A hybrid NFV/In-Network Computing MANO Architecture for provisioning Holographic Applications

Farzaneh Ghasemi Javid

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Master of Computer Science) at

Concordia University

Montréal, Québec, Canada

November 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Farzaneh Ghasemi Javid**

Entitled: **A hybrid NFV/In-Network Computing MANO Architecture for provisioning Holographic Applications**

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

Master of Applied Science (Master of Computer Science)

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

Signed by the Final Examining Committee:

_____ Chair
Dr. S Cespedes

_____ Examiner
Dr. A Bentaleb

_____ Examiner
Dr. S Cespedes

_____ Supervisor
Dr. R. Glitho

Approved by _____
Joey Paquet, Chair
Department of Computer Science and Software Engineering

_____ 2024

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

A hybrid NFV/In-Network Computing MANO Architecture for provisioning Holographic Applications

Farzaneh Ghasemi Javid

The emergence of innovative holographic applications, such as holographic concerts, have requirements on network infrastructure such as high bandwidth and ultra-low latency. As demonstrated in recent research, for efficient provisioning of these types of applications, we require a Hybrid Network Function Virtualization (NFV) / IN-Network Computing (INC) network infrastructures.

INC, a novel technology, distributes computational workloads across the network by deploying computational tasks on programmable devices like routers and switches. However, integrating INC into existing infrastructure presents significant challenges in terms of life-cycle management and orchestration of network functions and components. The network functions in these hybrid environments consist of two types of components, including VNFs and INCs. While the ETSI Management and Orchestration (MANO) framework enables application provisioning in NFV-enabled networks, it does not provide support for INC-enabled environments. Therefore it is necessary to develop a new MANO architecture capable of provisioning applications that incorporate both VNF and INC components.

The main contribution of this thesis is twofold. First, we propose a novel hybrid NFV-INC MANO architecture which is specifically designed for provisioning holographic applications within a hybrid NFV/INC environment. This architecture builds upon the foundation established by the ETSI NFV MANO framework. Second, the proposed architecture is prototyped and measurements are made to evaluate the deployment time of the hybrid Network Services and showing that deploying INC components takes less time than deploying VNFs.

The essential NFV-INC MANO functional entities, such as INC management module, NFV-INC orchestration, and INC infrastructure manager are identified. In addition, a set of RESTful interfaces is proposed to enable interaction with components.

To validate the proposed concept, the prototype is constructed utilizing Open Source MANO (OSM) for orchestration and management network services. Microk8s is employed for container orchestration, automating the deployment of Kubernetes Network Functions (KNFs). Also, the Mininet emulator prepares the programmable switches which serve as the INC infrastructure. The performance of the proposed architecture is evaluated through comprehensive measurements. Additionally, the architecture is validated by concrete measurements on the deployment latency for five different test cases.

Acknowledgments

First of all, I would like to give my sincere thanks to my supervisor Dr. Roch Glitho, without whose guidance and constant support I would not have reached this far. I wholeheartedly thank him for accepting me as his student and for helping me stay on track and progress well by providing me with his expertise. His patience, knowledge, and motivation truly helped me in not only enhancing my abilities but also in overcoming challenging tasks that seemed impossible initially.

I would also like to thank my colleagues at Concordia University for their guidance, help, and encouragement. I would like to especially thank Felipe Estrada-solano and Mouhamad Dieye for helping me and guiding me at every stage of the thesis and providing me with their valuable advice whenever I faced issues.

Finally, I am forever indebted to my parents Farkhondeh and Alireza and my sister Bahareh, without whose constant motivation, encouragement, and support I would not have reached this far in life and accomplished so much. I can never thank them enough for supporting me through every step of my journey and always believing in me incessantly.

Contents

List of Figures xi

List of Tables xii

Abbreviations 1

1 Introduction 4

1.1 Definitions . . . 4

1.1.1 Holographic Applications . . . 4

1.1.2 In-Network Computing . . . 5

1.1.3 NFV Management and Orchestration (MANO) Architectures . . . 5

1.2 Motivation . . . 6

1.3 Problem Statement . . . 6

1.4 Thesis Contributions . . . 7

1.5 Thesis Organization . . . 7

2 Background 9

2.1 Holographic Applications . . . 9

2.1.1 General Definition of Holographic Applications . . . 9

2.1.2 Requirements of Holographic Applications . . . 11

2.1.2.1 High Bandwidth . . . 11

2.1.2.2 Ultra-low latency . . . 12

2.2 In-Network Computing . . . 12

2.2.1 Definition . . . 12
2.2.2 Software Defined Network . . . 13
2.2.3 Benefits of In-Network Computing . . . 15
2.2.3.1 High Throughput . . . 15
2.2.3.2 Low Latency . . . 15
2.2.3.3 Bandwidth Usage Reduction . . . 16
2.2.3.4 Load Balancing . . . 16
2.2.3.5 Energy Efficiency . . . 16
2.3 NFV Management and Orchestration . . . 17
2.3.1 Definition . . . 17
2.3.2 NFV MANO . . . 18
2.3.3 Procedures . . . 19
2.4 Conclusion . . . 21

3 Use cases and State of the art **22**

3.1 Use Case . . . 22
3.1.1 Holographic concert scenario . . . 22
3.2 Requirements . . . 24
3.2.1 *A hybrid management module* . . . 24
3.2.2 *A hybrid orchestration module* . . . 24
3.2.3 *Hybrid infrastructure and infrastructure manager* . . . 25
3.2.4 *A hybrid network service* . . . 25
3.3 State of the Art . . . 25
3.3.1 Works on resource provisioning for holographic applications in hybrid NFV/INC networks . . . 26
3.3.2 Works on the integration of NFV and INC . . . 27
3.3.3 Works on the implementation of NFV MANO . . . 28
3.3.4 Works on the extension of NFV MANO . . . 29

3.3.5 Summary of the State of the Art of the Hybrid NFV/INC MANO Architectures for Holographic Applications . . . 30
3.4 Conclusion . . . 30

4 Proposed Architecture **31**
4.1 Proposed Architecture . . . 31
4.1.1 The untouched modules . . . 32
4.1.1.1 *Operation/Business Support System (OSS/BSS)* . . . 32
4.1.1.2 *VNF Manager (VNFM)* . . . 33
4.1.1.3 *NFV Resources* . . . 33
4.1.1.4 *VNF Catalog* . . . 33
4.1.1.5 *Element Management (EM)* . . . 33
4.1.2 The modified modules . . . 34
4.1.2.1 *NFV/INC Infrastructure Manager (NIIM) and NFV/INC Infrastructure (NII)* . . . 34
4.1.2.2 *NFV/INC Orchestrator (NIO)* . . . 34
4.1.2.3 *NS Catalog* . . . 35
4.1.2.4 *NS Instances* . . . 35
4.1.2.5 *NII Resources* . . . 35
4.1.3 The new modules . . . 35
4.1.3.1 *INF Manager (INFM)* . . . 35
4.1.3.2 *INF Catalog* . . . 36
4.1.3.3 *NIO-INFM interface (Nio-Infm)* . . . 36
4.1.3.4 *NIIM-INFM interface (Niim-Infm)* . . . 36
4.1.3.5 *INFM-INF interface (Infm-Inf)* . . . 36
4.2 Procedures . . . 37
4.2.1 Deployment of a VNF network function . . . 37
4.2.2 Deletion of a VNF network function . . . 39
4.2.3 Deployment of an INC network function . . . 40

4.2.4 Deletion of a INF network function . . . 41
4.3 Evaluation of Proposed Architecture against the Requirements . . . 42
4.4 Conclusion . . . 43

5 Validation of the Architecture **44**
5.1 Prototype Architecture Overview . . . 44
5.1.1 Used Softwares . . . 44
5.1.1.1 OSM . . . 45
5.1.1.2 MicroK8s . . . 45
5.1.1.3 OpenLDAP . . . 45
5.1.1.4 Mininet . . . 45
5.1.1.5 BMv2 Switch . . . 46
5.1.1.6 Flask . . . 46
5.1.2 Description of Implemented Prototype . . . 46
5.1.3 Implemented Scenario . . . 48
5.1.4 Interfaces . . . 49
5.1.4.1 Interface for deploying an INC component . . . 49
5.1.4.2 Interface for deleting an INC component . . . 50
5.1.5 Programming Language and IDE Used . . . 50
5.1.6 Experimental Setup . . . 51
5.2 Performance Evaluation . . . 51
5.2.1 Proof-of-concept . . . 52
5.2.2 Performance Metric . . . 52
5.2.2.1 Deployment delay . . . 52
5.2.3 Test Cases . . . 53
5.2.4 Results and Analysis . . . 55
5.3 Conclusion . . . 57

6 Conclusion **58**
6.1 Contributions Summary . . . 58

6.2 Future Research Direction 59

List of Figures

Figure 2.1 An Architecture of SDN 13
Figure 2.2 NFV-MANO architecture 19
Figure 2.3 Sequence diagram of VNF instantiation in NFV-MANO 20
Figure 3.1 Summary of the related works for hybrid NFV/INC MANO architecture . . 30
Figure 4.1 NFV/INC-MANO architecture 32
Figure 4.2 Sequence diagram of VNF instantiation in NFV-MANO 38
Figure 4.3 Sequence diagram of VNF termination and deletion in NFV-MANO 39
Figure 4.4 Sequence diagram of INF instantiation in NFV/INC-MANO 40
Figure 4.5 Sequence diagram of INF termination and deletion in NFV/INC-MANO . . 42
Figure 5.1 Prototype implementation 47
Figure 5.2 Sequence diagram of deploying a hybrid Network Service 48
Figure 5.3 Screenshot of INF and VNF deployment 53
Figure 5.4 Screenshot of INF deployment on bmv2 switch in Mininet 54
Figure 5.5 Screenshot of VNF deployment on Mikrok8s 55
Figure 5.6 The Average Deployment Delay for deploying a hybrid Network Service . . 56

List of Tables

Table 5.1 Example of POST API exposed by the NIO to the NIIM 50
Table 5.2 Example of DELETE API exposed by the NIO to the NIIM 51
Table 5.3 The average deployment delay and standard deviation of deploying a network service 57

Abbreviations

6DoF six degrees of freedom.

