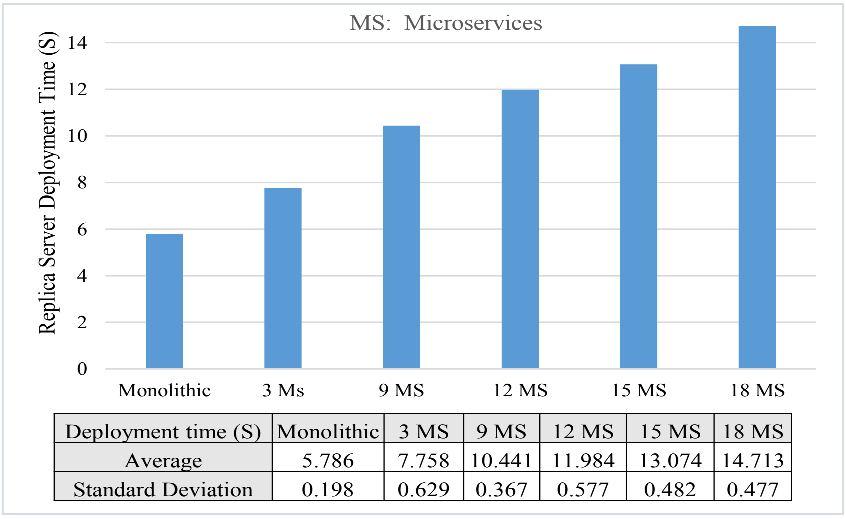
Upgrade Time	Monolithic	Microservice-based
Average (S)	10.302	2.277
Standard Deviation (S)	1.012	0.328

Figure 3.9. Replica server upgrade time.

using microservice-based designs. The main reason is that the upgrade process in the monolithic case involves disposing the existing generic replica server and deploying a new one with ABR streaming functionality, while in the upgrade process of microservice-based replica server, an *ABR generator microservice* is deployed and the interacting microservices are configured to enable the communications with the newly-deployed *ABR generator microservice*. It should be noted that, as shown in Fig. 3.6, in our design of a replica server with ABR functionality, the *content management microservice* is the microservice that interacts with the *ABR generator microservice* to get the video and saves it in the local repository for subsequent requests.

According to the above-mentioned measurements, it can be observed that microservices bring noteworthy benefits to CDNs in terms of shorter upgrading latency and service downtime. However, this comes with a penalty in terms of deployment and content delivery latency.

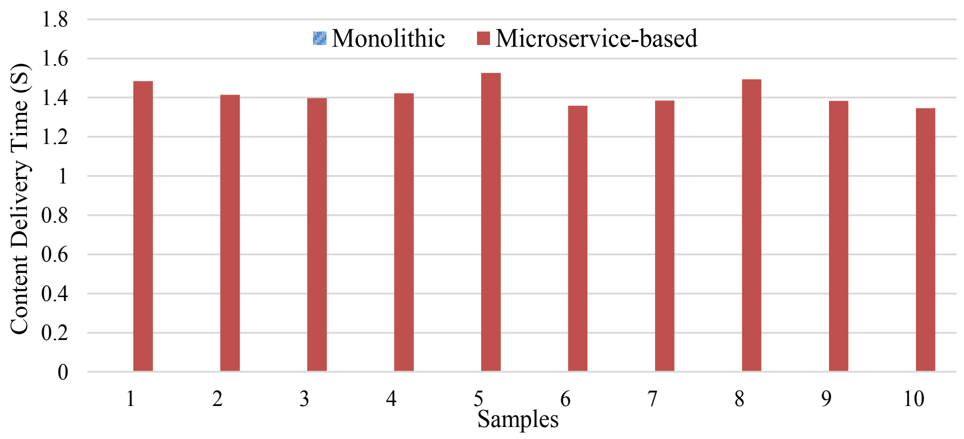
Fig. 3.10 shows the latency for deployment of a generic replica server, including the deployment and post-deployment processes. As it can be observed in Fig. 3.10, the



Deployment time (S)	Monolithic	3 MS	9 MS	12 MS	15 MS	18 MS
Average	5.786	7.758	10.441	11.984	13.074	14.713
Standard Deviation	0.198	0.629	0.367	0.577	0.482	0.477

Figure 3.10. Replica server deployment time.

average latency for deployment of a monolithic generic replica server is about 5.786 seconds, while this latency is slightly increased to 7.758 seconds for deployment of a microservice-based replica server including three microservices. The deployment time in microservice-based designs is always higher than that of the monolithic case, due to the facts that multiple deployments are required. This, in fact, shows the overhead of using microservices. In order to conduct an accurate insight on how deployment latency behaves depending on the number of microservices, several cases are carried out. The case with three microservices implements the generic replica server is depicted in Fig. 3.2. For other cases, we gradually increase the number of instances of each microservice in Fig. 3.2, while considering a load balancer microservice for each group of microservices of the same type to distribute the load among them. Therefore, the case with 9 microservices, for instance, includes 2 instances of each microservice (i.e. communication, content delivery, and content management microservices) and 3 load balancers. It can be observed in Fig. 3.10 that by increasing the number of microservices, the deployment latency behaves at a slight constant rate, linearly. When



Content Delivery Time	Monolithic	Microservice-based
Average (S)	1.081	1.421
Standard Deviation (S)	0.038	0.057

Figure 3.11. Content delivery time.

the number of microservices is increased, the deployment latency slightly increases as well, due to the additional microservice reconfigurations that are needed.

Fig. 3.11 shows the content delivery latency when an end-user sends a request for a 1 MB video to the newly deployed replica server. The average content delivery latency in the microservice case is about 1.421 seconds while this is slightly reduced to 1.081 seconds in the monolithic case. This shows the overhead of microservice designs in content delivery, which is expected due to the inter-microservice communications that are excluded in the monolithic case.

Note that some fluctuations can be observed between the samples in all measurements, which is mainly because the Internet is used for enabling the interactions between modules.

3.4. Conclusion

This chapter proposed an architecture for the on-the-fly provisioning of replica servers as CDN components using NFV and microservice architecture principles. The replica servers are designed as a set of microservices and they are implemented, packaged and deployed using NFV technology. Then, the required process is proposed to integrate them into the existing CDN and fill them with the popular contents. A prototype is implemented and measurements are taken to evaluate the effectiveness of the method. An observation is that the microservice-based design notably outperforms the monolithic case in the upgrade process. However, this comes with a penalty in terms of deployment and content delivery latencies. Another observation is that the deployment latency increases linearly as we increase the number of microservices that make the replica server.

Chapter 4

4. An NFV and SDN-based Architecture for VAS Provisioning in CDNs

4.1. Introduction

As mentioned earlier, provisioning the VASs in an efficient manner requires a redesign of the traditional architecture of the CDNs. Based on the Internet Engineering Task Force (IETF) categorization [9], the building blocks of video VASs are application-level middle-boxes. Transcoder, mixer, and compressor are among the examples. Traditionally, these middle-boxes are provisioned as physical building blocks [10], at fixed network locations and on a proprietary/dedicated hardware, like most of the middle-boxes nowadays. The shortcomings of this traditional mode of middle-box provisioning are widely known. It is subject to lack of dynamicity, automation, and flexibility when deploying and managing the services. This is because those tasks require complex and labor-intensive manual efforts [10][73][48]. Furthermore, the configuration, deployment, and orchestration of such middle-boxes are inflexible and time-consuming [10]. Unlike predicted events (e.g. sports events), in unpredicted events when many videos go viral (e.g. Terrorist attacks of September 11, 2001 [7]), CDNs are challenged to react fast to provision such VASs in various locations, in a timely manner, and with sufficient flexibility.

In order to tackle those challenges when provisioning VASs, CDN providers recently leverage technologies such as NFV and SDN to re-architect the traditional CDN components and middle-boxes. However, the existing literature in this overall context, including CDNs (e.g. [48], [43], and [47]), mainly focuses on using NFV and SDN technologies for provisioning the services that use IP-level middle-boxes such as NAT, firewall, and load balancers. In fact, they overlook the required features to support the orchestration of application-level middle-boxes, which should be considered in the CDN context.

This chapter aims at proposing and validating an architecture for enabling VAS provisioning in CDNs, using NFV and SDN technologies. The second objective is to reduce the significant overhead in the end-user's perceived latency while using application-level middle-boxes, as much as possible. The third objective is to show the feasibility of this method by prototyping the architecture and performing additional validation measurements. The experimental measurements lead to some practical insights concerning the overhead reduction and the significant impact of re-ordering VNFs in a chain (where applicable) on the end-user's perceived latency.

4.2. The proposed Architecture

This section introduces the architectural principles that our work is based on. Then, it discusses the high-level architecture of the designed system, including its architectural planes, modules, and interfaces. Afterward, illustrative scenarios are provided.

4.2.1. Architectural Principles

The first architectural principle is the implementation of middle-box as VNFs and the use of SDN technology to dynamically provision the paths once VNFs are deployed. In this chapter, it is assumed that the middle-boxes are already implemented as VNFs and deployed in the network. The focus is therefore on how to dynamically provision the paths, i.e. to dynamically compose the VNFs.

The second architectural principle is the establishment of a signaling path between the end-user and the replica server that holds the raw video. This path includes the VNFs that add more and more value to the video as it goes from the replica server to the end-user.

As the third principle, the videos are cached in the replica servers as they go through the VNFs. Here, each VNF is assumed to be co-located with a replica server. In this case, the video is cached in the replica server after being processed by the associated VNF. The fundamental reason behind this principle is performance. This means that when an end-user requests a value-added video that was already delivered for another requester, the end-user is directly served from a cache. The same raw video does not have to go through the same VNFs again. Even better, a video that needs to go through a set of VNFs never goes through a subset of VNFs it has already traversed.

The fourth principle is that the REST architectural style is used for designing the interactions between the application-level SDN switches and the SDN controller [74].

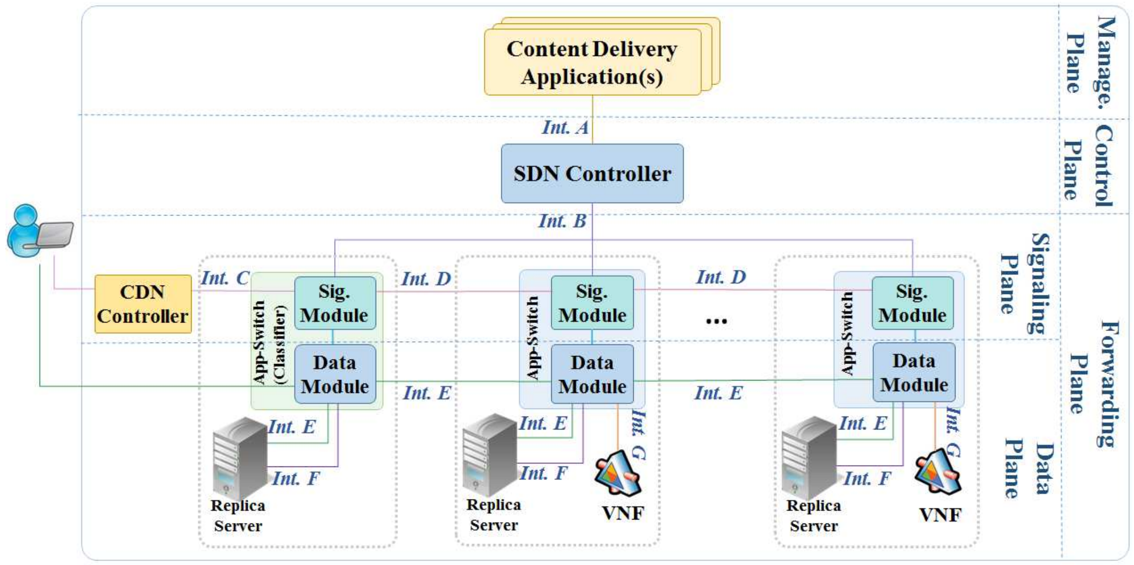


Figure 4.1. The proposed high-level architecture

The interactions with these entities are based on a standardized communication protocol (e.g. HTTP). REST is selected because it is standard-based, lightweight and flexible for data representation, allowing us to describe APIs in a generic and abstract way. Following this principle, the application-level SDN switches expose REST API to the SDN controller in order to install and/or modify the routing rules in the SDN switches.

4.2.2. High-Level Architecture

Fig. 4.1 shows a high-level view of the designed architecture. The proposed architecture is layered over a set of planes. These planes and the contained modules are discussed here. We also discuss the interfaces between the modules in this section.

A) Architectural planes

Similar to the traditional SDN-driven architectures [32][75][76], the proposed architecture consists of management, control, and forwarding planes. The management plane handles the value-added video delivery policies. These policies are embedded in a set of value-added video delivery applications that specify how the middle-boxes, implemented as VNFs, are composed for specific value-added video services. The control plane enforces the policies of the management plane by programming the application-level SDN switches of the forwarding plane. The control plane includes an SDN controller that interacts with the architectural modules of the forwarding plane. The reader should note that our forwarding plane, unlike the traditional SDN forwarding planes, has a signaling sub-plane beside the regular data plane. This is due to our second architectural principle that ensures the establishment of a signaling path between the end-user and the replica server that holds the raw videos. The forwarding plane consists of the following architectural modules: CDN controller, application-level switches, replica server, and VNFs.

B) Architectural modules

In what follows, we focus on the modules of the forwarding plane, i.e. the innovative modules in this architecture. The main module is the application-level switch. Unlike the stateless traditional SDN switches, this switch is stateful. It has two modules: The signaling module, related to the signaling sub-plane, and the data module, related to the data sub-plane. The signaling module is programmable by the SDN controller. It establishes the path between VNFs, through which the raw video goes for a value-adding purpose. The data module is responsible for redirecting the raw video through the appropriate VNFs to get the required value-added services. The data module is also responsible for caching the video in the associated replica server. Regarding the application-level switches, the first application-level switch in the flow is a particular one. It is the flow classifier. It extracts the end-user preferences (e.g. content quality, formats) and tags the request with a chain-ID. The chain-ID serves to identify the packets belonging to a given chain of VNFs. Like other application-level switches, the flow classifier might be connected to a replica server.

Table 4.1. Examples of the API operations exposed by the application-level switches.