ACK acknowledgement.

API Application Programming Interface.

BMv2 Behavioral Model version 2.

CLI Command Line Interface.

EM Element Management.

HD high-definition.

HMDs head-mounted displays.

INC IN-Network Computing.

INF In-Network Function.

INFM INF Manager.

K8s Kubernetes.

KNF K8s-based VNF.

LCM Life Cycle Management.

LDAP Lightweight Directory Access Protocol.

LFV Light field video.

MANO Management and Orchestration.

NBI North Bound Interface.

NF Network Functions.

NFV Network Functions Virtualization.

NFVI NFV Infrastructure.

NFVO NFV Orchestrator.

NICs network interface cards.

NII NFV/INC Infrastructure.

NIIM NFV/INC Infrastructure Manager.

NIO NFV/INC Orchestrator.

NS Network Service.

NSO Network Service Orchestrator.

ONAP Open Network Automation Platform.

OSM Open Source MANO.

OSS Operations Support System.

REST Representational State Transfer.

RO Resource Orchestrator.

t-MANO tenant MANO.

VIM Virtual Infrastructure Managers.

VNF Virtual Network Functions.

VNFD VNF Descriptor.

VNFM VNF Manager.

VR virtual reality.

Chapter 1

Introduction

1.1 Definitions

This section starts with an overview of the key terms associated with our research such as Holographic Applications, IN-Network Computing (INC), and Network Functions Virtualization (NFV) Management and Orchestration (MANO) Architectures. Then, the motivation and problem statement are discussed. Finally, the summary of our contributions and organization of the thesis are presented.

1.1.1 Holographic Applications

The functionality of holographic applications involves transmitting and interacting with holographic data remotely. For instance, in a holographic concert, performers in one location can be presented as holograms for an audience in another location, enabling real-time interaction. Unlike 2D and 3D content, holographic content introduces parallax, allowing viewers to engage actively, with the image changing based on perspective. This shift from passive to interactive viewing increases the demands on hardware, such as head-mounted displays (1).

The required network bandwidth for such applications varies from 100 Gb/s to 2 Tb/s, compared to the 1–5 Mb/s bandwidth for HD video transmission. In addition, the ultra-low latency, specifically when using a head-mounted display, is needed to avoid cybersickness. The jitter value need to be within the range of 15 ms, in order to preserve the stability of the hologram (2).

For transmission over the network, the three important functions are encoding, decoding, and transcoding which consists of decoding, reformatting, and re-encoding. Transcoding typically decreases the size of the holograms in order to enhance quality on the receiving endpoint (3).

1.1.2 In-Network Computing

In-Network Computing (INC) is a promising technology that aims to distribute the computational workload across the network by placing network functions on programmable devices (e.g., routers or switches). This technology offers an alternative to computing on servers that are outside the network (4; 5). According to reference (4), this concept is described as a "dumb idea whose time has come". While its origins can be traced back to the active networks of the late 1990s, it has recently gained momentum in the late 2010s and early 2020s, driven by advancements in data plane programmability.

The initial step into data plane programmability is Software-Defined Networking (SDN). This technology separates the control plane and data plane (6). While it permits customization of the control plane, it provides only protocol-dependent actions in the data plane. Additionally, it lacks the capability for in-field runtime re-programmability. Addressing these limitations, the emerging programmable data plane devices, including switches and network interfaces, offer significantly enhanced flexibility. Users can directly program these devices using languages such as P4 (6).

1.1.3 NFV Management and Orchestration (MANO) Architectures

NFV decouples Network Functions (NF) from network hardware and allow them to run on commodity servers (7; 8). This decoupling leads to the creation of Virtual Network Functions (VNF), which interact with both the hardware and software infrastructure in new ways. The infrastructure required for NFV is referred to as NFV Infrastructure (NFVI).

To manage and coordinate these new relationships and operations, a specific architectural framework known as NFV MANO is essential (9). NFV-MANO is composed of two primary components: the NFV Orchestrator (NFVO) and the VNF Manager (VNFM). The NFVO is responsible for allocating resources across various Virtual Infrastructure Managerss (VIMs) to oversee the life cycle of network services for VNFs. Meanwhile, VIMs manage the NFVI resources, and the VNFM

handles the setup and life cycle management of VNFs deployed in virtual machines or containers. (7).

1.2 Motivation

Recent research by (3) underscores the critical role of Hybrid Network Function Virtualization (NFV) / INC network infrastructures for efficiently delivering applications with high bandwidth and ultra-low latency demands, such as holographic applications. NFV provides flexibility and efficiency in network management, while INC distributes computation tasks across the network to meet these stringent requirements.

The study by (3) investigated the impact of transcoder placement on latency, jitter, and network load. According to their scenarios for deploying transcoder on edge server, user's computer, and within the network near the audience, their findings demonstrate that placing the transcoder near the server delivering the hologram (INC) resulted in the lowest latency for most viewers. This is because it avoids sending the large data all the way to the edge server first. All scenarios achieved acceptable jitter levels below 15ms, with slight variations based on network connections. Importantly, INC significantly reduced network load compared to other options by shrinking data size before transmission, minimizing overall traffic.

1.3 Problem Statement

The integration of NFV and INC into a hybrid environment that incorporates both VNFs (virtualized network functions) and INC components (P4 programs) presents a new challenge: managing and orchestrating these different network functions and components. Effective lifecycle management of network functions, including deployment, instantiation, termination, upgrades, and migrations, necessitates dedicated management modules.

The existing Management and Orchestration (MANO) framework for NFV includes a VNF management module that handles VNFs within the network. However, this module lacks the capability to place components on INC-enabled devices such as programmable routers or switches. This limitation necessitates the development of new functional modules or extending the existing

modules of the MANO architecture. For the lifecycle management of an INC component, the development of a new INC management module is needed. Additionally, extending the existing modules such as INC/NFV orchestration module is required to receive the output of the placement algorithm and use it to fetch a component from the repository and determine the appropriate management module (VNFs Management or INC-programs Management) for its placement on network devices. Consequently, defining clear interfaces, protocols, and management and orchestration modules is crucial for effectively placing these components within a hybrid environment.

1.4 Thesis Contributions

- A set of requirements on the general architecture of MANO for a hybrid NFV/INC environment;
- A review on the state of the art solutions for MANO architecture based on our sets of our requirements;
- Proposed extensions of existing functional entities of MANO architecture for hybrid NFV/INC, consisting: NFV/INC Orchestrator (NIO), NFV/INC Infrastructure Manager (NIIM), NFV/INC Infrastructure (NII), Network Service (NS) Catalog, NFV/INC Infrastructure (NII) Resources, and NS Instances;
- Development of new functional entities for hybrid NFV/INC MANO architecture, consisting: INF Manager (INFM), In-Network Function (INF) Catalog, and a set of interfaces including NIO-INFM interface (Nio-Infm), NIIM-INFM interface (Niim-Infm), and INFM-INF interface (Infm-Inf);
- An implementation architecture, a proof-of-concept prototype, and performance evaluation.

1.5 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 presents the background key concepts related to our research domain in detail.

- Chapter 3 introduces the motivating scenarios and the set of requirements derived from these scenarios. The state of the art is also evaluated against the requirements.
- Chapter 4 presents the proposed architecture for the Management and Orchestration of a hybrid NFV/INC environment. Functional entities and the proposed interfaces are discussed.
- Chapter 5 describes the implementation architecture and technologies used for the proof-of-concept prototype. Then the performance measurements evaluating the architecture are presented.
- Chapter 6 concludes the thesis by providing a summary of the overall contributions and identifying the future research directions.

Chapter 2

Background

This chapter presents the background concepts relevant to the research domain of this thesis. The following concepts are explained in the upcoming sections: Holographic Applications (HA), In-Network computing and a brief description of Software-Defined Networking (SDN), and finally the NFV Managemen and Orchestration (MANO).

2.1 Holographic Applications

2.1.1 General Definition of Holographic Applications

Holographic applications revolutionize remote interaction by enabling the real-time transmission, interaction, and manipulation of data representing objects in six degrees of freedom (6DoF) like holographic data. To achieve truly immersive experiences like holographic streaming, six degrees of freedom (6DoF) are essential (10). Unlike traditional video where viewers are passive observers or 360-degree video that only allows head movement, 6DoF enables users to freely move and interact within a virtual environment.