REST Resource	Operation	HTTP Action and Resource URI	Request Body Parameters	Most Important Info in the Response
Routing Rule	Add new Routing Rule	POST: /RoutingRule	Routing rule: match criteria, action	ID and URI of the created routing rule
	Retrieve a specific Routing Rule	GET:/RoutingRule/{ID}	None	Routing rule properties
	Retrieve all Routing Rules	GET:/RoutingRule/all	None	List of all routing rules
	Remove a Routing Rule	DELETE: /RoutingRules/{ID}	None	Success/failure code
	Update a Routing Rule	PUT:/RoutingRules/{ID}	Routing rule: match criteria, action	Success or failure indication

Other modules are the *CDN controller*, the *replica server,* and *VNFs*. Unlike regular CDN controllers that redirect the request directly to the optimal *replica server*, our *CDN controller* is designed to redirect the end-user requests to the *flow classifier*. This is due to our second architectural principle. The *replica server* here, similar to the traditional ones, is part of the CDN network and holds a copy of the raw content. However, the replica servers associated with application-level switches also cache a copy of the value-added content to serve the next requests right away. In fact, any content is cached after being customized by each VNF. This brings the advantage that any content that traverses through VNFs is cached in each customization step rather than going through the VNFs it has already traversed. Each VNF is linked to an application-level switch, following a linear topology model as shown in Fig. 4.1.

C) Interfaces

In what follows, the main interfaces of the proposed architecture are described. Int. A is the northbound interface of the SDN controller. It is a high-level interface. Existing northbound APIs of SDN controllers could be reused to implement this interface. We extend Int. B, i.e. the southbound API that is a lower-level abstraction interface. It defines the communication protocol between the SDN controller and the SDN switches. It also defines the instruction set, used by the SDN switches. As stated earlier, our architecture proposes the use of application-level switches as the forwarding elements in the content delivery process, instead of conventional IP-level switches. Consequently, this requires some enhancements in the southbound API. These enhancements include adding application-level fields to the matching fields of the flow rules. The examples of the standard fields are application-level addresses and the examples of the extended fields are content quality information in HTTP. These extensions are indispensable for routing the messages between the application-level switches and for manipulating the application-level messages to add value to the videos.

Based on our fourth architectural principle, *Int. B* is REST-based. The *SDN switches* expose Restful APIs, enabling them to receive the application-level rules from the *SDN Controller*. For this API, the application-level rules are designed as a RESTful resource. Table 4.1 shows four API examples including resources, supported operations, required request parameters and the response content of each operation.

Other interfaces in the architecture are designed using a request-response model in a client-server interaction fashion. *Int. C* is the interface between the *CDN controller* and the *flow classifier*. It redirects end-user requests to the *flow classifier*. *Int. D* is the signaling interface between the *application-level switches*. It passes the request for a fully/partially customized content. *Int. E* is the data interface and pushes the video to the *application-level switches/replica servers*. *Int. F* is the data interface between the *application-level switch* and the *replica server*, to retrieve the video. *Int. G* is the data interface between *VNF* and the *application-level switch*. It can push or retrieve the raw and value-added video.

4.2.3. Illustrative Sequence Diagram

The envisioned scenario includes the ad-insertion VAS discussed in subsection 2.1.2. We assume that the content provider offers ad-free video clips to the registered end-users who watch videos for monthly fees. Meanwhile, it inserts an advertisement on top of the raw videos for guest log-ins. End-users might select the lower quality video for a lower price or for less bandwidth usage. To that end, two VNFs are needed: (i) a mixer to insert an advertisement and (ii) a compressor to degrade the video quality. Therefore, three *application-level switches* and one *flow classifier* are needed.

Table 4.2 shows four prospective chains that could be defined based on end-user preferences. Here, we consider the fourth case, where the 'guest' end-user logs in and requests the low-quality video. The raw video, initially stored in *replica server 3*, should then pass through the mixer and the compressor *VNFs*. To make this happen, a set of policies/chains is defined by the value-added video delivery applications. Afterward, a set of application-level flow rules is injected into the *application-level switches* by the

Table 4.2. Prospective chains according to the end-user's preferences.

	End-user preferences		Final Video	VNFs Chains
	Membership	Quality		
1	Registered	High	Raw Video	No VNF
2	Registered	Low	Compressed Video	Compressor
3	Guest	High	Mixed Video	Mixer
4	Guest	Low	Mixed & Compressed Video	Mixer, Compressor

Table 4.3. Application-level flow rules.

Rule	App-Switch ID	Application-level Flow Rule	
		Matching field	Action
1	SW0	Membership=Guest & Quality=Low	Push tag Chain-ID:X
2	SW0	Chain-ID=X	Output to SW1
3	SW1	Chain-ID=X	Output to SW2
4	SW2	Chain-ID=X	Output to SW3

SDN controller. An abstracted form of these rules is listed in Table 4.3 given that *SW0* is the *flow classifier*.

Fig. 4.2 illustrates a sequence diagram of the interaction of the architectural components to enable this scenario. Firstly, a guest end-user sends a request for a low-quality video (Fig. 4.2, action 1). The *CDN controller* redirects the received request to the *flow classifier* (action 2). The *SM0* performs the flow classification. It receives the request packet and extracts application-level header values such as the end-user membership status (e.g. guest) and the required video quality (e.g. low). Then, it pushes the chain-ID on the request (action 3) based on a matching rule (Table 4.3, Rule 1) in its Routing Table (RT). The chain-ID indicates that the mixed and compressed video is being requested. Before sending a request to other *application-level switches*, *SM0* checks the availability of the fully customized requested video (i.e. mixed and compressed video) with its *data module, DM0* (action 4). Similarly, *DM0* asks the associated *replica server*, *RS0*, for the fully customized video availability (action 5). If available, *RS0* sends the video to *DM0* (action 6), to be delivered to the end-user (action 7). If the video is not available in the *replica server*, a false message is sent back to *DM0* (action 8) and then from *DM0* to *SM0* (action 9). Once *SM0* receives a false message, indicating that the fully customized video is not being cached in the associated *replica server*, it selects *SW1* as the next hop (action 10), using the chain-ID (Table 4.3, Rule

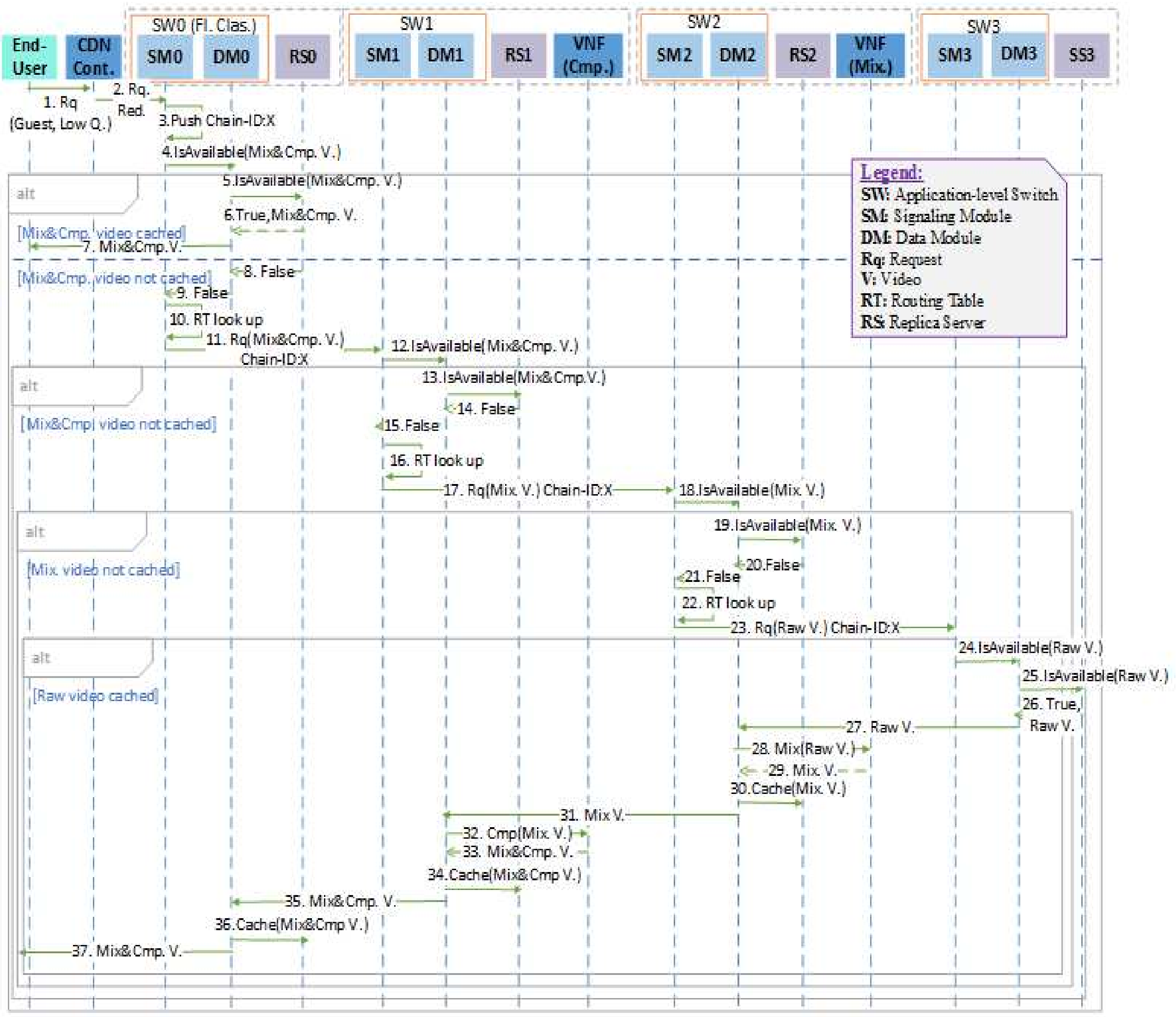


Figure 4.2. Sequence diagram of value-added video delivery.

2). After that, it initiates a request for the fully customized video (action 11).

Once *SM1* receives the request, with chain-ID: X, it checks the availability of the requested video with *DM1* (action 12). *DM1* also asks the associated *replica server*, *RS1*, for video availability (action 13). Here, we assume the customized video is not available in the *RS1 cache*; so, a false message is sent back to *DM1* (action 14) and then from *DM1* to *SM1* (action 15).

Upon receiving this message, *SM1* selects the next hop (action 16), i.e. *SW2* (Table 4.3, Rule 3) and initiates a request to it (action 17). *SM1* requests the partially customized video, i.e. mixed video. This is because it is connected to a compressor *VNF,* and upon receiving the partially customized video, it redirects it to the compressor *VNF* to produce the fully customized video. *SM2* receives the request for the partially customized video (i.e. mixed video) with chain-ID: X. Similar to other *signaling modules*, *SM2* asks the *data module* for video availability (action 18). Afterward, *DM2* verifies the availability of the mixed video in *RS2* (action 19). Assuming that the partially customized video is not available in *RS2*, a false message is sent back to *DM2* (action 20) and accordingly to *SM2* (action 21). *SM2* finds *SW3* (Table 4.3, Rule 4) as the next hop (action 22). It sends a request for the raw video to *SW0* (action 23). This is because it has a mixer *VNF* associated to it and produces the requested mixed video upon receiving the raw video. *SM3* receives this request for the raw video and delegates the video retrieval to *DM3* (action 24). *DM2* initiates a request for the raw video to *RS3* (action 25).

Assuming that the raw video is available on replica server *RS3, RS3* delivers it to *DM3* (action 26). Following the signaling path saved by the stateful signaling module, *DM3* pushes the mixed video to *DM2* (action 27). Then, *DM2* pushes it to the mixer for ad-insertion and receives the mixed video back (actions 28, 29). It should be noted that the advertisement video is assumed to be co-located with the mixed *VNF*. *DM2* pushes the mixed video to *RS2* to be cached there as a partially customized video for subsequent similar requests (action 30). Likewise, it pushes the mixed video to *DM1* (action 31), following the signaling path. *DM1* pushes the mixed video to the compressor and receives the fully customized video (actions 32, 33). Similarly, *DM1* sends the fully customized video to *RS1* for caching (action 34) and to *DM0* (action 35) to serve the *end-user* with the requested mixed and compressed video (action 37) after being cached in *RS0* (action 36).

4.3. Implementation and Validation

In this Section the developed prototype is discussed first that implements the proposed architecture. Second, the measurements performed on the prototype are presented followed by related discussions.

The implementation covers the illustrative scenario discussed in the previous subsection. However, to increase the possibility of chains, we add a transcoder as the third VNF to the chain. Assuming the original video is saved in MP4 format, the end-user might request the video in AVI format. Therefore, the third VNF (i.e. a transcoder) changes the video format from MP4 to AVI.

4.3.1. Prototype Architecture

The prototype is implemented based on the architecture in Fig. 4.3. All the developed modules are deployed on Virtual Machines (VMs) and those VMs are provisioned by OpenStack as VIM. The NFVI is provided by either the Smart Applications on Virtual Infrastructure (SAVI) testbed [68] or the Open Platform for NFV (OPNFV) [77] test lab since both are used for validation. SAVI is a Canadian distributed testbed for future Internet applications and OPNFV is a new open source project for accelerating the NFV technology evolution.

We have used HTTP as the basis of our implementation since it is pervasive in content delivery settings. The *application-level switches* are extended HTTP proxies. Specifically, they are implemented as Java servlets and are packaged as Java Web ARchives (WAR). They also expose a REST API to the *SDN controller* for the application-level rule handling.