Computer-generated holograms can be categorized into two primary types: image-based and volumetric. Image-based holograms are constructed by assembling a collection of images captured from various perspectives of an object. Light field video (LFV) is a prime example of this approach (11). While conceptually straightforward, image-based holograms present challenges in terms of storage and data transmission due to the vast number of individual images required. For instance,

a hologram encompassing a 30-degree viewing angle and a 10-degree tilt angle would necessitate an array of 3300 images if each image is captured at a 0.3-degree interval. Nevertheless, the redundancy inherent in these image sets allows for the application of advanced compression techniques to mitigate storage and transmission demands, albeit at the expense of increased computational requirements (12).

The contemporary approach to holographic representation leans towards volumetric media, specifically ”point clouds.” Here, objects are depicted as collections of three-dimensional volume pixels, or voxels, within a conceptual three-dimensional space. The visual image is subsequently generated dynamically from any perspective at the receiving end, allowing for the integration of multiple point cloud objects into a single scene. This process entails intricate preprocessing and rendering, utilizing systems equipped with multiple cameras capable of capturing both color and depth information. A key advantage of volumetric media is its inherent compressibility, as each voxel is transmitted only once, irrespective of the number of viewing angles or tilts. To establish a standardized compression format for volumetric media, the MPEG group has recently adopted a reference encoder known as V-PCC (13). This encoder transforms point clouds into two distinct video sequences, encapsulating geometric and textural data, which are subsequently compressed using conventional video coding methods.

Regardless of the specific holographic media format, the processes of compression and decompression introduce computational overhead, which directly impacts the resulting latency. Consequently, a higher compression ratio necessitates increased computational resources and potentially higher latency, while a lower compression ratio demands greater network bandwidth and latency. This trade-off between computational and network resources is a critical factor in the design and optimization of holographic systems (1).

For instance, in a holographic concert the performers which are located at one end-point in the network can be appeared as life-size, interactive holograms for a geographically distributed audience remotely. In this concert, hologram streaming has three main phases: capturing, rendering, and streaming an aim object which is the performer. First the target object is captured by a camera array from multiple various perspectives and angles. The output images are merged and rendered into a hologram. Then this hologram object is encoded and streamed through the network. On

the receiving end point the audiences receive the stream, decode it, and render it to provide the hologram to show on a holographic display. Unlike traditional 2D and 3D content, holographic displays leverage parallax, allowing viewers to actively interact with the content as their perspective change. This transition from passive viewing to active participation necessitates advanced hardware, such as head-mounted displays (HMDs) (1).

Holographic applications necessitate a network infrastructure significantly more demanding than conventional video streaming. The required network bandwidth for such application varies from 100 Gb/s to 2 Tb/s, while for the HD video transmission the 1-5 Mb/s bandwidth is needed. Furthermore, the ultra-low latency is demanded to avoid cybersickness especially when using a head-mounted display. The jitter value need to be within the range of 15 ms, in order to preserve the stability of the hologram. (2).

To facilitate efficient network transmission, holographic data undergoes a three-stage processing pipeline: encoding, decoding, rendering, and transcoding. During transcoding, the data is decoded, reformatted, and re-encoded, typically with the goal of reducing file size while optimizing quality at the receiving end. (3).

2.1.2 Requirements of Holographic Applications

According to (3), provisioning holographic communication and applications necessitates addressing two critical challenges: bandwidth and latency. While data compression techniques can mitigate bandwidth demands to some extent, holographic data inherently requires a significant amount of information to be transmitted. In this section, we delve into these requirements discuss about them in more detail (1):

2.1.2.1 High Bandwidth

The emergence of holographic applications, like HTC streaming, is projected to trigger a significant growth in network bandwidth requirements. These applications require the high bandwidths reaching the terabit-per-second (Tb/s) range, representing a significant increase by several orders of magnitude compared to conventional high-definition (HD) video streaming or even 3D virtual reality (VR) video.

Visual capture technologies like the Microsoft Kinect sensor for Windows v2 generate significant bandwidth demands. Capturing high-resolution color and depth information results in roughly 8.8 MB of raw data per frame, or 2.06 Gb/s of raw data per second at 30 FPS. This burden intensifies with additional sensors, viewpoints, or higher resolutions. Light Field Video (LFV), for example, may require bandwidths exceeding 1 Tb/s. Efficient data compression and robust network infrastructure are crucial to support these evolving data requirements.

Under these circumstances, high-bandwidth demands are undeniable for supporting holographic applications, even with advanced compression and user interaction prediction. 5G uses higher regions of the electromagnetic spectrum to address this, but novel management techniques are crucial to fully exploit its potential.

2.1.2.2 Ultra-low latency

In addition to the need for extremely high bandwidths achievable through novel wireless infrastructures, the 5G paradigm establishes a target latency of 1 millisecond (ms) for round-trip communication. However, achieving such low latency over vast distances remains a significant challenge due to the fundamental physical limitation imposed by the speed of light. Nonetheless, ultra-low latency is paramount for fostering a truly immersive holographic experience, particularly when utilizing head-mounted displays (HMDs), where latency can induce simulator sickness. Additionally, real-time communication applications necessitate stringent latency requirements for a seamless user experience.

2.2 In-Network Computing

2.2.1 Definition

The term ”in-network computing” lacks a universally agreed-upon definition. One perspective, offered by ACMSIGARCH (14), describes it as running programs typically executed on individual computers directly within the network itself. This approach leverages existing network elements, like switches, that already manage data flow within the network. Alternatively, Sapio et al. (4) define in-network computing by its ability to offload computational tasks from individual computers.

These tasks are instead handled by network elements like switches and specialized network interface cards (NICs). Finally, Ports and Nelson (5) focus on the speed benefits of in-network computing. They define it as the ability to run application-specific functions directly on programmable network hardware at the same speed as data transmission (line rate). This approach offers significant performance advantages, achieving much higher throughput (data processing speed) and lower latency (data transmission delay) compared to traditional server-based computing.

2.2.2 Software Defined Network

The initial step into data plane programmability is Software-Defined Networking (SDN). Figure 2.1 demonstrate a simplified view of an SDN architecture. As stated by (6), the SDN revolutionized network management by introducing a explicit separation between the control plane and data plane. This separation allows the control plane to reside as software on a dedicated controller, independent of individual network switches. While it permits customization of the control plane, it provides only protocol-dependent actions in the data plane. Additionally, it lacks the capability for in-field runtime re-programmability. Addressing these limitations, the emerging programmable data plane devices, including switches and network interfaces, offer significantly enhanced flexibility (6).

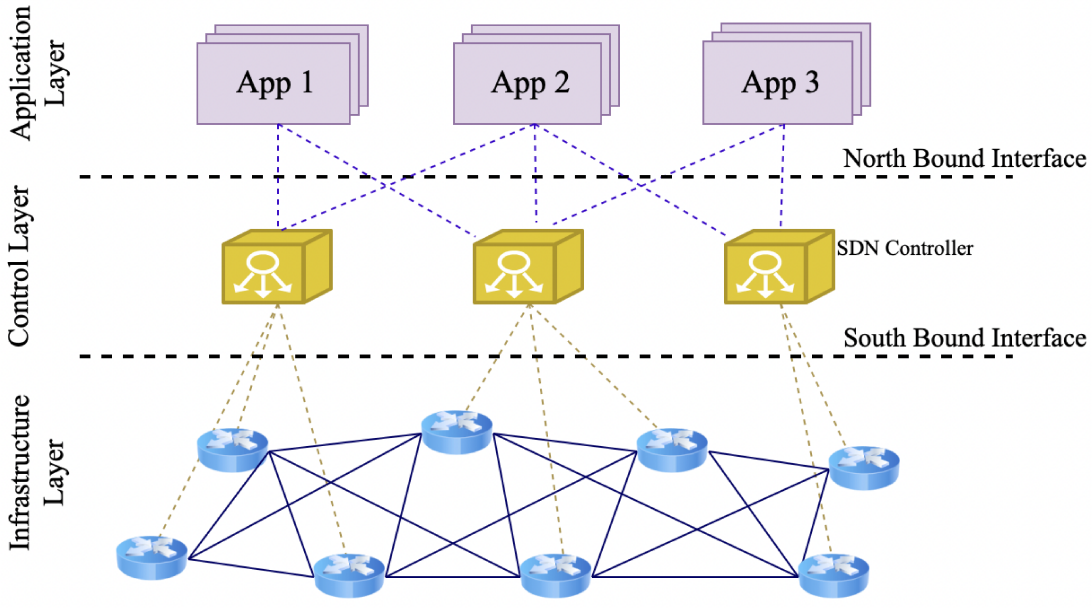

Figure 2.1: An Architecture of SDN

There are four types of SDN interfaces (15):

- *Northbound Interface:* it enables the connection between the network's control center (SDN controller) and the software that uses the network (network application). The software decides what kind of information is shared, how it's formatted, and how often.
- *Southbound Interface:* This link connects the network's control center with the physical network equipment (like switches and routers) so they can communicate and work together.
- *Eastbound Interface:* This is the bridge between the traditional internet (using IP addresses) and the new network (SDN). There's no standard way to do this yet, so it depends on how the old network is built. To make them work together, a special tool is needed to translate information between the two types of networks.
- *Westbound Interface:* This connection lets different control centers in different parts of the network talk to each other. It also makes it possible to manage network traffic smoothly across these different areas while getting a complete picture of the entire network.