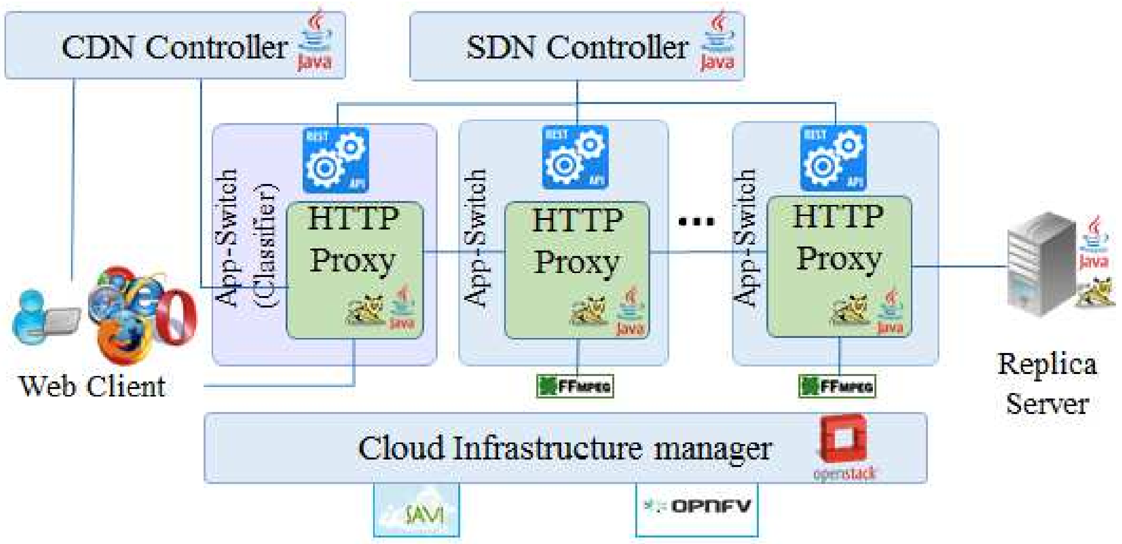


Figure 4.3. The developed prototype architecture

For the *VNF* implementation, the FFmpeg is used – an open source software [78] providing libraries and tools to handle multimedia operations. Upon installation, it is configured to provide mixing, transcoding and compressing features.

The *replica servers* are implemented as Java servlets. They are co-located with *application-level switches* on the same hosting VMs. The *replica servers* cache the content on Linux file system directories. The *CDN controller* is also implemented as Java servlet. It should also be noted that the interfaces in the *forwarding* and *data planes* are implemented based on HTTP rendering – i.e. *Int. C* and *Int. D* as HTTP requests and *Int. E* as HTTP responses.

4.3.2. Performance Evaluations

This section analyzes the experiments done to get some insights about the application-level chaining. This research presents five sets of experiments to (i) prove the feasibility of our approach when NFV and SDN technologies are used for implementing the middle-boxes, and (ii) evaluate the gains when the caching features are integrated into the replica servers and consequently prove the reduction of the content customization overhead.

The following environment has been set up for the experiments: A medium-size OpenStack flavor is allocated to each *application-level switch*, *VNF*, *CDN controller* and the *replica server*, with 4 GB of RAM, 2 vCPUs and 40 GB of Disk. For all scenarios, the *application-level switches*, *CDN controller*, *replica servers* and the *end-user* are deployed on the Toronto site. The communication between all architecture modules is enabled via the Wide Area Network (WAN) – the Internet. In all scenarios, the measurements are taken for 10 times. The average and the standard deviation are also taken into account. The standard deviation is calculated by taking the square root of the variance while the variance is the average of squared differences from the mean (i.e. the average of measured values).

The first set of experiments measures value-added video delivery when the customized video is not cached i.e. the raw video needs to pass through the mixer,

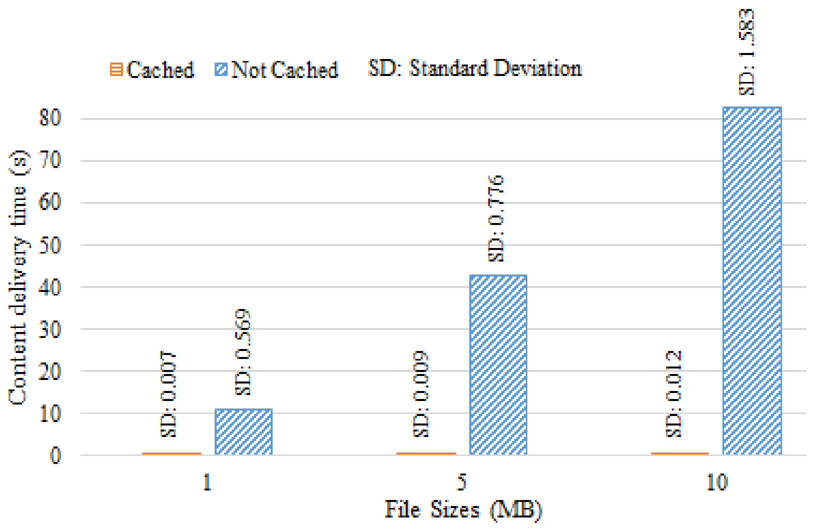
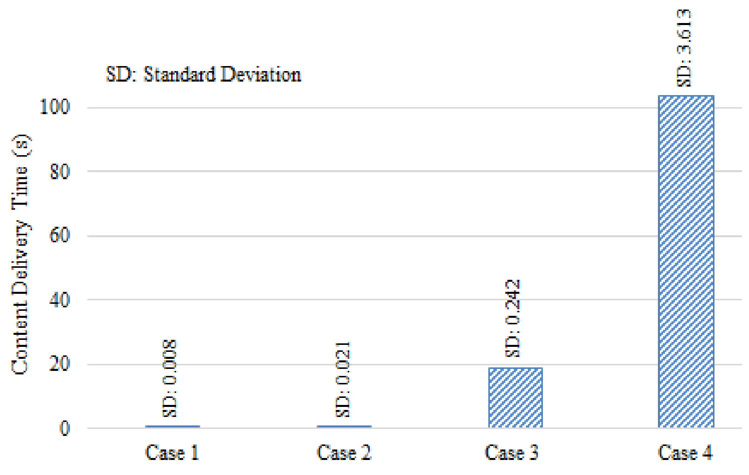


Figure 4.4. Content delivery time for various file sizes when video is cached vs. when the video is not cached, for VNFs are located in Toronto.

transcoder and compressor VNFs to gain value, vis-à-vis when the value-added/customized video is cached on the replica server and is delivered to the end-users directly. In the second case, the video does not need to traverse through VNFs. Fig. 4.4 shows the value-added video delivery latency for a raw video with sizes of 1, 5 and 10 MB in two cases. The first case is about passing the raw video through the required *VNFs* for getting customized. The second case is when the fully customized video is cached in the *replica server* to avoid going through *VNFs*. For example, for a 5-MB raw video file, the latency in the first case is 42.7198 seconds and is reduced to 0.0566 seconds in the second case. This measurement shows that caching the fully customized video has a great impact on the content delivery time.

The second set of experiments measures and compares the video delivery latency when the raw or partially customized video is cached in various locations in the network. Fig. 4.5 shows the video delivery time for four different cases. This set of experiments



Case 1: Fully customized (mixed&compressed) video cached at replica server associated to flow classifier

Case 2: Fully customized (mixed&compressed) video cached at replica server associated to the second application-level switch

Case 3: Partially customized (mixed) video cached at replica server associated to the third application-level switch

Case 4: No customized video is cached at replica server and raw video passes through mixer and compressor

Figure 4.5. Content delivery time with caching partially/fully customized video

evaluates the caching of the partially and fully customized video and studies its impact on video delivery delay. The delivery latency for a 10 MB video file is measured as such when VNFs are located in Vancouver. Case 1 is when the fully customized video (i.e. mixed and compressed video) is cached in the *replica server* associated with the first application-level switch (i.e. replica server 0 associated with the *flow classifier,* Fig. 4.2). In this case, the signaling process stops at this replica server and the fully customized video is sent back to the end-user. Case 2 is when the fully customized video is cached in the replica server associated with the second application-level switch (i.e. replica server 1 associated with the application-level switch 1, Fig. 4.2). In this case, the signaling process stops at this replica server and the fully customized video is sent back to the end-user through the signaling path.

In case 3, the fully customized video is not cached in the replica servers and only the partially customized video (i.e. mixed video) is cached in the second replica server (Fig. 4.2). In this case, the signaling process stops in this replica server and the partially customized video should pass through the signaling path. It goes through the compressor VNF to change to the fully customized video. After being cached in the replica servers, the fully customized video is served to end-users. Case 4 shows the case when no fully or partially customized video is cached in the network. In this case, the raw video should pass through all VNFs; it is then saved in the replica servers and finally served to the end-user. As expected, the delivery time increases when we move from case 1 (75 milliseconds) to case 4 (103.337 seconds) due to the transfer latency and the video transformation latency. This, in fact, shows the importance of in-network caching the partially customized video in addition to the fully customized ones. Moreover, a comparison of case 3 (18.921 seconds) and case 4 (103.337 seconds) brings in an interesting observation, that is the high mixing latency compared to the compressing latency.

Table 4.4. Video delivery latency for various order of VNFs in VNF-FGs.

Chains	VNFs			Latency (s)	Standard Deviation (s)
	First	Second	Third		
1	Mixer	Transcoder	Compressor	91.4916	2.5787
2	Mixer	Compressor	Transcoder	86.3219	2.1604
3	Transcoder	Mixer	Compressor	36.5288	0.9512
4	Compressor	Mixer	Transcoder	31.8374	0.4225
5	Transcoder	Compressor	Mixer	30.4018	0.4224
6	Compressor	Transcoder	Mixer	24.8588	0.9307

The third set of experiments shows the impact of reordering VNFs in the chain on the video delivery delay. Table 4.4 shows the video delivery latency measured for various orders of VNFs in a given chain. This set of measurements is taken for a 10 MB file when VNFs are located in Toronto. To increase the possibility of chains, we add a transcoder as the third VNF to the chain. Assuming the original video is saved in MP4 format, the end-user might request the video in AVI format. Therefore, the third VNF (i.e. a transcoder) changes the video format from MP4 to AVI. It should be noted that, in our video delivery case, any possible order of VNFs in the chain leads to relevant, subsequent results.

As table 4.4 shows, with three VNFs, six different chains can be defined. The measurements show that, as we move the mixing VNF to the end of the chain, the video delivery latency is reduced. For example, for a 10-MB raw video file, the latency in the first chain is 91.491 seconds and is reduced to 24.858 seconds in the sixth case. This is because video compression and transcoding (from MP4 to AVI format) reduce the file size. The smaller the video file, the less time it takes it to be mixed with an advertisement. If the compressing is done as the first VNF in the chain, video latency is less compared to the case when transcoding is done at the beginning. This is because

Table 4.5. Signaling and content delivery latency.

Chains	VNFs			Signaling Latency (S)		Video Delivery Latency (S)	
	First	Second	Third	Average	Standard Deviation	Average	Standard Deviation
1	Mixer	-	-	0.1774	0.0132	58.0193	2.8050
2	Mixer	Compressor	-	0.1922	0.0220	76.628	2.3135
3	Mixer	Compressor	Transcoder	0.2259	0.0386	83.0235	2.6049

the file size reduction by compressing is larger, compared to that by transcoding. For example, in chain 5 where transcoding is done first, the video delivery latency is 30.4018 seconds. This number is reduced to 24.8588 seconds for chain 6, where compressing is done before transcoding.

The fourth set of experiments measures the signaling time for various chains, including the various number of VNFs. Table 4.5 details the signaling and video delivery latency measurements. Those values correspond to the chains including the various number of VNFs located in Toronto. The size of the considered raw video is 10 MB. The video delivery latency is measured as the previous sets of measurements were. It includes the signaling, video transfer and customization times. The signaling latency is measured from the time the end-user initiates the request for a video to the time the request is received by the replica server holding the raw video. Network Time Protocol (NTP) is used for the time synchronization between the end-user machine and the replica server machine. As the results show, the signaling latency is negligible compared to the video delivery latency. This highlights the gain of application-level dynamic chaining. As expected, the signaling latency increases as the number of VNFs in the chain increases. For example, for a chain including a single mixer, the signaling latency is 0.177 seconds and it increases to 0.2259 when a compressor and a transcoder are added to the chain. However, this increment is trivial, as in this prototype, all the application-level SDN switches are deployed on the same SAVI edge on the Toronto site.

Finally, the fifth experiment consists of changing the locations of the VNFs between the two SAVI sites, one in Vancouver and the other in Toronto, and measuring the corresponding value-added video delivery delay, for an end-user located in Toronto. This measurement shows that the placement of VNFs has an impact on the delivery time. Fig. 4.6 shows the obtained video delivery latency values measured for a raw video with the sizes of 1, 5 and 10 MB. The measurements are made when the raw video goes through a mixer first, then a compressor and finally a transcoder VNF. For example, for the 5 MB raw video file, the latency is 58.2358 seconds when VNFs are located in Vancouver and it is reduced to 45.6654 seconds when they are located in Toronto. The results show the significant impact of the physical location of VNFs on the end-user's perceived latency. This indicates the need for algorithms to optimally place these VNFs based on a well-defined set of placement criteria, such as QoS and the network

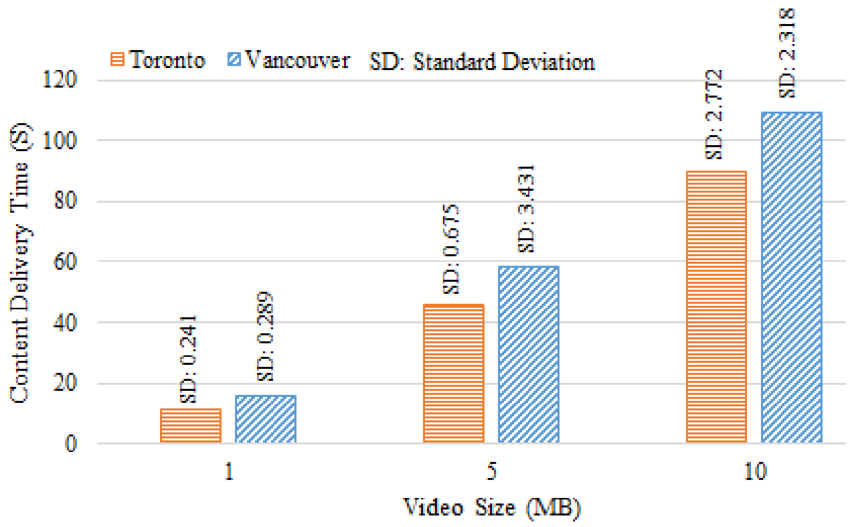


Figure 4.6. Video delivery delay for VNFs deployed in Toronto vs. Vancouver for end-users in Toronto

overhead. The placement of middle-boxes is a critical requirement in distributed environments such as CDNs, to satisfy the required QoS at a minimum cost.