In this context, there have been introductions of programming languages like P4, aimed at enabling the programming of networking devices, both in software and hardware. P4, which stands for Protocol-independent Packet Processors, is a specialized language designed for configuring how data plane devices, including switches, NICs, routers, and filters, handle incoming packets. P4 presents the opportunity to leverage the advantages of two core methodologies: the heightened packet processing performance offered by hardware-based networking solutions and the inherent adaptability and programmability found in software-centric network operations (16). Overall, P4 contains the following key elements ():

- **Header Format:** P4 can handle different types of data packets, including standard ones like Ethernet, IP, TCP, or UDP, as well as custom ones. The P4 program needs to clearly define how these packets are structured.
    - *Parser:* The parser figures out the structure of incoming packets. It breaks down the packet into smaller parts and stores this information for later use. The parser can make decisions about what to do next based on the information it finds.

- *Match+action pipeline:* This part decides what to do with the packet based on its contents. It uses a set of rules to match packet information and then performs specific actions, like sending the packet out a certain port or changing its address.
- *Deparser:* The deparser takes the changed packet and prepares it to be sent to the next network device.

- **P4 Compilers:** P4 compilers translate P4 code into instructions that the network hardware can understand. They have two main parts: one that converts P4 code into a general format, and another that changes this general format into instructions for specific hardware. The compiler also creates tools to control the network devices.

- **P4 Runtime:** The P4 Runtime manages communication between the network hardware and the software that controls it. This allows different types of network hardware to work together and gives software the flexibility to control the network from different locations.

2.2.3 Benefits of In-Network Computing

The adoption of in-network computing offers a multitude of advantages. In this section, according to the (17) we propose some of the in-network computing benefits that raise the importance of existing this technology.

2.2.3.1 High Throughput

In-network computing offers a significant advantage in terms of throughput. Network elements within this paradigm are capable of processing data at rates exceeding billions of packets per second. To illustrate this, the Tofino chip developed by Barefoot demonstrates the feasibility of line-rate processing at 12.8 Tb/s. Consequently, in-network computing surpasses host-based solutions by providing orders of magnitude higher throughput processing capacity.

2.2.3.2 Low Latency

In contrast to the inherent variability and delays experienced in host-based solutions, in-network computing offers significant advantages in terms of latency. Network elements within this paradigm

boast sub-microsecond processing latencies. This low and stable latency (jitter) is achieved due to the pipeline design, which avoids accessing external memory at each processing stage. As highlighted by the use cases discussed earlier, in-network computing performs computations directly within the network. This enables transactions to terminate within their path, eliminating the need for detours to distant servers. Consequently, the delays associated with data transmission between network elements and end-hosts are significantly reduced. In essence, in-network computing brings computation closer to the user, achieving a level of proximity that surpasses even edge/cloud computing architectures.

2.2.3.3 Bandwidth Usage Reduction

By performing computations directly within the network, in-network computing enables data processing to be completed on the path, eliminating the need to transmit data all the way to edge or cloud servers. This significantly reduces the amount of data traversing the backhaul links, thereby minimizing bandwidth consumption and alleviating potential traffic congestion.

2.2.3.4 Load Balancing

In-network computing inherently introduces a novel form of load balancing. By strategically processing requests within the network itself, network elements can offload computation from traditional end-hosts. This capability fosters a distribution of workload, enabling network elements to handle latency-sensitive applications, while tasks with less stringent latency requirements can be directed to end-hosts. For instance, traffic generated by latency-tolerant applications could be forwarded to end-hosts for processing, while network elements could prioritize and handle requests demanding ultra-low latency.

2.2.3.5 Energy Efficiency

In-network computing presents a significant advantage in terms of energy consumption compared to traditional host-based processing. Network elements employed in this paradigm are inherently more energy-efficient for performing computations. This is due to their specialized architecture, enabling them to execute billions of operations per watt of energy utilized. For instance, the

Arista 7170 series programmable switch demonstrates exceptional efficiency, consuming less than 5 watts per 100 Gigabit Ethernet port. Further substantiating this claim, research presented in (18) indicates that processing millions of queries within network elements can be achieved with a power consumption of less than 1 watt. This translates to a significant improvement in processing efficiency per unit of energy compared to general-purpose computers. Additionally, network elements are designed for packet forwarding as their primary function, resulting in minimal energy consumption during idle periods. In contrast, general-purpose computers often experience higher baseline energy consumption even when not actively processing tasks.

2.3 NFV Management and Orchestration

2.3.1 Definition

Network Functions Virtualization (NFV), as explored by (7) and (8), stands as a transformative technology for next-generation networks. It fundamentally disrupts the traditional approach by decoupling network functions (NFs) from specialized hardware. This separation allows NFs to operate on readily available, commercially-produced servers, often referred to as ”commodity servers”. This shift offers significant advantages.

Firstly, NFV fosters the creation of Virtual Network Functions (VNFs). Unlike their hardware-bound predecessors, VNFs exist purely in software. This software-defined nature enables them to interact with the underlying infrastructure, encompassing both hardware and software components, in novel and more flexible ways (7; 8). This flexibility empowers network operators to tailor their network services to specific needs.

Secondly, NFV unlocks the potential for network consolidation. Traditionally, diverse network functions relied on dedicated hardware, leading to a complex network environment with a multitude of specialized devices. NFV paves the way for the integration of various network equipment, including servers, switches, and storage, into a centralized data center network (8). This consolidation simplifies network management and potentially reduces operational costs.

In essence, NFV allows for the virtualization of network functions previously confined to specialized hardware, often referred to as ”middleboxes”. These functions are transformed into VNFs,

becoming software-defined and readily deployable. This software-oriented approach streamlines service implementation and fosters greater network agility (8).

2.3.2 NFV MANO

By decoupling network functions from hardware and embracing software-defined VNFs, NFV empowers network operators with a more flexible, efficient, and manageable network infrastructure.

To manage and coordinate these new relationships and operations, a specific architectural framework known as Network Functions Virtualisation Management and Orchestration (NFV-MANO) is essential (9). Figure 2.2 shows the NFV-MANO architecture. NFV-MANO is composed of two primary components: the NFV Orchestrator (NFVO) and the VNF Manager (VNFM). The NFVO is responsible for allocating resources across various virtual infrastructure managers (VIMs) to oversee the life cycle of network services for VNFs. Meanwhile, VIMs manage the NFVI resources, and the VNFM handles the setup and life cycle management of VNFs deployed in virtual machines or containers. (7).

The emergence of network functions virtualization (NFV) necessitates a management and orchestration framework to guide the lifecycle of virtualized resources. NFV Management and Orchestration (NFV-MANO) a framework designed to handle these complex virtualized network operations (9). As stated by (7), NFV-MANO operates through two key components: the NFV Orchestrator (NFVO) and the VNF Manager (VNFM).

The NFVO acts as the central conductor, allocating resources across diverse virtual infrastructure managers (VIMs) (7). This enables it to oversee the entire lifecycle of network services for VNFs, ensuring their smooth operation. Meanwhile, VIMs meticulously manage the underlying NFVI infrastructure, providing the foundation upon which VNFs reside. Finally, the VNFM takes center stage for individual VNFs, handling their setup, lifecycle management, and ensuring their proper functioning within virtual machines or containers (7). In essence, NFV-MANO acts as the invisible hand, coordinating the complex interplay between these components to create a seamless and efficient virtualized network environment.

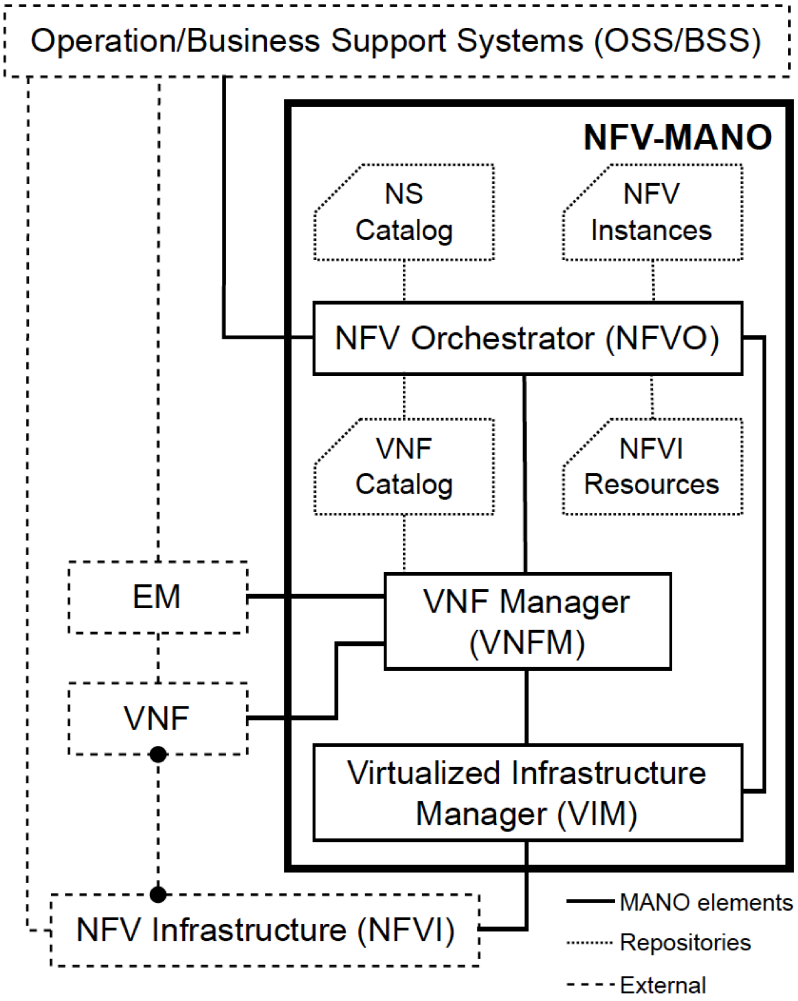

Figure 2.2: NFV-MANO architecture

2.3.3 Procedures

Figure 2.3 illustrates the sequence of interactions among NFV-MANO components involved in deploying a new VNF instance. This process may be part of establishing a complete Network Service (NS), such as the transcoder component of a holographic service. The initial request to deploy the NS can originate from an external Operations Support System (OSS) or a user interface directly integrated with the NFV Orchestrator (NFVO). If a required VNF instance is absent, the NFVO provisions a new one. Our example assumes the VNF package has been previously registered, meaning its corresponding VNF Descriptor (VNFD) resides in the VNF Catalog repository.

As shown in Figure 2.3, the NFVO initiates the VNF instantiation process by instructing the

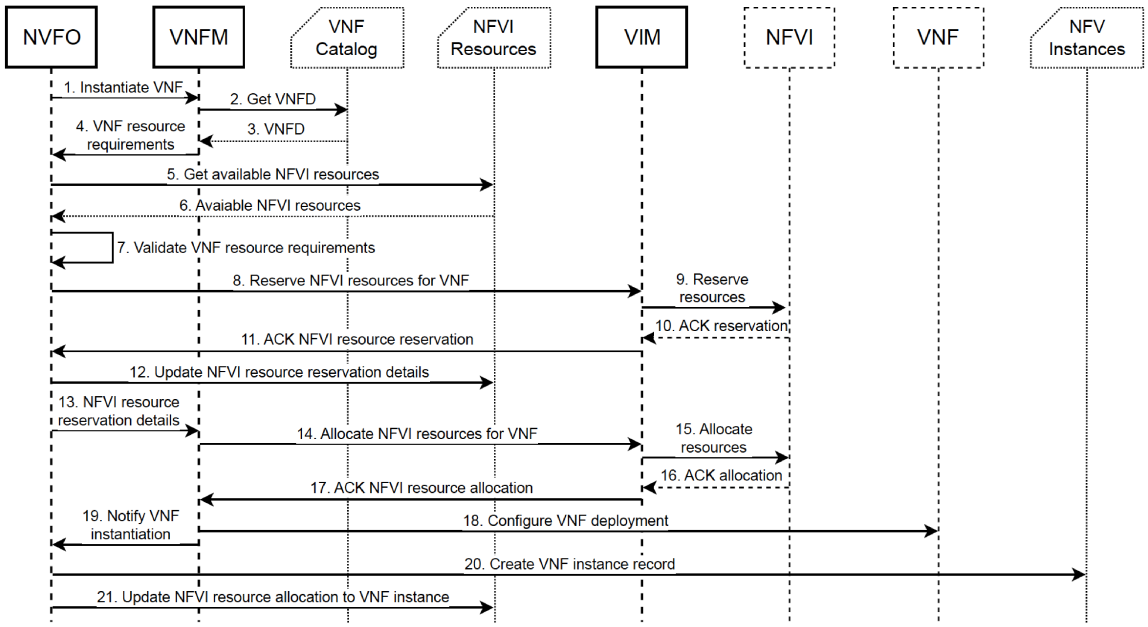

Figure 2.3: Sequence diagram of VNF instantiation in NFV-MANO

VNF Manager (VNFM). The VNFM subsequently retrieves the VNFD from the VNF Catalog to acquire the VNF's deployment and operational specifications. Next, the VNFM seeks NFVO authorization to proceed with VNF instantiation, providing the necessary Network Functions Virtualization Infrastructure (NFVI) resource requirements (compute, storage, network). The NFVO then examines available NFVI resources against the VNF's demands. If sufficient resources exist, the NFVO reserves the required NFVI resources through the Virtual Infrastructure Manager (VIM). Upon resource reservation confirmation from the VIM, the NFVO updates resource status and grants the VNFM permission to proceed with VNF instantiation, providing reservation details.

The VNFM then collaborates with the designated VIM to allocate the reserved NFVI resources for VNF deployment. Once resource allocation is confirmed, the VNFM configures the VNF based on the VNFD specifications. Upon VNF configuration completion, the VNFM informs the NFVO of the successful VNF instantiation. Finally, the NFVO records the VNF's operational and runtime data and updates the NFVI resource mapping to reflect the allocated resources.

2.4 Conclusion

In this chapter, we focused on the major concepts that are relevant to this thesis. The chapter began with a brief discussion on the Holographic Applications couvering the general definition and the requirements of Holographic Applications. Then, we followed by discussing the concept of In-Network computing, its initial step which is SDN, and the benefits of In-Network Computing. The definition of Network Function Vitualization (NFV), its advantages, and the NFV MANO described in brief before concluding the chapter.

Chapter 3

Use cases and State of the art

In order to capture the requirements of a hybrid NFV/INC MANO architecture for provisioning holographic applications, two motivating use cases of holographic applications are first presented, then the requirements are derived from them. Finally, we evaluate and summarize the current state of the arts against the requirements followed by the conclusion.

3.1 Use Case

This section presents an illustrative motivating scenario; a holographic concert scenario. The motivating scenario help to derive the requirements for a hybrid NFV/INC MANO framework solution that enables the provisioning of the holographic applications. These types of application require ultra-low-latency and high bandwidth which leads us to a hybrid management and orchestration solution for their network services. The required network service for provisioning the holographic applications are composed of VNFs and INC components and need a deployment of service chains on the network devices.

3.1.1 Holographic concert scenario

Holographic concerts are a practical application of holographic technology. In this type of performance, artists located in one place can appear as life-sized, interactive holograms to audiences

in different locations. This process involves three main steps: capturing, processing, and transmitting the performer's image. Initially, the performer is recorded from various angles using multiple cameras. These images are then combined and transformed into a hologram. This holographic representation is compressed and sent over a network. At the concert venue, the received data is decoded and reconstructed to create a lifelike holographic image of the performer for the audience to view (1). To transmit a hologram across a network, a three-step process is employed: capturing, rendering, and streaming the target object. The initial stage involves capturing the subject from diverse angles using an array of cameras. The resulting images are then combined and transformed into a holographic representation. This hologram is subsequently compressed and transmitted over the network. Upon reaching the destination, the data is decoded and reconstructed to produce a lifelike holographic image for display (12). There are two primary types of computer-generated holograms: image-based and volumetric. Image-based holograms are constructed from a series of images captured from various perspectives of an object. Light field video (LVF) is an example of this approach (11). While conceptually straightforward, image-based holograms demand substantial storage and network capacity due to the immense quantity of individual images required. For instance, representing an object with images captured every 0.3 degrees for a 30-degree viewing angle and 10-degree tilt would necessitate an array of 3300 images. Nevertheless, as adjacent images within the array exhibit minimal variations, compression techniques can be employed to reduce data volume, albeit at the expense of increased computational effort. The prevailing method for creating holographic representations is through volumetric media, specifically point clouds. This approach involves representing objects as collections of three-dimensional pixels, or voxels, within a conceptual three-dimensional space. Images are then generated dynamically from any viewpoint at the viewing location, allowing for the integration of multiple point cloud objects into a single scene. This process necessitates sophisticated preprocessing and rendering techniques, utilizing camera setups that capture both color and depth information. A key advantage of volumetric media is its high compressibility, as each voxel is transmitted only once, regardless of the number of viewing angles or tilts (12). To establish a standard for volumetric media compression, MPEG has recently adopted V-PCC as a reference encoder (13). This encoder separates point cloud data into two distinct video sequences representing geometry and texture information, respectively, and employs conventional

video compression methods to reduce data volume. Regardless of the holographic media format, data compression and decompression introduce computational overhead, which directly impacts the resulting delay. Consequently, increasing the compression ratio necessitates a trade-off between computational resources and latency on one hand, and network bandwidth and latency on the other (12).

3.2 Requirements

According to the motivating scenario, the identified requirements of ultra-low latency and high bandwidth lead us to a hybrid NFV/INC network to tackle these challenges. In this novel network, we need the deployment of holographic network services that composed of both VNFs and INC components. These components require life cycle management and orchestration. For provisioning the requirements of holographic applications we need to have a hybrid management, a hybrid orchestration, a hybrid infrastructure, and hybrid network service composed of VNFs and INC components. In the next subsections we discuss these needs.