4.4. Conclusion

This chapter proposes an NFV- and SDN- based approach that enables the provisioning of value-added services in CDNs. Those services dynamically generate and deliver individually tailored value-added contents to end-users. Our proposed architecture consists of extensions to the existing SDN switches and controllers to support the dynamic chaining of application-level middle-boxes. The application-level middle-boxes that provide value-added video services are provisioned as VNFs and are chained on the fly, using the application-level SDN controller and switches. SDN controller programs the SDN switches and the SDN switches steer the traffic through the ordered set of pre-deployed VNFs, for meeting the content providers' enforced policies and the end-user requirements. Besides, this approach reduces the significant delay of using application-level middle-boxes, by providing additional features for caching partially and fully customized videos for value-added services. The measurements show the feasibility of the approach and highlight the fact that the caching features reduce the content delivery time significantly. Another observation is that the location of VNFs has a great impact on the content delivery time.

Chapter 5

5. Dynamic VNF Placement for Value Added Services in CDNs

5.1. Introduction

In the NFV environment, the VNFs can be easily deployed in the network, however, the optimal number of VNFs and their locations impact the CDN provider expenses and end-users' perceived QoS. Although VNF placement has attracted research interests recently, very few proposals focus on VNF placement and chaining for CDN VASs. CDN VASs have a specific characteristic which is the fact that they have an unknown endpoint prior to VNF placement. This end-point corresponds to the replica server that is selected to serve the content to the end-user.

The approaches proposed to date to tackle this problem have focused on static placement. However, such mode of placement is not adequate for dynamic systems such as CDNs, in which the end-user requests for VAS arrive at the CDN dynamically and are difficult to forecast, mainly because they may vary in time and space. In real life, there could be a variety of motivations for modifying the topology of the VNFs in the network. Examples of such motivations include the introduction of new VASs and changes in the end-user pattern. Such situations call for an efficient method that dynamically places VNFs for CDN VASs.

In this chapter, we propose an approach called DVPVC for the dynamic placement and chaining of VNFs for CDN VASs. DVPVC considers an unknown end-point for VAS VNF-FGs and advocates a VNF topology reconfiguration approach to embed both existing and newly-arrived VNF-FG requests such that the total reconfiguration cost including VNF migration, instantiation, hosting, and routing costs are minimized. Our proposed method ensures that the QoS of all end-users is satisfied in terms of content delivery time. Moreover, unlike the existing dynamic VNF placement proposals, our method supports the traffic variations in VAS VNF-FGs that might occur as a result of video customization. Our measurements show that DVPVC outperforms the greedy first fit approach and obtains solutions very close to the optimal solution in several cases.

5.2. System Model

In this section, we first give an overview of the problem and then explain our system model for VAS VNF placement and chaining. We assume that a number of ad-insertion VASs have been deployed in the network, i.e., the associated VNFs like mixer, transcoder, and compressor VNFs are already deployed and chained in the network. There are multiple VNF-FG requests for each VAS, each reflecting a flow of content from the content server through some ordered VNFs and finally to the end-user (see Fig.

2.2). In a dynamic manner, new end-users subscribe for new ad-insertion VASs. The problem is to find a placement for all the existing and newly-arrived requests with minimum reconfiguration cost while respecting network bandwidth and VNF processing constraints and meeting the end-users QoS in terms of delay. To address this aim, the placement strategy may reuse the already-deployed VNFs, migrate them or deploy new VNF instances. The network, the VNFs and the VNF-FG requests are modeled as specified below.

Network- We consider *N* as a set of network nodes, *S* as a set of replica servers, and *U* as a set of end-users, where $N = S \cup U$. Given a network with *n* nodes, we represent the network bandwidth with a matrix $B_{n\times n}$, where $b_{i,j}$ determines the bandwidth of the link between nodes i and j. Similarly, matrices $C_{n\times n}$ and $D_{n\times n}$ are defined such that the entries $c_{i,j}$ and $d_{i,j}$ determine the transmission cost and delay of the links between the nodes i and j, respectively, for the unit of traffic transmission.

Replica Servers- for each replica server $s \in S$, we represent the cost of resources per vCPU with ρ_s. The processing capacity of each replica server is delineated with G_s.

VNFs- Let K denote the set of all VNF types defined in the system, such as mixer, transcoder, and compressor. Each VNF type $k \in K$ has a predefined license cost L_k, processing capacity P_k, and hosting resource requirement R_k. The set of available instances for VNF type k is delineated as I_k. For each VNF type k, the CDN provider can deploy a maximum number of instances i.e. $|I_k| \le Max_{instances}$, however less VNF instantiation leads to lower deployment costs. The processing delay of VNF type *k* on a replica server *s* is delineated as M_k^s.

VNF-FG requests- We represent the set of VNF-FG requests including the existing and the new ones with F. Each VNF-FG request is indicated as *f* and is modeled as a

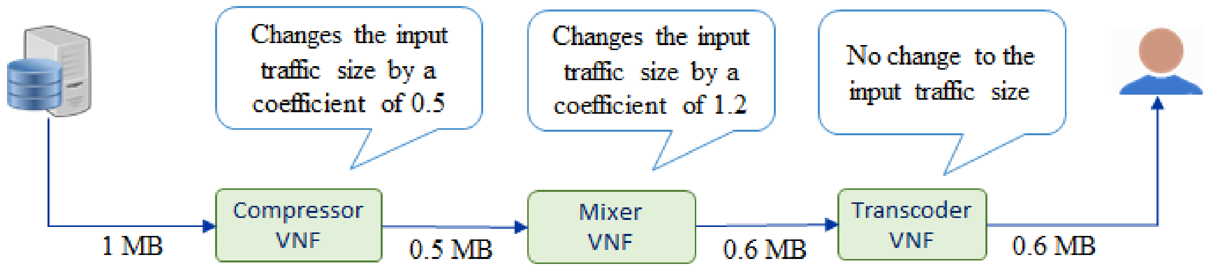

Changes the input traffic size by a coefficient of 0.5
Changes the input traffic size by a coefficient of 1.2
No change to the input traffic size
Compressor VNF
Mixer VNF
Transcoder VNF
1 MB
0.5 MB
0.6 MB
0.6 MB

Figure 5.1. Traffic bitrate variation.

VNF-FG. The maximum delay tolerated by a VNF-FG request *f* is denoted as X_f. The node indicating the end-user of VNF-FG request *f*, the set of required VNF types for VNF-FG request *f*, the required chain type for *f*, the first and last VNFs in VNF-FG are represented as u_f, V^f, c^f, fst_f, and lst_f, respectively.

Let T_f be the initial traffic of a VNF-FG request. The VNF-FG request traffic load may vary as it traverses through the VNFs. For a VAS application, a video compressor VNF reduces the input video size. However, a video mixer might increase the video size on the output link, as it mixes the input video with another overlay video. Traffic variation as a result of other VNFs' operations such as Firewall and Deep Packet Inspector (DPI) have also been reported [79][77][78]. To model such variation we define q_k as the coefficient of traffic variation as the result of VNF of type k. When the coefficient is more than one the traffic is increased; values less than one indicate traffic shrinking; a coefficient of 1 means no changes on the traffic load. Fig. 5.1 shows the associated coefficients of 0.5, 1.2, and 1 for a compressor, mixer and a transcoder, respectively. Table 5.1 summarizes the ILP parameters and variables.

Table 5.1. Input parameters and variables.

Input Parameters	
Network	
S	Set of servers
U	Set of end-users
N	Set of network nodes, $N = S \cup U$
$B_{i,j}$	The bandwidth between nodes i and j, $i, j \in N$
$C_{i,j}$	The transmission cost between nodes i and j, $i, j \in N$
$D_{i,j}$	The delay between nodes i and j, $i, j \in N$
μ	Maximum node/link/VNF usage threshold
Servers	
ρ_s	Replica server's cost per unit, $s \in S$
G_s	Replica server's capacity (in processing resource units), $s \in S$
VNFs	
K	Set of VNF types
L_k	License cost for VNF type *k*, $k \in K$
P_k	The processing capacity of VNF type *k*,(in traffic units), $k \in K$
R_k	The resource requirements of VNF type *k,* (in processing units), $k \in K$
I_k	Set of VNF instances associated to VNF type *k*, $k \in K$
$\varphi_k^{s,t}$	Cost of migrating VNF type *k* from server *s* to server t,$k \in K$, $s, t \in S$
M_k^s	The processing delay of VNF type *k* on server *s,* $k \in K,\ s \in S$
VNF-FG requests	
F	Set of VNF-FG requests
u_t^f	1, if node t is the end-user of VNF-FG request *f,* $f \in F$
u_f	The node indicating the end-user of VNF-FG request *f,* $u_f \in U, f \in F$
V^f	Set of required VNF types for VNF-FG request *f,* $V^f \subset K$
C^f	The required chain type for VNF-FG request *f*
fst	The first VNF in VNF-FG request *f* , $fst_f \in V^f$
lst_f	The last VNF in service chain of VNF-FG request *f* , $lst_f \in V^f$
T_f	Traffic units of VNF-FG request *f* , $f \in F$
X_f	Maximum delay tolerated by VNF-FG request *f* as per SLAs.
c_s^f	1, if replica server ***s*** can be content server for VNF-FG request *f.*
$w_{k,l}^f$	The bitrate change coefficient for traffic of VNF-FG request *f* between VNFs *k* and *l* *k, l* $\in V^f$
Decision Variables	
γ_s^f	Binary variable, indicating if server *s* is selected to serve content for VNF-FG request *f*
$\tau_{k,i}^s$	Binary variable, indicating if instance *i* of VNF type *k* is instantiated on server *s*
$\lambda_{s,k,}^f$	Binary variable, indicating if instance *i* of VNF type *k*, instantiated on server *s*, is assigned to VNF-FG

5.2.1. Optimization Formulation

We formulate the problem of VAS placement in CDNs as an optimization problem in this section. The optimization variables are given below:

- $\gamma_s^f \in \{0,1\}$: specifies content server assignment; if γ_s^f is 1, the replica server *s* is selected to serve the content for VNF-FG request *f*.
- $\tau_{k,i}^s \in \{0,1\}$: specifies VNF deployment; if $\tau_{k,i}^s$ is 1, the VNF instance *i* of VNF type *k* is deployed on replica server *s*.
- $\lambda_{s,k,i}^f \in \{0,1\}$: specifies VNF assignments to the requests; if $\lambda_{s,k,i}^f$ is 1, the instance *i* of VNF type *k* instantiated on replica server *s* is assigned to VNF-FG request *f*.

Our model operates over two network snapshots: the current snapshot and the next one. The network configuration for already-deployed services is considered in the current snapshot; while new reconfiguration will be managed for the next snapshot when new requests arrive at the CDN provider. Accordingly, we define the variables belonging to the current snapshot with a ˜ accent. In the rest of this section, we first discuss the objective function and then explain the constraints. The objective is to minimize the total costs, as shown in Eq. (1).

$$Min\left(\Delta C_{hst} + C_{mig} + C_{inst} + \Delta C_r\right) \tag{1}$$

The hosting cost, ΔC_{hst} in Eq. (1), is the differential cost of resources between the current and the next snapshots. Eq. (2) shows the calculation. Note that some resources may be released during reconfigurations, an action which could contribute to cost reduction.

$$\Delta C_{hst} = \sum_{s\in S}\sum_{k\in K}\sum_{i\in I_k} R_k.\rho_s.(\tau_{k,i}^s - \tilde{\tau}_{k,i}^s) \tag{2}$$

The migration cost, C_{mig} in Eq. (1), is calculated by Eq. (3). This is the total cost for migrating the already-deployed VNFs from one replica server to another.

$$C_{mig} = \sum_{s,t\in S} \sum_{k\in K} \sum_{i\in I_k} \varphi_k^{s,t} . \tilde{\tau}_{k,i}^{s} . \tau_{k,i}^{t} \tag{3}$$

The VNF instantiation cost, C_{inst} in Eq. (1), is the total software license costs for new VNF instantiations. Eq. (4) shows the calculation.

$$C_{inst} = \sum_{s\in S} \sum_{k\in K} \sum_{i\in I_k} L_k . (\tau_{k,i}^{s} - \tilde{\tau}_{k,i}^{s}) \tag{4}$$

The routing cost, ΔC_r in Eq. (1), is calculated by Eq. (5). The routing cost is the differential cost of the assigned links between next and current snapshots. The first term is the routing cost between the content server and the first VNF of a VNF-FG request. The second term is the routing between the VNFs of the VNF-FG request. The third term indicates the routing cost between the last VNF and the end-user of a VNF-FG request. Note that the rerouting of already-existing VNF-FG requests in currthe ent snapshot that could happen as a result of reconfigurations has been included in Eq. (5).

$$\begin{aligned}\Delta C_r = &\sum_{\forall f\in F} \sum_{s,t\in S} \sum_{k\in K} \sum_{i\in I_k} \left(\gamma_s^f . \lambda_{t,fst_f,i}^f - \tilde{\gamma}_s^f . \tilde{\lambda}_{t,fst_f,i}^f \right) . C_{s,t} . T_f \\ &+ \sum_{\forall f\in F} \sum_{s,t\in S} \sum_{k,l\in K} \sum_{i,j\in I_{k,l}} (\lambda_{s,k,i}^f . \lambda_{t,l,j}^f - \tilde{\lambda}_{s,k,i}^f . \tilde{\lambda}_{t,l,j}^f) . C_{s,t} \ . T_f . w_{k,l}^{Cf} \\ &+ \sum_{\forall f\in F} \sum_{s,t\in N} \sum_{lst_f\in K} \sum_{i\in I_k} (\lambda_{s,lst_f,i}^f - \tilde{\lambda}_{s,lst_f,i}^f) . C_{s,t} . u_t^f . T_f \ . w_{lst_f,u_f}^{Cf}\end{aligned} \tag{5}$$

The parameter $w_{k,l}^f$ in Eq. (5) is the coefficient of traffic variation relative to the initial traffic of VNF-FG request f. This coefficient is calculated by the multiplication of the traffic variation coefficient of VNF k and its predecessor VNFs, i.e. $pred(k)$ in the chain of f, or C^f, as shown in (6). Eqs. (7)-(15) are the constraints.