3.2.1 *A hybrid management module*

The hybrid NFV/INC MANO architecture should enable the management of both VNFs and INC components on the network devices. This is needed while for provisioning holographic applications and meeting its requirements both VNFs and INC components are necessary. In the holographic concert senario, for instance, we may need to deploy an encoder and decoder as a VNF and a transcoder as an INC component. As these components have different lifecycle management such as different deployment and configuration process and the existing NFV MANO is not supporting INC lifecycle management, so we need to propose new INC management module.

3.2.2 *A hybrid orchestration module*

In the holographic concert scenario, the holographic application is composed of a chain of network functions for instance encoder as VNF, transoder as INC component, and decoder and renderer as VNF. For deploying these components as one Network Service there should be an orchestrator

to distinguish these function and decide which management module should deploy them on the targeted network. Therefore, a hybrid NFV/INC MANO architecture should be exist to distinguish the targeted VNF and INC components and decide which management module have to deploy that component on network devices. This module also need to maintains the information of the repositories and coordinates the functions and infrastructure managers.

3.2.3 *Hybrid infrastructure and infrastructure manager*

The holographic concert scenario has specific ultra-low latency and high bandwidth requirements. As demonstrated in (3), we need to execute some of the computational tasks on the programmable switches or router to meet these two requirements. With regards to this fact, we require to have a hybride NFV/INC infrastructure in our MANO architecture by adding INC-enabled routers or switches. Also we need to have a hybrid infrastructure manger which able to manage both NFV and INC infrastructures.

3.2.4 *A hybrid network service*

Acording to (3), in the holographic concert scenario, the functions enabling transmission and interactions with holographic data from remote locations across a network are the encoder, transcoder, decoder, and renderer. These components are deployed as VNF on edge servers or INC components on the programmable swiches or routers. In the existing NFV MANO we have the VNF components but there is still need to add the INC components. These network functions are connected to each other as a service chain to produce a holographic network service.

3.3 State of the Art

In the subsequent sections, we evaluate the state of arts and draw summary from it focusing on MANO architectures for hybrid NFV/INC environments.

3.3.1 Works on resource provisioning for holographic applications in hybrid NFV/INC networks

To the best of our knowledge, no work has so far considered the management and orchestration of resources in a hybrid NFV/INC infrastructure. There are actually very few works on NFV/INC infrastructure. (3) present a novel architectural framework that addresses the critical need for efficient resource provisioning for holographic-type applications. The authors try to propose an architecture centers around the concept of INC-enabled slices. These slices provide a dedicated and optimized network environment specifically tailored to meet the demanding requirements of holographic applications. Their motivation for exploring In-Network Computing (INC) stems from its inherent focus on achieving these goals: high bandwidth and ultra-low latency. Network slicing offers a compelling solution for accommodating applications with diverse requirements within a single network. This concept becomes particularly relevant for holographic applications, which will undoubtedly coexist with other services in next-generation networks. However, to cater to the specific needs of holographic experiences, these slices will require INC capabilities. The core of their proposed architecture, is to provide a method for creating slices specifically designed for holographic applications with INC capabilities. These slices cater to the unique needs of holographic experiences. The architecture is built with several key goals in mind. The proposed architecture has some main goals: First, it allows holographic application providers to easily request slices with customizable features, such as bandwidth, latency, and the ability to handle varying numbers of participants, as seen in holographic concerts. Second, the architecture ensures the slice provider can seamlessly integrate these requested slices into the network's existing resources. This requires constant monitoring of network usage to optimize allocation and avoid conflicts. To handle the diverse needs of future networks, the architecture utilizes network slicing. This allows multiple applications, including holographic ones, to coexist within the same network infrastructure, each operating on its own dedicated slice with its specific requirements met. Additionally, the architecture prioritizes incorporating INC functionalities within these slices. This is crucial for achieving the demanding bandwidth and latency requirements necessary for smooth holographic experiences. Finally, the architecture acknowledges the inherent heterogeneity of network environments and is

designed to adapt and function seamlessly even when network components or functionalities differ.

While this study shows that combining Network Function Virtualization (NFV) and In-Network Computing (INC) is a good approach for setting up holographic applications, it does not address how to manage and orchestrate issues of having integrated NFV and INC and the nouvel relationship between new components.

3.3.2 Works on the integration of NFV and INC

While the exploration of novel approaches for NFV deployments continues, it is important to acknowledge their limitations in the context of hybrid scenario. One such example is P4NFV, introduced by (16). This architecture leverages the P4 programming language for implementing Network Functions (NFs), aiming to achieve dynamic data plane reconfiguration.

P4NFV addresses the challenges of managing network functions in dynamic environments through three key design principles. Abstraction simplifies development by shielding NF creators from hardware intricacies, enabling a wide range of NF offerings. Flexibility ensures adaptability through mechanisms like feature upgrades, instance migrations, and runtime reconfigurations to meet evolving network needs. Consistency guarantees reliable performance and NF functionality across diverse hardware and during runtime changes, safeguarding critical stateful NFs. By prioritizing these principles, P4NFV enhances the agility and reliability of network function management.

P4NFV enables dynamic network function reconfiguration while preserving consistency through state management. A prototype evaluation demonstrates the effectiveness of NF relocation for adapting to traffic changes and the importance of data plane pipeline design. P4NFV maintains function liveness and acceptable performance compared to traditional VM solutions, but some hardware reconfigurations may induce packet loss. Future research should focus on lossless reconfiguration mechanisms.

P4NFV's exclusive reliance on programmable devices limits its applicability to hybrid NFV/INC environments. While it can provision individual network functions, P4NFV lacks support for function chaining, essential for holographic applications. Additionally, the architecture's inability to perform network slicing on the programmable data plane restricts its flexibility and efficiency in handling diverse traffic demands.

3.3.3 Works on the implementation of NFV MANO

While there are work on NFV management and orchestration, including official standards, open-source projects, and even efforts to improve upon these standards, none of them take Information Centric Networking (INC) into account. This section will explore these prominent NFV management and orchestration frameworks and explain why they are currently incapable of managing the lifecycle of INC components.

ETSI MANO (19) provides a high-level architecture for NFV management and orchestration, defining functional components and communication channels. While it outlines the system blueprint, it lacks specific implementation details and focuses solely on VNFs and software components, neglecting INC management and orchestration.

Building upon the ETSI MANO standard, Open Source MANO (OSM) offers an open-source implementation for practical use (20). Our proof-of-concept prototype leverages this framework for its functionality. The goal of ETSI OSM (Open Source MANO) is the development of a community-driven production-quality Network Service Orchestrator (NSO) designed for telecom services. This NSO is built for real-world use and can handle the intricate nature of managing telecom services in production environments. ETSI OSM offers several benefits: it accelerates the development of NFV technologies and standards, fosters a wider range of vendors who can create Network Functions, and provides a testing ground to validate how the NSO interacts with other crucial components like commercial network infrastructure and various types of Network Functions. This reference also does not support the management and orchestration of INC components' lifecycle.

While OSM provides a solid foundation for automating network services, Open Network Automation Platform (ONAP) (21) takes things a step further. It's an open-source platform designed for real-time, policy-driven service orchestration and automation. This translates to faster and simpler network service management for providers and developers. ONAP automates the setup and configuration of both physical and virtual network functions, and even supports their entire lifecycle. What sets ONAP apart is its incorporation of big data and artificial intelligence. This adds an extra layer of sophistication by optimizing policy management for network services. The architecture behind ONAP boasts two key components: 1) The ONAP Design Time Framework and 2) The

ONAP Execution Time Framework. The first key component streamlines the creation of blueprints for virtual network functions (VNFs), making it easier to design and deploy them. The second key component employs metadata-driven modules to automate the configuration and instantiation of VNFs, further simplifying the process. Contrary to these efforts, ONAP also does not offers a solution for managing and orchestrating neither INC components nor hybrid components that combine both VNFs and INC components.

3.3.4 Works on the extension of NFV MANO

In this among we revised the examples of an architectures which is an extension of ETSI MANO. A first example of a proposal for enhancing / extending is MANOaaS (22). ETSI MANO adopts a centralized approach to management and orchestration. This leads to significant performance issues. This paper proposes MANOaaS, a novel architecture that abstracts the centralized NFV-MANO framework into distributed tenant MANO (t-MANO) instances. This empowers tenants with greater control over their allocated network slice resources. 5G networks require managing diverse services with stringent performance demands for various industries. Network slicing enables provisioning isolated virtual networks over shared infrastructure. A robust Network Slice Management and Orchestration (MANO) system is crucial. The current, centralized ETSI NFV-MANO framework might face performance and management challenges. It introduces MANOaaS, which breaks down the centralized MANO stack to enable network slicing. The architecture includes distributed t-MANO instances and differentiated management agreements between infrastructure providers and tenants. The authors believe MANOaaS is a significant step towards full network slicing isolation with performance and management benefits.

Another example is LightMANO (23). It also tackles the issues inherent to the centralized approach followed by ETSI MANO. While the ETSI MANO framework provides a reference architecture for NFV, challenges remain for deploying NFV in distributed edge environments like those required for 5G and Industrial IoT. LightMANO introduces a lightweight platform for managing and orchestrating network services at the network edge. It converges SDN and NFV functionalities and leverages containerization and programmable packet processing technologies. The authors demonstrate its effectiveness through a content caching use case and plan to further improve its

runtime system and automation capabilities.

3.3.5 Summary of the State of the Art of the Hybrid NFV/INC MANO Architectures for Holographic Applications

Figure 3.1 shows the summary of meeting the requirements against the proposed frameworks for hybrid NFV/INC MANO architectures. A "✓" means that the corresponding requirement was met by the proposed framework. A "x" means that the framework failed to meet the requirement. The paper (3) is not included in this figure as it is not proposed any management and orchestration framework.

Scope	References & papers	Hybrid Management		Hybrid Orchestration		Hybrid Infrastructure & Infrastructure Manager				Hybrid Network Service
		VNFM	INCM	NFVO	INCO	VIM	NFVI	INCI	IIM	
Hybrid NFV/INC MANO	P4NFV [10]	✓	✓	✓	✓	✓	✓	✓	✓	x
	NFV-MANO [5]	✓	x	✓	x	✓	✓	x	x	x
	OSM [7]	✓	x	✓	x	✓	✓	x	x	x
	ONAP [8]	✓	x	✓	x	✓	✓	x	x	x
	ManoaaS [22]	✓	x	✓	x	✓	✓	x	x	x
	LightMANO [18]	✓	x	✓	x	✓	✓	x	x	x

Figure 3.1: Summary of the related works for hybrid NFV/INC MANO architecture

3.4 Conclusion

In this chapter, we first presented two motivating use cases. Using these use cases, we derived the requirements for the hybrid NFV/INC MANO architecture. Then, we reviewed the current state of the arts against our derived requirements. For each paper, we showed the requirements which are met and the ones which are not met. Apparently, none of the states of art was able to fulfill all of the requirements. Finally, we presented a summary table showing which of the requirements were fulfilled by which state of the art.

In the next chapter, we present our proposed architecture, describe the associated components, and functionalities in detail.

Chapter 4

Proposed Architecture

In this chapter, we focus on our proposed hybrid NFV/INC MANO architecture for holographic applications. First we provide a description of our designed architecture and each module in detail. The required interfaces are also discussed to show the communication way between necessary modules. Then the main procedures about deployment of VNF and INC component are explained with the help of presenting of the illustrative sequence diagrams for the interaction between involved modules within the hybrid NFV/INC MANO. Finally, we conclude this chapter by a brief evaluation of proposed architecture against the requirements followed by a conclusion section.

4.1 Proposed Architecture

This section outlines an extension to the existing NFV MANO architecture to accommodate the lifecycle management of Network Services (NSs) composed of multiple interconnected Virtual Network Functions (VNFs) and In-Network Functions (INFs) operating within a hybrid NFV/INC infrastructure (NII). INFs represent networking applications or functionalities executed directly on INC-enabled devices, such as programmable switches or SmartNICs. As illustrated in Figure 4.1, the proposed NFV/INC MANO architecture introduces an INF Manager (INFM) and modifies two existing components: the NFV/INC Infrastructure Manager (NIIM) and the NFV/INC Orchestrator (NIO), which replace the VIM and NFVO, respectively. Furthermore, our architecture includes an INF Catalog and modifies three additional repositories: the NS Catalog, NS Instances, and NII

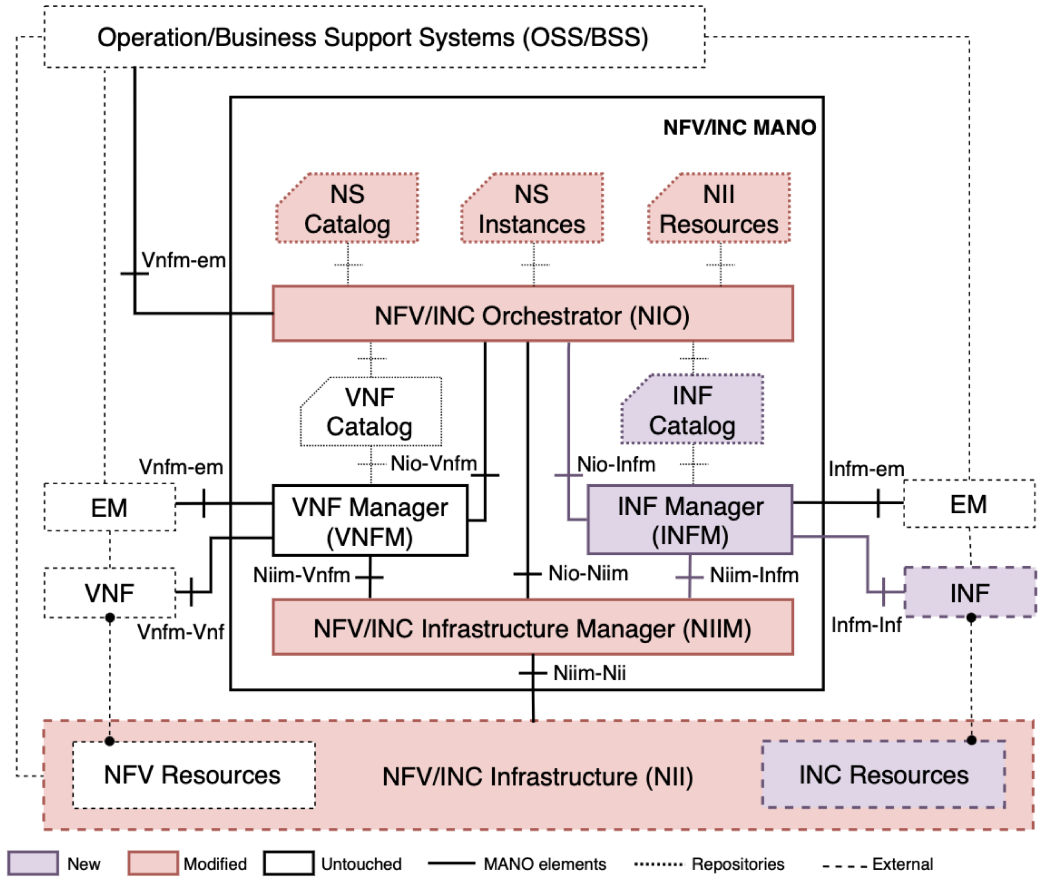

Figure 4.1: NFV/INC-MANO architecture

Resources, replacing the NFV Instances and NFVI Resources, respectively. Also, the required Nio-Infm, Niim-Infm, and Infm-Inf interfaces are discussed to show the communication way between necessary modules.

In the following subsections we describe each developed and modified component in three different groups of untouched, modified, and new modules and components.

4.1.1 The untouched modules

4.1.1.1 *Operation/Business Support System (OSS/BSS)*

OSS/BSS are the systems that handle the business and operational support functions, but they are not directly part of the NFV-MANO framework. OSS/BSS are expected to have information exchanges with functional blocks in the NFV-MANO architectural framework. They work together

with the NFV-MANO system to manage older network technologies. OSS/BSS have a complete view of all services provided by both new and old network equipment.

4.1.1.2 *VNF Manager (VNFM)*

The VNF Manager (VNFM) is responsible for managing the entire lifecycle of individual VNF instances. It obtains the necessary information about each VNF from its corresponding VNF Descriptor, which is stored in the VNF Catalog. To manage the VNF's resources, the VNFM works closely with the VIM. Additionally, it may interact with an external Element Management system to handle various management tasks related to the VNF.

4.1.1.3 *NFV Resources*

NFV Resources refers to the physical and software components that host VNFs. This includes servers and virtualization technologies like hypervisors and containers. NFV Resources also includes network components like SDN controllers that allow the NIIM to connect and link multiple VNFs together to create network services.

4.1.1.4 *VNF Catalog*

The VNF Catalog is a central repository for storing information about VNFs. It contains detailed descriptions of VNFs, including software components and configuration details. Both the NIO and VNFM can access this catalog to obtain information needed for managing and deploying VNFs.

4.1.1.5 *Element Management (EM)*

Element Management (EM) is responsible for overseeing the various aspects of a Virtual Network Function (VNF) and In-Network Function (INF). This includes managing its configuration, identifying and resolving faults, tracking usage, measuring performance, and ensuring security. The EM can work together with the VNF and INF Manager to manage the VNF's and INF's resources within the virtualized environment.

4.1.2 The modified modules

4.1.2.1 *NFV/INC Infrastructure Manager (NIIM) and NFV/INC Infrastructure (NII)*

The NIIM expands upon the VIM by incorporating the allocation, release, and monitoring of INC resources within the NII. NII, comprising the underlying infrastructure of programmable network devices and servers, provides the physical resources for executing both VNFs and INFs. INC resources comprise tables, memory, and processing capabilities of programmable data plane devices, such as switches and SmartNICs, which execute the deployed INFs. Consequently, NIIM oversees the hybrid NFV and INC infrastructure, assigning VNFs and INFs to appropriate NII resources. Similar to NFVI, NII encompasses network controllers, such as SDN controllers, which empower NIIM to interconnect and sequence multiple VNFs and INFs for the deployment of hybrid NSs.