$$w_k^{C^f} = \prod_{i \in pred(k)\ in\ C^f \cup \{k\}} q_i \tag{6}$$

$$\sum_{\forall s \in S} \gamma_s^f = 1 \qquad \forall f \in F \tag{7}$$

$$\gamma_s^f \leq c_s^f \qquad \forall s \in S\ , \forall f \in F \tag{8}$$

$$\sum_{\forall s \in S} \sum_{\forall i \in I_k} \lambda_{s,k,i}^f = 1 \qquad \forall k \in V^f\ , \forall f \in F \tag{9}$$

$$\lambda_{s,k,i}^f \leq \tau_{k,i}^s \qquad \forall s \in S, \forall i \in I_k, \forall k \in V^f\ , \forall f \in F \tag{10}$$

$$\sum_{\forall s \in S} \tau_{k,i}^s \leq 1 \qquad \forall k \in K, \forall i \in I_k \tag{11}$$

$$\sum_{k \in K} \sum_{i \in I_k} R_k . \tau_{k,i}^s \leq \mu . G_s \qquad \forall s \in S \tag{12}$$

$$\sum_{f \in F} T_f . w_k^{C^f} . \lambda_{s,k,i}^f \leq \mu . P_k \quad \forall k \in V^f\ , \forall i \in I_k, \forall s \in S \tag{13}$$

$$\sum_{f \in F} \gamma_s^f . \lambda_{t,fst_f,i}^f . T_f + \sum_{f \in F} \lambda_{s,k,i}^f . \lambda_{t,l,j}^f . T_f . w_l^{C^f} + \sum_{f \in F} \lambda_{s,lst_f,i}^f . u_t^f . T_f . w_{lst_f}^{C^f} \leq \mu . B_{s,t} \qquad \forall s, t \in N \tag{14}$$

$$Comm\_delay_f = \sum_{i \in I_{fst_f}} \sum_{\forall s,t \in S} (D_{s,t} + T_f / B_{s,t}) \,.\, \gamma_s^f \,.\, \lambda_{t,fst_f,i}^f + \sum_{k,l \in V^f} \sum_{\substack{i \in I_k \\ j \in I_l}} \sum_{\forall s,t \in S} (D_{s,t} + T_f . w_l^f / B_{s,t}) \,.\, \lambda_{s,k,i}^f . \lambda_{t,l,j}^f + \sum_{i \in I_{lst_f}} \sum_{\forall s,t \in N} (D_{s,t} + T_f . w_{lst_f}^{Cf} / B_{s,t}) \,.\, \lambda_{s,lst_f,i}^f . u_t^f \quad (15)$$

$$Proc\_delay_f = \sum_{\forall k-1, k \in V^f} \sum_{\forall s \in S} \sum_{\forall i \in I_k} T_f . w_k^{Cf} \,.\, M_k^s . \lambda_{s,k,i}^f \quad \forall f \in F \quad (16)$$

$$Comm\_delay_f + Proc\_delay_f \leq X_f \quad \forall f \in F \quad (17)$$

Eq. (7) ensures that only one content server is selected to serve the VNF-FG request *f*, while Eq. (8) ensures that the content server is selected among the capable replica servers. Eq. (9) ensures that only one instance of each required VNF type is assigned to VNF-FG request *f* and Eq. (10) ensures that the assigned VNF instances are already deployed in the network. Eq. (11) ensures that each VNF instance is deployed not more than once in the network.

Eq. (12) ensures that the replica servers hosting the VNFs are not overloaded. Eqs. (13) and (14) ensure that the VNFs and the communication links, respectively, are not overloaded. Eqs. (15)-(17) ensure that the required QoS (in terms of service delay) for each VNF-FG request is satisfied. The delay is calculated as the sum of the communication delay and the video processing delay.

Equations (5), (14), and (15) are non-linear. However, they can be linearized by replacing them with linear equations. For example, the first term of Eq. (5) can be linearized by introducing an auxiliary variable $X^{f}_{s,t,fst_f,i}$, where:

$$X^{f}_{s,t,fst_f,i} = \gamma^{f}_{s} \, . \lambda^{f}_{t,fst_f,i} \tag{18}$$

$$X^{f}_{s,t,fst_f,i} \leq \gamma^{f}_{s} \qquad \forall s,t \in S, \forall f \in F, \ \forall i \in I_{fst_f} \tag{19}$$

$$X^{f}_{s,t,fst_f,i} \leq \lambda^{f}_{t,fst_f,i} \quad \forall s,t \in S, \forall f \in F, \forall i \in I_{fst_f} \tag{20}$$

$$X^{f}_{s,t,fst_f,i} \geq \lambda^{f}_{t,fst_f,i} + \gamma^{f}_{s} - 1 \quad \forall s,t \in S, \qquad \forall f \in F, \forall i \in I_{fst_f} \tag{21}$$

In this regard, the problem becomes an ILP in which the search space size is exponential, with the parameters including the number of VNF types and instances, number of VNF-FG requests, and the number of replica servers. As will be discussed in next subsections, finding the optimal solution in a practical length of time is not feasible for real medium/large scale scenarios. Therefore heuristics must be employed to solve the optimization problem in a reasonable time.

5.3. The Proposed Heuristics

In this section, we present a Tabu-based algorithm for the problem of VNF placement for VASs. Tabu, a meta-heuristic search approach based on a local search method, has been found to be promising in VNF placement problems [58][82]. It starts from a given initial solution, performs moves to generate new neighbor solutions and investigates their fitness. It moves toward the best neighbor, and the process continues for several iterations. In order to avoid visiting a solution repeatedly, the moves of the visited solutions are placed in the Tabu list. However, some aspiration criteria are defined to remove a certain move from the Tabu list if that move has a sufficiently attractive performance in terms of reconfiguration cost. The Tabu algorithm is outlined in Algorithm 1. We present the various aspects of our proposed method in the following subsections.

Algorithm 1: Tabu Search Algorithm

```
1  $S_0 \leftarrow Initial\ Solution$
2  $S_{current} \leftarrow S_0$
3  $S_{best} \leftarrow S_0$
4  $p \leftarrow 0$
5  **Repeat**:
6      $Neighbors \leftarrow GenerateNeighbors(S_{current})$
7      **for** $each\ neighbor\ \in Neighbors$
8          $Calculate\ the\ cost\ of\ neighbor$
9      **end**
10     $best\_neighbour \leftarrow \underset{Neighbour}{\text{argmin}}\ cost(neighbour)$//Eq. 1
11     $best\_move \leftarrow the\ move\ that\ leads\ to\ the\ best\_neighbor$
12     $p \leftarrow p + 1$
13     **if**$(best\_move\ is\ not\ in\ Tabu\_list)$ //update Tabu list
14                 $Tabu\_list \leftarrow best\_move\ \ for\ i_{tabu}\ iterations$
15     **else if**$(cost(best_{neighbor}) < cost(S_{best}))$//asp. criteria
16         $Tabu\_list \leftarrow\ Tabu\_list \setminus best\_move$
17     **end**
18     $S_{curtent} \leftarrow best\_neighbor$
19     **if**$(cost(S_{curtent}) < cost(S_{best}))$
20          $S_{best} \leftarrow S_{curtent}$
21         $p \leftarrow 0$
22     **end**
23 $until\ p < i_{stop}$
```

5.3.1. Initial Solution

Although the Tabu search can start its exploration with any initial solution (placement), feeding it with a more qualified initial solution can lead to placements with lower reconfiguration cost, which is our defined objective function. Algorithm 2 is the pseudo-code of our proposed algorithm for initial solution generation. The algorithm establishes the new placement around the existing one to reduce the reconfiguration cost. It consists of two main phases. In the first phase (lines 1-5), the algorithm implements the decisions that were made in the current snapshot for the new snapshot as well. Hence, for the already-existing requests in the system, it assigns the replica servers, VNFs instances, and the content servers in the same way they were assigned in the current snapshot.

In the second phase, the algorithm aims at assigning VNFs and content servers to the newly-arrived VNF-FG requests. For VNF assignment, this begins with finding a low-cost VNF among the already-deployed VNFs (Alg.2 lines 8-10). A VNF that has the capacity for serving the VNF-FG request combined with the lowest replica server hosting and communication costs to the previous and the next hops (i.e. content server, VNFs in the VNF-FG, and end-user) is assigned to the request.

If the deployed VNF instances do not have enough capacity for the new VNF-FG request or if the required VNF type is not deployed in the network, the algorithm finds a low-cost replica server to host a VNF of the required type (Alg.2 lines 11-14). The replica server is selected such that it has the capacity for hosting a VNF of the required type and the lowest hosting and communication costs to the previous and the next hops (i.e. content server, VNFs in the VNF-FG, and end-user). Next, the algorithm deploys a VNF with the required type on the selected replica server (Alg.2, line 13) and assigns the VNF to the VNF-FG request (Alg.2, line 15).

Finally, the algorithm finds a content server among the capable ones for the newly-arrived VNF-FG request such that it has the lowest communication cost to the first VNF of the VNF-FG (Alg. 2, lines 17, 18).

Algorithm 2: Initial Greedy Algorithm

```
1   for $each\ existing\ service\ request\ f \in F, s \in S, k \in V^f, i \in I_k$
2       $Assign\ content\ server\ to\ f: \gamma_s^f = \tilde{\gamma}_s^f$
3       $Assign\ VNFs\ to\ f:\ \lambda_{s,k,i}^f = \tilde{\lambda}_{s,k,i}^f$
4       $Assign\ replica\ servers\ to\ VNFs: \tau_{k,i}^s = \tilde{\tau}_{k,i}^s$
5   end
6   for $each\ new\ service\ request\ f \in F$
7       for $each\ required\ VNF\ type: k \in V^f$
8           If $any\ VNF\ of\ type\ k\ is\ already\ deployed\ and\ has\ enough\ capacity$
9               $v \leftarrow find\ a\ low - cost\ \ VNF\ instance$
10          end
11          else
12              $s \leftarrow find\ a\ low - cost\ replica\ server$
13              $v \leftarrow deploy\ a\ VNF\ instance\ of\ type\ k\ on\ server\ s$
14          end
15          $Assign\ VNF\ v\ to\ flow\ f$
16      end
17      $CS \leftarrow Find\ a\ low - cost\ content\ server$
18      $Assign\ CS\ to\ f$
19  end
```

5.3.2. Moves and Tabu List Management

We define five moves to generate neighbors for exploration in Tabu search:

- VNF reassignment: A VNF is selected randomly and moved to another replica server that has enough capacity as well as minimum hosting cost. All of the subsequent requests are also rerouted to the new location.
- Bulk VNF reassignment: A replica server is selected randomly, and all the VNFs on that replica server are migrated to another replica server with enough capacity and minimum hosting cost. All the requests that are served by those VNFs are also re-routed to the new location.
- VNF-FG request reassignment: A VNF-FG request is selected randomly and assigned to another instance of one of its VNFs (with enough capacity to tolerate the request traffic) such that the sum of the hosting costs and the communication costs to the predecessor and successor VNFs in the chain is reduced.
- Bulk VNF-FG request reassignment: A VNF is selected randomly, and all of its VNF-FG requests are assigned to another VNF instance that has a replica server with enough capacity and minimum hosting cost. The former VNF instance and its associated resources are then released.
- Content server reassignment: A VNF-FG request is selected and reassigned to another content server with a minimum communication cost to the first VNF of this VNF-FG request.

The moves that lead to already-visited solutions are marked as "Tabu" and placed in the Tabu list to avoid visiting the same solution repeatedly. The algorithm forbids employing the Tabu moves kept in the Tabu list for i_{tabu} iterations according to the Tabu-tenure concept [83]. However, some aspiration criteria are defined to remove a certain move from the Tabu list. The criteria are satisfied when the move results in a placement with lower reconfiguration costs than the reconfiguration cost of the best found solution. If after i_{stop} iterations, the best found solution is not improved, the Tabu process will be terminated.

5.3.3. Solution Evaluation

We evaluate a placement solution through an aggregation of reconfiguration costs, considering constraint violations as penalties. This directs the search process towards a feasible placement with minimum reconfiguration cost. The fitness of a solution $S_{current}$ is calculated in Eq. (22) as follows:

$$E(S_{current}) = \begin{cases} obj(S_{current}) & \text{If } S_{current} \epsilon \; Feasible\ solution \\ obj(S_{current}) + P(S_{current}) & \text{otherwise} \end{cases} \quad (22)$$

where $P(S_{current})$ presents the penalty function for the current solution. For constraints (12), (13), (14), and (17), the penalty is calculated proportional to the level of violations. Considering the L_c and R_c as the left and right sides of the above-mentioned constraint equations, these constraints can be defined as $L_c \leq R_c$. Accordingly, the penalty is calculated as follows:

$$P(S_{current}) = \sum_{c \in C} y_c . \max(0, L_c - R_c) \quad (23)$$

where y_c is the normalization coefficient required to put the penalty and objective function on the same scale.

5.4. Performance evaluation

In this subsection, we first explain the simulation setup and then we provide the evaluation results.

5.4.1. Simulation Setup

We assume network nodes that are located according to the geographical distribution of the cities in North America. The communication bandwidth between each pair of nodes in the network has been selected randomly as between 100, 1000 and 10000 Mbps [22]. The cost for bandwidth usage is randomly selected between \$0.115 and \$0.09 per GB according to Amazon [84] and IBM's cloud [85] pricing policy. The delay between nodes was selected in the range of 2 to 50 ms, based on WonderNetwork [86], a service that provides real-time delay information between various pairs of locations.

Each replica server has the capacity of 8 vCPU [59], with the usage cost randomly selected between \$5 and \$10 per vCPU [22].

We assume that each VNF-FG request requires 1 to 3 VNF types which are randomly selected among the mixer, transcoder, and compressor, as discussed in subsection 5.2. The traffic variation coefficients are also set according to the discussions in the subsection 5.2 (see Fig. 5.1). The delay threshold of the VNF-FG requests is selected randomly in the range of 1800 to 2000 ms [87]. The initial traffic of the VNF-FG requests is assumed to be 0.5 MB. At least 2 replica servers are selected randomly as the content server for the end-users of each VNF-FG request. The license cost for each VNF instantiation is \$100 [22] and the processing delay is 100 ms for each unit of traffic being processed at the VNF[22]. We assume each VNF uses a medium OpenStack VM with 2 vCPUs for the execution [58]. The migration cost is calculated as the bandwidth cost of migrating the snapshot of the VM that is hosting the VNF, from one replica server to another. Snapshot migration is selected because it imposes less cost compared to other existing methods such as live migration [88][46]. The size of a medium size OpenStack snapshot is considered to be 7 GB. For the experiments, we assume that the whole capacity of the VNFs, replica servers and communication links can be used.

We set i_{tabu} with the value of 20, which we found appropriate for the experiments, to prevent cycling of the placements during the Tabu search process. The parameter i_{stop} was selected to be 50, which provides an appropriate trade-off between the execution time and the placement quality. Table 5.2 summarizes the simulation parameters. The simulation was carried out in JAVA, on a server with 2×12-Core 2.20 GHz Intel Xeon E5-2650 v4 CPUs with 128GB of memory.

Table 5.2. Simulation Parameters for VAS VNF placement.

Parameter	ILP Notation	Value
Link bandwidth capacity (Mb/s)	$B_{i,j}$	Rand [100,1000,10000]
Link bandwidth cost ($/GB)	$C_{i,j}$	Rand [0.115, 0.09]
Link delay (ms)	$D_{i,j}$	2-50
Number of replica servers	-	3-16
Replica servers' capacity (vCPU)	G_s	8
Replica servers' cost ($/vCPU)	ρ_s	5-10
VNF license cost ($)	L_k	100
VNF resource requirements (vCPU)	R_k	2
VNF processing delay (ms)	M_k^s	100
Number of end-users	-	6-60
VNFs in each VNF-FG request	V^f	Rand [1-3]
VNF-FG request delay threshold (ms)	X_f	Rand [1800-2000]
Traffic units of VNF-FG request f (MB)	T_f	0.5

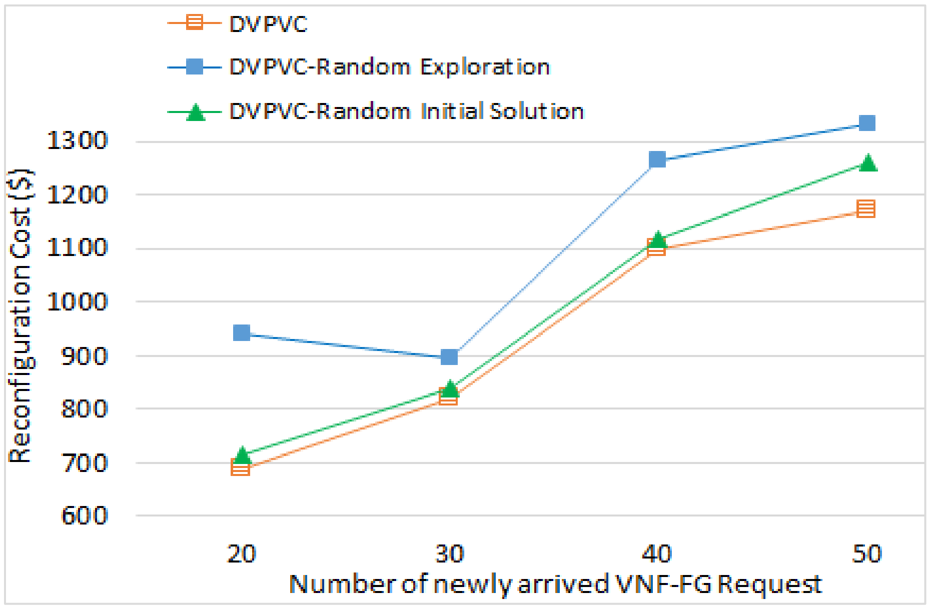


Figure 5.2. Effectiveness of the Greedy Initial Solution and proposed moves for 20-50 newly-arrived VNF-FG requests when 10 VNF-FGs already exist with 8 replica servers offered as NFVI.

5.4.2. Evaluation Results

To evaluate the effectiveness of the proposed initial solution and the moves in the placement quality, we define two modified versions of DVPVC: i) DVPVC- Random Initial Solution, which is the same as the algorithm described in subsection 5.3, but with the initial solution generated based on a random assignment. More precisely, for each required VNF type in a VNF-FG request, an already-deployed VNF is selected randomly and assigned to the VNF-FG request. If such a VNF instance is not found, the Random Initial Solution will instantiate a new VNF on a randomly selected replica server and will assign it to the VNF-FG request. Content servers are also selected randomly. ii) DVPVC- Random Exploration, which exploits random exploration instead of the suggested moves in subsection 5.3.2. Note that in DVPVC-Random Exploration, all the assignments described in the moves (in subsection 5.3.2) are done randomly. For example, for VNF reassignment, a VNF is selected randomly and is assigned to another randomly-selected replica server. Fig. 5.2 shows the reconfiguration cost when there are 8 replica servers in the system. In this figure, we assume 10 VNF-FG requests are already in the system and 20-50 new VNF-FG requests will arrive at the system in the next snapshot. As can be observed in Fig. 5.2, DVPVC outperforms the other two versions. This highlights the effectiveness of cost-oriented decisions made in the initial solution and the proposed moves as specified in subsection 5.3.2. We can also observe that DVPVC-Random Initial Solution shows a better performance than DVPVC-Random Exploration. This underlines the fact that the moves that direct the solution exploration towards a lower cost play more critical roles in cost reduction compared to the initial solution fed to the Tabu as a starting point to begin the search.

To assess the effectiveness of considering the traffic variation in placement decisions, we compare DVPVC with a modified version of it called "DVPVC-constant traffic",

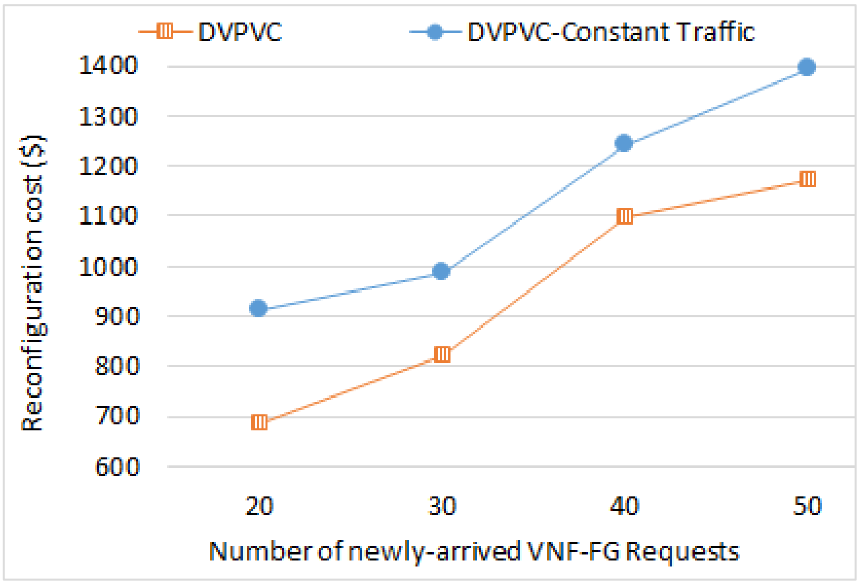


Figure 5.3. The impact of traffic variations on reconfigurations cost for 20-50 VNF-FG requests when 10 VNF-FGs already exist, with 8 replica servers offered as NFVI.

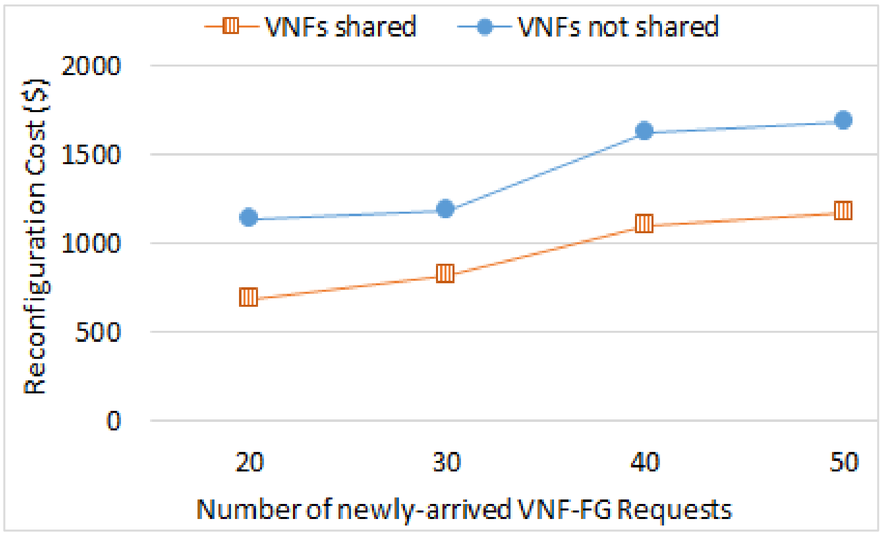


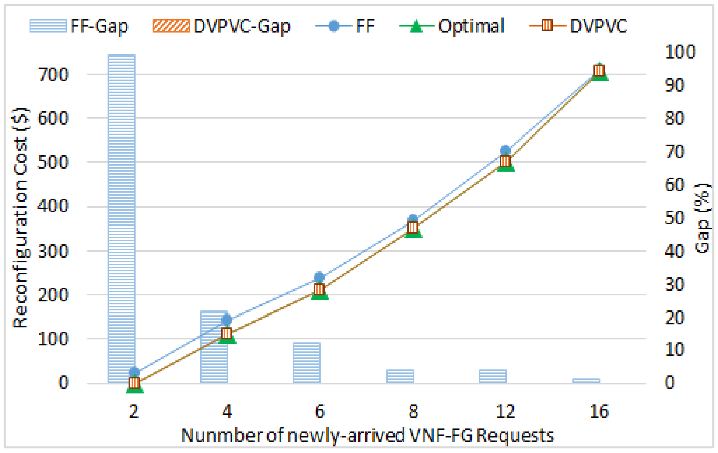
Figure 5.4. The impact of sharing VNFs between existing and new VNF-FG for 20-50 VNF-FG requests when 10 VNF-FG already exists with 8 replica servers offered as NFVI.

where the traffic is considered to be static. In this case, the traffic bitrate does not change as it traverses through the VNFs of the VNF-FG. Fig. 5.3 shows the evaluation results. As can be observed, DVPVC outperforms the modified version of it with constant traffic. We notice that applying the traffic variations yields to reductions of up to 24.7% in reconfiguration costs compared to the DVPVC-constant traffic. This is mainly because a VNF type like the compressor decreases the traffic, which can lead to less processing capacity usage in already-deployed VNFs that are the successors of the compressor in related VNF-FGs. Furthermore, less bandwidth and computation resources are needed to transmit and process the traffic.

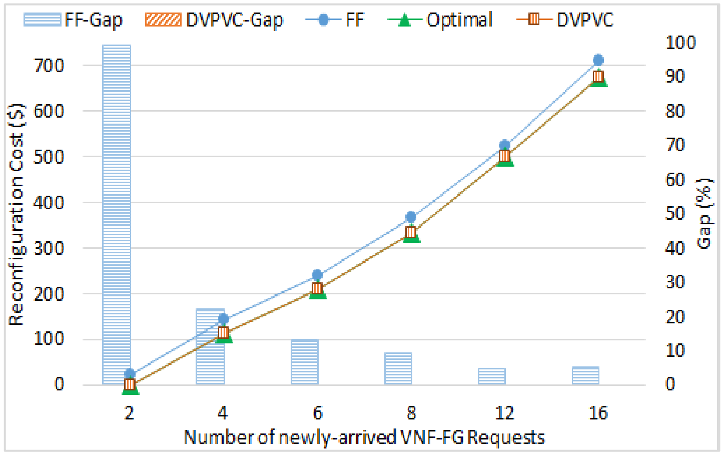
To investigate the impact of reusing the already-deployed VNFs for newly-arrived VNF-FGs, we compare DVPVC with a placement strategy called "VNFs not shared", in which the required VNFs for the newly-arrived VNF-FG requests in the next snapshot are deployed from scratch, without reusing the already-deployed VNFs of the current snapshot. Fig. 5.4 shows the results for 10 existing VNF-FG requests and 8 replica servers. DVPVC has reduced the reconfiguration cost by up to 58.8 % compared to the case where already-deployed VNFs are not shared with newly-arrived VNF-FG requests. This highlights the advantages of reusing already-deployed VNFs to reduce the reconfiguration costs, as advocated in DVPVC.

In the rest of this section we compare DVPVC performance with the optimal solutions obtained by IBM CPLEX 12.08, here called “Optimal”, and a proposed first-fit greedy algorithm, named “First-Fit”. The First-Fit algorithm iterates over the set of existing and newly-arrived VNF-FG requests. For each required VNF type of VNF-FG, it selects the first already-deployed VNF with adequate residual capacity (Eq. (13)) and bandwidth to communicate with the immediate predecessors/successors of the VNF and assigns it to the VNF-FG request. If the appropriate VNF is not found, First-Fit instantiates a VNF of the required type on the first non-saturated replica server in terms of hosting resources (Eq. (12)) with adequate bandwidth for communications with the immediate predecessors/successors of the VNF (Eq. (14)) and assigns it to the VNF-FG request. First-Fit ensures that all constraints defined in subsection 5.2 are satisfied throughout all of the steps.

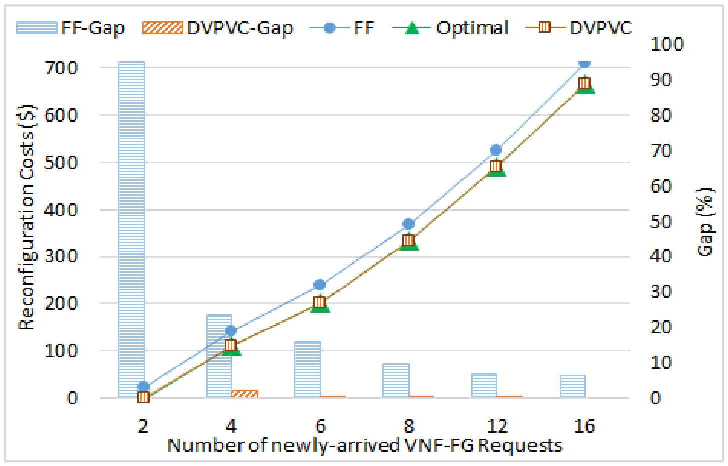
Fig. 5.5 shows the reconfiguration cost when 4 VNF-FG requests already exist in the current snapshot of the system and new requests arrive at the system in the next snapshot. The number of newly-arrived requests changes in the range of 2 to 16. Three sizes of NFVIs with 3, 5, and 8 replica servers are considered. As shown in Fig. 5.5, DVPVC outperforms the First-Fit algorithm and moreover, it reaches optimality, as the results are very close to the optimal ones obtained from CPLEX. In particular, we can see that the average gap to the optimal solution remains less than 2% for DVPVC. On the other hand, the gap between the First-Fit and Optimal performance reaches up to 90%. This very large gap is mainly because First-Fit always chooses the first replica server or VNF for assignments regardless of their imposed costs for VNF hosting and traffic transmission, while DVPVC considers those costs in the placement.



(a)



(b)

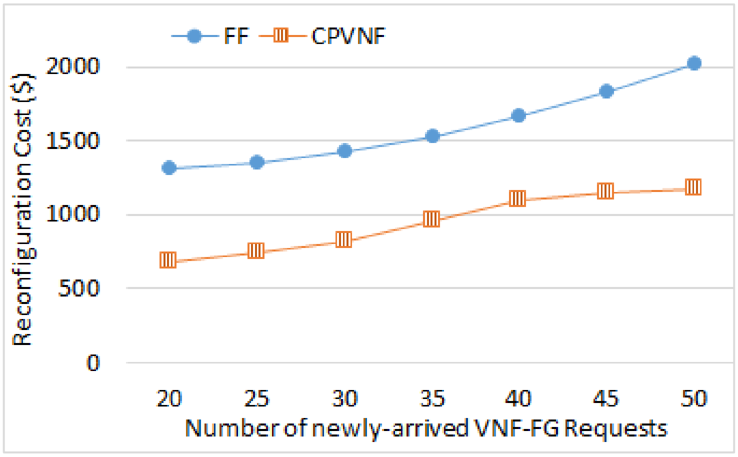


(c)

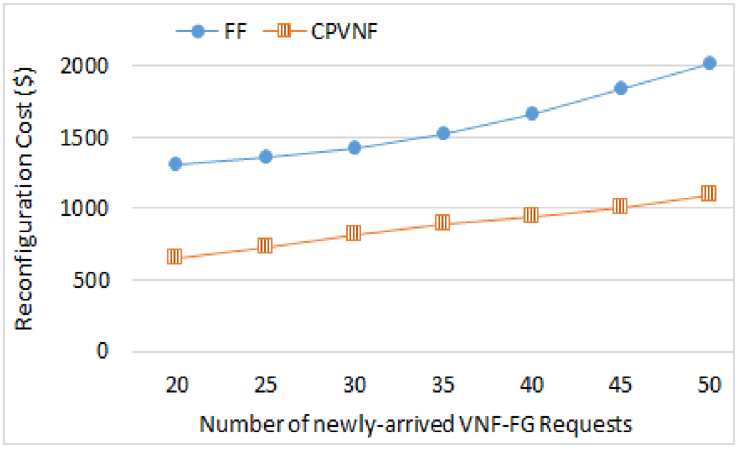
Figure 5.5. Reconfiguration cost for DVPVC, first-fit and optimal solution and the gap from optimality for 2-16 newly arrived VNF-FG requests when 4 VNF-FG requests already exist in the system with (a) 3 servers, (b) 5 servers, and (c) 8 servers.

As shown in Fig. 5.5, when the size of the NFVI increases, the gap between the First-Fit and Optimal approaches increases as well. This demonstrates that the First-Fit placement strategy in larger NFVIs with more diverse resources (from the aspect of cost) leads to placements farther from the optimal. On the other hand, there are only very slight gap increments for the DVPVC, which highlights the effectiveness of our proposed method to find placements with near-optimal reconfiguration cost.

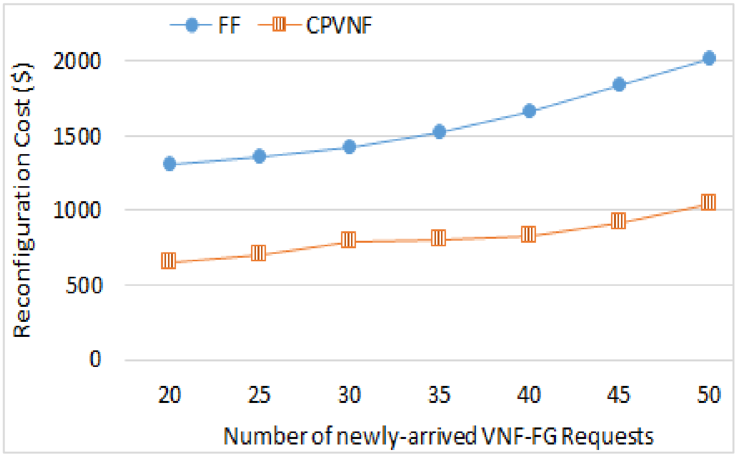
Fig. 5.6 shows the results for a larger scale of NFVI and VNF-FG requests where obtaining an optimal solution in a practical time is not feasible. In this figure, we assume 10 VNF-FG requests already exist in the system and 20-50 new VNF-FG requests arrive at the system in the next snapshot. Different NFVI sizes including 8, 12 and 16 replica servers are considered. As can be seen, DVPVC has reduced the reconfiguration cost by up to 51.3% in comparison with First-Fit, which is much better than its performance in small-scale scenarios (see Fig. 5.5).



(a)



(b)



(c)

Figure 5.6. Reconfiguration cost for DVPVC and first-fit algorithm

for 20-50 new VNF-FGs when 10 VNF-FG requests already exist in the system with (a) 8 servers, (b) 12 servers, and (c) 16 servers.

Table 5.3. Average execution time for optimal placement and DVPVC

Experiment Parameters		Execution Time (s)	
Number of VNF-FG requests	Number of replica servers	Optimal	DVPVC
6	3	29	6
20	3	338	38
6	5	72	12
20	5	665	46
6	8	93	15
20	8	6606	67
30	8	$> 24\ hours$	79

Table 5.3 compares the complexity of the DVPVC and Optimal approaches in terms of execution time. The results show that DVPVC is much faster than Optimal, even for small-scale scenarios where the execution time is in order of seconds. For 20 VNF-FG requests and 8 replica servers, Optimal finds the optimal placement in almost 2 hours, while DVPVC finds the near-optimal placement in 67 seconds. When the number of requests is increased to 30, Optimal could not give the result even though it ran for more than 24 hours.

5.5. Conclusion

This chapter proposes a method, called DVPVC, for the dynamic placement and chaining of VNFs for CDN VASs. The method exploits the reuse of already-deployed VNFs to serve newly-arrived VNF-FG requests. The traffic variations that occur as a result of content customizations performed by VNFs have been considered in the placement decision. The placement problem was modeled as an ILP optimization, which minimizes the reconfiguration costs including VNF hosting, instantiation, migration and traffic routing costs while satisfying the required QoS in terms of delay for all of the already-existing and newly-arrived VNF-FG requests. To increase the problem tractability, a Tabu-based algorithm was proposed to find the near-optimal placement. To improve the quality of the placement a heuristic has been presented for an initial solution. The simulation results show that reusing the already-deployed VNFs and considering traffic variation significantly reduces reconfiguration cost. Furthermore, the proposed algorithm operates very close to optimal placement in small-scale simulations and greatly improves the reconfiguration cost for larger scales of simulations.

Chapter 6

6. Conclusion and Future Work

Network softwarization has emerged in recent years to facilitate the provisioning of CDN components and services. It brings several benefits such as scalability, elasticity, adaptability, and flexibility. However, service and component provisioning in CDNs are still challenging. This thesis addressed key architectural and algorithmic challenges related to network softwarization in CDNs. The key architectural and algorithmic requirements are derived in chapter 2, according to some motivating scenarios. Thereafter, it presented three main contributions.

As the first architectural contribution, in chapter 3, it proposed an architecture for on-the-fly provisioning of CDN components. This is very beneficial especially when CDNs are required to react fast in extending their coverage by provisioning new components when unpredicted surge of requests for a specific content arrives at the CDN. It also enables the CDN component upgrades on-the-fly when needed. The proposed architecture is in-line with ETSI NFV framework and microservices architectural style. The CDN components are designed as a set of microservices and they are implemented, packaged and deployed using NFV technology. Then, the required process is proposed to integrate them into the existing CDN and fill them with the popular contents. The measurements show that the proposed architecture accelerates CDN component deployment and upgrades with minimum service downtime.

To enable on-the-fly provisioning of CDN VASs, chapter 4 of this thesis proposed an architecture based on NFV and SDN technologies. The proposed approach addressed architectural challenges in the dynamic composition of required middle-boxes for CDN VAS provisioning because existing SDN frameworks lack features to support the dynamic chaining of the application-level middle-boxes that are essential building blocks of CDN VASs. Our proposed architecture consists of extensions to the existing SDN switches and controllers to support the dynamic chaining of application-level middle-boxes. Moreover, this approach reduces the significant delay of using application-level middle-boxes, by providing additional features for caching partially and fully customized videos for value-added services. The measurements show the feasibility of the approach and highlight the fact that the caching features reduce the content delivery time significantly. The measurements also lead to an observation about the location of VAS VNFs that has a great impact on QoS.

The observations in chapter 4 yielded to the third contribution of this thesis about the optimal placement of CDN VASs. CDN VASs have a particular characteristic which is the fact that they have an unknown endpoint prior to VNF placement. The few proposals to date that tackle VNF placement for CDN VASs, have focused on static placement. However, such mode of placement is not adequate for dynamic systems such as CDNs, in which the end-user requests are difficult to forecast and may vary in time and space.

On algorithmic side, Chapter 5 proposes a method, called DVPVC, for the dynamic placement and chaining of VNFs for CDN VASs. The traffic variations that occur as a result of content customizations performed by VNFs have been considered in the placement decision. The placement problem was modeled as an ILP optimization, which minimizes the reconfiguration costs including VNF hosting, instantiation, migration and traffic routing costs while satisfying the required QoS in terms of delay for all of the already-existing and newly-arrived VNF-FG requests. To increase the problem tractability, a Tabu-based algorithm was proposed and validated to find the near-optimal placement. The measurements show that the proposed algorithm operates very close to optimal placement in small-scale simulations and greatly improves the reconfiguration cost for larger scales of simulations.

6.1. Future work

This thesis presented significant contributions in the provisioning of service and component for CDNs. Yet, there exist several research directions planned as the future work.

6.1.1. Distributed and virtualized CDN Controller

CDN Controller is a key component in CDN architectures that is in charge of redirecting end-user requests to the appropriate replica server. Traditionally, CDN controllers are designed as centralized building blocks that are hardly coupled to underlying hardware. However, a distributed and virtualized design of CDN controllers brings about notable benefits such as easy deployment and management and also an improved end-user perceived QoS, since they can be deployed at the edge of the network, close to end-users. For example in architectures that replica servers are designed as NFV-based micro-caches that can be deployed at the edge of the network such as home set-top boxes, local light-weight CDN controllers can be deployed close to the micro-caches, so that they redirect the end-user request to the local micro-caches if the requested content is available there. Therefore the end-user request does not need to traverse the whole network to hit the centralized CDN controller and leads to less bandwidth usage and reduces the load on the centralized CDN controller. It would be interesting to investigate the enabling architectures, placement algorithms, and collaboration models between distributed CDN controllers, to realize the distributed and virtualized CDN controllers.

6.1.2. VAS VNF chain composition

Chapter 4 proposed an architecture for on-the-fly provisioning of CDN VASs. As observed in the measurements, the order of VNFs in the VNF-FG affects the QoS in terms of content delivery latency. In CDN context, when it comes to content customization VNFs, the order of VNFs in VNF-FG does not affect the result; however, it affects the delivery time. For example, it is more efficient to start by reducing the video size via compression so that it takes less time when going through the transcoder. This is regarded as a key stage in NFV resource allocation [28] and is called VNFs-Chain Composition (VNFs-CC) that defines the way that VNFs should be composed in a VNF-FG such that the service provider requirements are met. . The selection of the optimal VNF order is a potential future work. It requires the introduction of innovative algorithms to define the best order of VNFs in a chain such that end-to-end content delivery time is minimized while the end-user QoS are met.

6.1.3. Prediction Algorithms

In Chapter 5, we studied the problem of dynamic placement of VNFs for CDN VASs. In real life, the VNF-FG requests can arrive at the CDNs at any given time. The approaches that dynamically place the VNFs, can reconfigure the VNF topology in the network when a change in service usage pattern happens or when new VASs are introduced that might reuse the already-deployed VNFs. Prediction algorithms claim that they can predict the upcoming VNF-FG requests so that the required resources can be planned and reserved in advance. It would be interesting to investigate the accuracy and the cost in both cases, i.e. the dynamic VNF placements vs. prediction models.

References

[1] M. Wang *et al.*, “An Overview of Cloud Based Content Delivery Networks: Research Dimensions and State-of-the-Art,” in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XX*, A. Hameurlain, J. Küng, R. Wagner, S. Sakr, L. Wang, and A. Zomaya, Eds. Springer Berlin Heidelberg, 2015, pp. 131–158.

[2] G. Pallis and A. Vakali, “Insight and Perspectives for Content Delivery Networks,” *Commun ACM*, vol. 49, no. 1, pp. 101–106, Jan. 2006.

[3] J. Sahoo, M. A. Salahuddin, R. Glitho, H. Elbiaze, and W. Ajib, “A Survey on Replica Server Placement Algorithms for Content Delivery Networks,” *IEEE Commun. Surv. Tutor.*, vol. 19, no. 2, pp. 1002–1026, Second quarter 2017.

[4] “Cisco Visual Networking Index: Forecast and Methodology, 2016–2021,” *Cisco*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html. [Accessed: 28-Nov-2017].

[5] “White paper: ‘Cisco VNI Forecast and Methodology, 2015-2020,’ Cisco. [Available online]: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html [Accessed: 24-April-2018]”.

[6] “DSA? FEO? Transparent Caching? CDN Value-Add Services Demystified - Streaming Media Magazine.” [Online]. Available: http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/DSA-FEO-Transparent-Caching-CDN-Value-Add-Services-Demystified-85376.aspx. [Accessed: 28-Oct-2016].

[7] “CNN.com: Facing a World Crisis | USENIX.” [Online]. Available: https://www.usenix.org/conference/Lisa-2001/cnncom-facing-world-crisis. [Accessed: 26-Oct-2018].

[8] “NFV, GS. ‘001: Network Functions Virtualisation (NFV); Use Cases, V 1.2. 1.’ ETSI.”.” May-2017.

[9] S. W. Brim and B. E. Carpenter, “Middleboxes: Taxonomy and Issues.” [Online]. Available: https://tools.ietf.org/html/rfc3234. [Accessed: 24-Jul-2016].

[10] N. Bouten *et al.*, “Towards NFV-based multimedia delivery,” in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 738–741.

[11] N. T. Jahromi, R. H. Glitho, A. Larabi, and R. Brunner, “An NFV and microservice based architecture for on-the-fly component provisioning in content delivery networks,” in *2018 15th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2018, pp. 1–7.

[12] N. T. Jahromi, R. H. Glitho, A. Larabi, and R. Brunner, “Microservices and Network Function Virtualization for on-the-fly Replica Server Provisioning in Content Delivery Networks,” *IEE Commun. Mag.*,-Under Revision.

[13] N. Dragoni *et al.*, “Microservices: yesterday, today, and tomorrow,” *Present Ulterior Softw. Eng. Springer*, p. (In press), Jun. 2016.

[14] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, “Network function virtualization: Challenges and opportunities for innovations,” *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[15] ETSI GS NFV, “Network Functions Virtualisation (NFV): Architectural Framework”, V 1.2.1, Dec., 2014.

[16] N. T. Jahromi *et al.*, “NFV and SDN-based cost-efficient and agile value-added video services provisioning in content delivery networks,” in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 671–677.

[17] N. T. Jahromi *et al.*, “A prototype for value-added video service provisioning in content delivery networks,” in *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, 2017, pp. 597–598.

[18] C. Mouradian, N. T. Jahromi, and R. H. Glitho, “NFV and SDN - based Distributed IoT Gateway for Large-Scale Disaster Management,” *IEEE Internet Things J.*, pp. 1–1, 2018.

[19] N. Tahghigh Jahromi, S. Kianpisheh, M. Abu-Lebdeh, C. Mouradian, and R. H. Glitho, “DVPVC: Dynamic VNF Placement of Value Added Services in Content Delivery Networks,” *IEEE J. Sel. Areas Commun.*,-Submitted.

[20] N. T. Jahromi, S. Kianpisheh, and R. H. Glitho, “Online VNF Placement and Chaining for Value-added Services in Content Delivery Networks,” presented at the IEEE International Symposium on Local and Metropolitan Area Networks, Washington, Dc, 2018.

[21] J. G. Herrera and J. F. Botero, “Resource Allocation in NFV: A Comprehensive Survey,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[22] M. Dieye *et al.*, “CPVNF:Cost-efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks,” *IEEE Trans. Netw. Serv. Manag.*, vol. PP, no. 99, pp. 1–1, 2018.

[23] S. Ahvar *et al.*, “PCPV: Pattern-based Cost-efficient Proactive VNF Placement and Chaining for Value-Added Services in Content Delivery Networks,” presented at the IEEE Netsoft, Canada, 2018.

[24] N. Herbaut, D. Negru, D. Dietrich, and P. Papadimitriou, “Service chain modeling and embedding for NFV-based content delivery,” in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

[25] G. Schaffrath *et al.*, “Network Virtualization Architecture: Proposal and Initial Prototype,” in *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, New York, NY, USA, 2009, pp. 63–72.

[26] N. M. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, Apr. 2010.

[27] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, “Network Function Virtualization: State-of-the-Art and Research Challenges,” *IEEE Commun. Surv. Tutor.*, vol. 18, no. 1, pp. 236–262, First quarter 2016.

[28] J. G. Herrera and J. F. Botero, “Resource Allocation in NFV: A Comprehensive Survey,” *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[29] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, “A Survey on Software-Defined Networking,” *IEEE Commun. Surv. Tutor.*, vol. 17, no. 1, pp. 27–51, First quarter 2015.

[30] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[31] “Handigol, Nikhil, et al. ‘Plug-n-Serve: Load-balancing web traffic using OpenFlow.’ ACM Sigcomm Demo 4.5 (2009): 6.”

[32] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, “Software-defined Internet Architecture: Decoupling Architecture from Infrastructure,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, New York, NY, USA, 2012, pp. 43–48.

[33] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, “Feature-based comparison and selection of Software Defined Networking (SDN) controllers,” in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, pp. 1–7.

[34] ONF Solution Brief, “OpenFlow-enabled SDN and network functions virtualization.” Open Networking Foundation, 2014.

[35] “Netflix Open Connect | Open Connect.” [Online]. Available: https://openconnect.netflix.com/en/. [Accessed: 25-Oct-2018].

[36] “Cloud Delivery, Performance, and Security | Akamai.” [Online]. Available: https://www.akamai.com/. [Accessed: 25-Oct-2018].

[37] ETSI GS NFV, “Network Functions Virtualisation (NFV); Use Cases,” V 1.2.1, May, 2017.

[38] P. A. Frangoudis, L. Yala, and A. Ksentini, “CDN-as-a-Service Provision over a Telecom Operator’s Cloud,” *IEEE Trans. Netw. Serv. Manag.*, vol. PP, no. 99, pp. 1–1, 2017.

[39] P. Veitch, M. J. McGrath, and V. Bayon, “An instrumentation and analytics framework for optimal and robust NFV deployment,” *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 126–133, Feb. 2015.

[40] D. L. Tennenhouse and D. J. Wetherall, “Towards an Active Network Architecture,” *SIGCOMM Comput Commun Rev*, vol. 37, no. 5, pp. 81–94, Oct. 2007.

[41] S. R. Srinivasan, J. W. Lee, D. L. Batni, and H. G. Schulzrinne, “ActiveCDN: Cloud Computing Meets Content Delivery Networks,” 2011.

[42] S. R. Srinivasan *et al.*, “NetServ: Dynamically Deploying In-network Services,” in *Proceedings of the 2009 Workshop on Re-architecting the Internet*, New York, NY, USA, 2009, pp. 37–42.

[43] K. Giotis, Y. Kryftis, and V. Maglaris, “Policy-based orchestration of NFV services in Software-Defined Networks,” in *2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–5.

[44] H. Liu, Y. Hu, G. Shou, and Z. Guo, “Software Defined Networking for HTTP video quality optimization,” in *2013 15th IEEE International Conference on Communication Technology (ICCT)*, 2013, pp. 413–417.

[45] M. Wichtlhuber, R. Reinecke, and D. Hausheer, “An SDN-Based CDN/ISP Collaboration Architecture for Managing High-Volume Flows,” *IEEE Trans. Netw. Serv. Manag.*, vol. 12, no. 1, pp. 48–60, Mar. 2015.

[46] C. Mouradian, T. Saha, J. Sahoo, R. Glitho, M. Morrow, and P. Polakos, “NFV based gateways for virtualized wireless sensor networks: A case study,” in *2015 IEEE International Conference on Communication Workshop (ICCW)*, 2015, pp. 1883–1888.

[47] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, “Dynamic chaining of Virtual Network Functions in cloud-based edge networks,” in *2015 1st IEEE Conference on Network Softwarization (NetSoft)*, 2015, pp. 1–5.

[48] W. Ding, W. Qi, J. Wang, and B. Chen, “OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV,” *IEEE Netw.*, vol. 29, no. 3, pp. 30–35, May 2015.

[49] J. Martins *et al.*, “ClickOS and the Art of Network Function Virtualization,” in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, Berkeley, CA, USA, 2014, pp. 459–473.

[50] S. Paul and R. Jain, “OpenADN: Mobile apps on global clouds using OpenFlow and Software Defined Networking,” in *2012 IEEE Globecom Workshops*, 2012, pp. 719–723.

[51] F. S. Tegueu, S. Abdellatif, T. Villemur, P. Berthou, and T. Plesse, “Towards application driven networking,” 2016, pp. 1–6.

[52] L. Dolberg, J. François, S. R. Chowdhury, R. Ahmed, R. Boutaba, and T. Engel, “A generic framework to support application-level flow management in software-defined networks,” in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 121–125.

[53] “VNF-P: A model for efficient placement of virtualized network functions - IEEE Conference Publication.” [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7014205/. [Accessed: 12-Mar-2018].

[54] “Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions - IEEE Conference Publication.” [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7140281. [Accessed: 25-Oct-2018].

[55] “A Scalable Algorithm for the Placement of Service Function Chains - IEEE Journals & Magazine.” [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7530847/. [Accessed: 06-Mar-2018].

[56] I. Benkacem, T. Taleb, M. Bagaa, and H. Flinck, “Optimal VNFs Placement in CDN Slicing Over Multi-Cloud Environment,” *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 616–627, Mar. 2018.

[57] H. Ibn-Khedher, E. Abd-Elrahman, and A. Kamal, “OPAC: An optimal placement algorithm for virtual CDN,” *Comput. Netw.*, vol. 120, pp. 12–27, Jun. 2017.

[58] M. Abu-Lebdeh, D. Naboulsi, R. Glitho, and C. Wette Tchouati, “On the Placement of VNF Managers in Large-Scale and Distributed NFV Systems,” *IEEE Trans. Netw. Serv. Manag.*, vol. 14, no. 4, pp. 875–889, Dec. 2017.

[59] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, “Elastic virtual network function placement,” in *2015 IEEE 4th International Conference on Cloud Networking (CloudNet)*, 2015, pp. 255–260.

[60] J. Xia, Z. Cai, and M. Xu, “Optimized Virtual Network Functions Migration for NFV,” in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, 2016, pp. 340–346.

[61] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, “An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures,” *IEEEACM Trans. Netw.*, vol. 25, no. 4, pp. 2008–2025, Aug. 2017.

[62] “A workload characterization study of the 1998 World Cup Web site - IEEE Journals & Magazine.” [Online]. Available: http://ieeexplore.ieee.org/abstract/document/844498/. [Accessed: 05-Apr-2018].

[63] I. Sodagar, “The MPEG-DASH Standard for Multimedia Streaming Over the Internet,” *IEEE Multimed.*, vol. 18, no. 4, pp. 62–67, Apr. 2011.

[64] R. Pantos and W. May, “HTTP Live Streaming, RFC 8216, RFC Editor.” Aug-2017.

[65] “NFV, GS. ‘001: Network Functions Virtualisation (NFV); Use Cases, V 1.2. 1.’ ETSI.” May-2017.

[66] S. Patanjali, B. Truninger, P. Harsh, and T. M. Bohnert, “CYCLOPS: A micro service based approach for dynamic rating, charging amp; billing for cloud,” in *2015 13th International Conference on Telecommunications (ConTEL)*, 2015, pp. 1–8.

[67] “NFV, GS. ‘Network Functions Virtualisation (NFV) Management and Orchestration; Report on Architectural Options, V 1.1. 1.’ ETSI.” Jul-2016.

[68] “Smart Applications on Virtual Infrastructure.” [Online]. Available: https://www.savinetwork.ca/. [Accessed: 25-Oct-2018].

[69] “Alfresco Software and Services | ECM | BPM.” [Online]. Available: https://www.alfresco.com/. [Accessed: 25-Oct-2018].

[70] “Docker,” *Docker*. [Online]. Available: https://www.docker.com/. [Accessed: 25-Oct-2018].

[71] “Docker Hub.” [Online]. Available: https://hub.docker.com/. [Accessed: 25-Oct-2018].

[72] F. L. Presti, N. Bartolini, and C. Petrioli, “Dynamic replica placement and user request redirection in content delivery networks,” in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, 2005, vol. 3, p. 1495–1501 Vol. 3.

[73] F. Hu, Q. Hao, and K. Bao, “A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation,” *IEEE Commun. Surv. Tutor.*, vol. 16, no. 4, pp. 2181–2206, Fourth quarter 2014.

[74] “Architectural Styles and the Design of Network-based Software Architectures.” [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. [Accessed: 28-Oct-2016].

[75] A. Doria *et al.*, “Forwarding and Control Element Separation (ForCES) Protocol Specification,” RFC Editor, RFC5810, Mar. 2010.

[76] S. K. Rao, “SDN and its use-cases-NV and NFV.” NEC Technologies India Limited, White paper, 2014.

[77] “Home,” *OPNFV*. [Online]. Available: https://www.opnfv.org/. [Accessed: 25-Oct-2018].

[78] *Mirror of git://source.ffmpeg.org/ffmpeg.git. Contribute to FFmpeg/FFmpeg development by creating an account on GitHub*. FFmpeg, 2018.

[79] L. Qu, C. Assi, and K. Shaban, “Delay-Aware Scheduling and Resource Optimization With Network Function Virtualization,” *IEEE Trans. Commun.*, vol. 64, no. 9, pp. 3746–3758, Sep. 2016.

[80] S. Mehraghdam, M. Keller, and H. Karl, “Specifying and placing chains of virtual network functions,” in *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, 2014, pp. 7–13.

[81] B. Addis, B. Dallal, M. Bouet, and S. Stefano, “Virtual network functions placement and routing optimization,” presented at the CloudNet, 2015, pp. 171–177.

[82] W. Wang, P. Hong, D. Lee, J. Pei, and L. Bo, “Virtual network forwarding graph embedding based on Tabu Search,” presented at the 9th International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, china, 2017.

[83] M. Laguna, “Tabu Search,” *Handb. Heuristics Springer*, pp. 741–758, 2018.

[84] “Amazon AWS GovCloud (US) Data Transfer Pricing in the AWS GovCloud (US) Region,” *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/govcloud-us/pricing/data-transfer/. [Accessed: 25-Oct-2018].

[85] “Bandwidth Packaging & Pricing - Cloud | IBM.” [Online]. Available: https://www.ibm.com/cloud/bandwidth. [Accessed: 25-Oct-2018].

[86] “Network Testing Solutions,” *WonderNetwork*. [Online]. Available: https://wondernetwork.com/. [Accessed: 25-Oct-2018].

[87] R. A. Cacheda, D. C. Garc´ıa, A. Cuevas, and F. Casta˜no, “QoS requirements for multimedia services,” *Resour. Manag. Satell. Netw. Springer Boston MA*, pp. 67–94, Jan. 2007.

[88] C. Mouradian *et al.*, “Network functions virtualization architecture for gateways for virtualized wireless sensor and actuator networks,” *IEEE Netw.*, vol. 30, no. 3, pp. 72–80, May 2016.