4.1.2.2 *NFV/INC Orchestrator (NIO)*

The NIO expands upon the NFVO by incorporating the onboarding and lifecycle management of hybrid network services (NSs), which consist of interconnected virtual network functions (VNFs) and In-Network Functions (INFs) forming a service chain topology. Similar to the NFVO, the NIO maintains information within the various repositories (INF Catalog, VNF Catalog, NS Catalog, NS Instances, and NII Resources), coordinates the functions and infrastructure managers (VNFM, INFM, and NIIM), and may interact with external OSS/BSS platforms. The NIO stores INF descriptors (INFDs) of onboarded INFs in the INF Catalog repository while maintaining VNFs in the VNF Catalog repository. Within the NS Catalog repository, network service descriptors (NSDs) of onboarded hybrid NSs include references to INFs and their connections to other VNFs and INFs within the service chain topology. The NS Instances repository extends upon the NFV Instances by recording operational data and runtime information of instantiated INFs and hybrid NSs. The NII Resources repository expands upon the NFVI Resources by supporting allocation and management information for INC Resources within the NII.

4.1.2.3 *NS Catalog*

The NS Catalog stores information about hybrid Network Services (NSs). These descriptions include details about how Network Instances (INFs) are incorporated into the service and how they connect with other VNFs and INFs within the service's structure.

4.1.2.4 *NS Instances*

The NS Instances repository is an extension of the NFV Instances repository. It specifically tracks and stores operational data and runtime information related to deployed Network Instances (INFs) and hybrid Network Services (NSs).

4.1.2.5 *NII Resources*

The NII Resources repository is an expansion of the NFVI Resources repository. It specifically handles the allocation and management of resources within the NII environment.

4.1.3 The new modules

4.1.3.1 *INF Manager (INFM)*

The INFM is responsible for managing the entire lifecycle of INF instances, including their installation, configuration, updating, and removal. It retrieves the INF Descriptor (INFD) from the INF Catalog repository to acquire deployment and operational details for the corresponding INF. INFDs encompass specifications and templates essential for INF management, such as programmable architectures and matching tables. During the INF onboarding process, the NIO stores INFDs within the INF Catalog repository. The INFM collaborates with the NIIM to allocate and release INC resources by requesting the installation, updating, and removal of INFs on programmable data plane devices. For INF configuration, the INFM can interact directly with INF instances or through an external Element Manager (EM). The EM also facilitates FCAPS management operations on INFs for the INFM. Additionally, the INFM can subscribe to and receive monitoring and configuration reports from the NIIM regarding INC resources (tables, memory, and processing cycles) utilized by a specific INF.

4.1.3.2 *INF Catalog*

The INF Catalog is a central repository for storing information about INFs. It contains detailed descriptions of INFs, including software components and configuration details. Both the NIO and INFM can access this catalog to obtain information needed for managing and deploying INFs. The NIO stors INFDs in the INF Catalog repository when new INF instantiated. The INFM retrieves the INFD from the INF Catalog to get the necessary details for deploying and operating the INFs.

4.1.3.3 *NIO-INFM interface (Nio-Infm)*

This interface serves as the communication link between the NFV/INC Orchestrator (NIO) and the INF Manager (INFM) , facilitating directives for the deployment, management, and termination of INC instances, coordinating in-network resource allocation, enforcing policies, and relaying performance metrics to ensure cohesive and compliant operation of in-network computing functions within the orchestrated network services.

4.1.3.4 *NIIM-INFM interface (Niim-Infm)*

The Niim-Infm interface enables exchanging information between NFV/INC Infrastructure Manager (NIIM) and INFM . This interfaces provides managing and monitoring services such as performance management, fault management, and change notification for virtualized resources (cloud, network, storage).

4.1.3.5 *INFM-INF interface (Infm-Inf)*

This interface represents the link through which the INFM configures, manages, and monitors the lifecycle of INFs, ensuring that these embedded computational elements within the network infrastructure operate in harmony with the orchestrated network services.

4.2 Procedures

The proposed architecture includes the following VNFs and INC components lifecycle procedures: network function deployment and deletion. We describe the deployment and deletion procedures for the both VNF and INF in the subsequent subsections below. First we start by describing the deployment of VNF followed by the deletion procedure of a VNF using their sequence diagram. Then we move forward with describing the deploying and deleting procedures of an INF by using their sequence diagram. While the lack of documentation and CLI commands presents a challenge, future work could explore alternative approaches or the development of necessary tools to facilitate OSM migration.

4.2.1 Deployment of a VNF network function

As an example scenario, figure 4.2 illustrates the procedural and interactive elements within the NFV-MANO framework requisite for the deployment of a novel VNF instance. It is imperative to note that the deployment of a VNF may constitute a component of an instantiation of a complete NS, such as the transcoder function of a holographic service. The request for deploying the network service can originate from an external orchestration and support system or a user interface directly integrated within the NFV orchestrator. In the event of a non-existent VNF instance comprising the network service, the NFV orchestrator initiates the deployment of a new VNF instance as necessary. Moreover, this exemplar posits the prior onboarding of the VNF package by the NFV orchestrator, thereby ensuring the presence of the corresponding VNF descriptor within the VNF catalog repository.

Figure 4.2 illustrates the sequential interactions among NFVO, VNFM, VNF Catalog, and VIM for VNF instantiation. Initiated by the NFVO (1), a request is forwarded to the VNFM, accompanied by pertinent VNF details. Upon receiving this request, (2-3) the VNFM retrieves the requisite VNF descriptor from the VNF Catalog repository to ascertain the VNF's deployment and operational specifications. Subsequently, (4) the VNFM requests authorization from the NFVO to proceed with VNF instantiation, concurrently communicating the necessary NFVI resource demands (computational, storage, and networking). (5-7) The NFVO then evaluates the availability of these resources

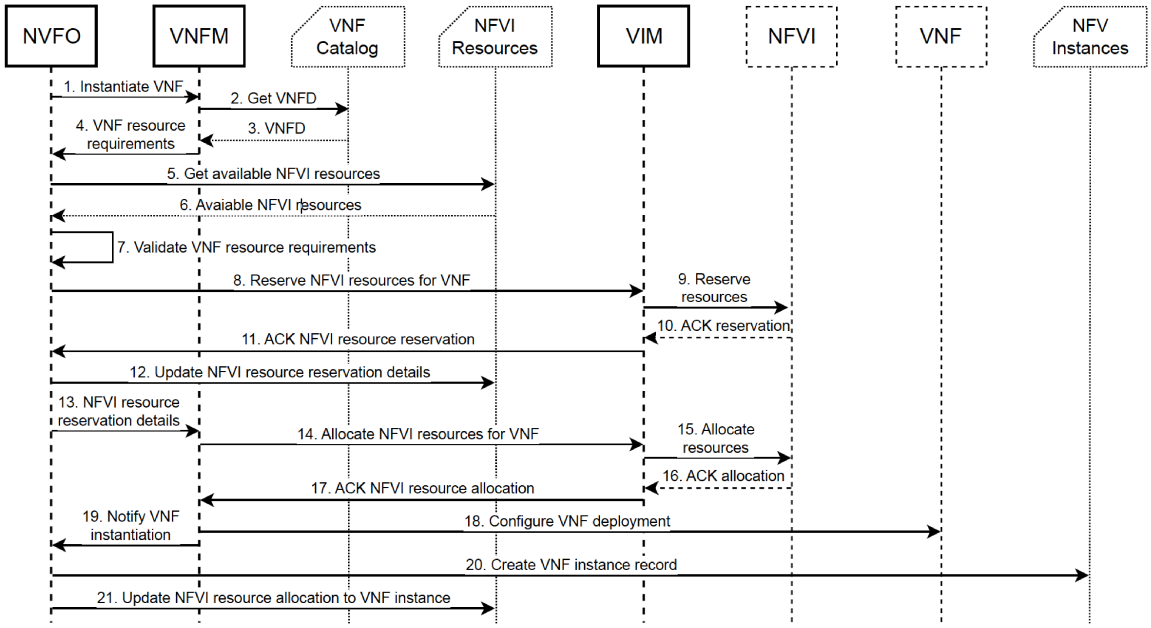

Figure 4.2: Sequence diagram of VNF instantiation in NFV-MANO

within the NFVI Resources repository against the VNFM's requirements. Upon confirmation of sufficient resources, (8) the NFVO directs the VIM to reserve the specified NFVI resources for VNF instantiation. (9-11) The VIM subsequently allocates the requested resources within the NFVI environment and awaits acknowledgment before transmitting a confirmation to the NFVO.

Following this, (12-13) the NFVO modifies the NFVI Resources repository by designating the VNF's resources as reserved and authorizes the VNFM to commence VNF instantiation, providing essential NFVI resource reservation information (such as VIM identifier and virtualization parameters). Consequently, (14-15) the VNFM collaborates with the designated VIM to allocate the reserved NFVI resources for VNF instantiation. (16-17) Upon mutual confirmation of successful resource allocation by the VIM and VNFM, (18) the VNFM directly interacts with the allocated resources to configure VNF deployment according to the specifications outlined in the VNF descriptor. Upon completion of VNF configuration, (19) the VNFM informs the NFVO of the newly instantiated VNF. (20- 21) The NFVO subsequently records the operational and runtime data of the VNF in the NFV Instances repository and updates the NFVI Resources repository by associating the VNF instance with the allocated resources.

4.2.2 Deletion of a VNF network function

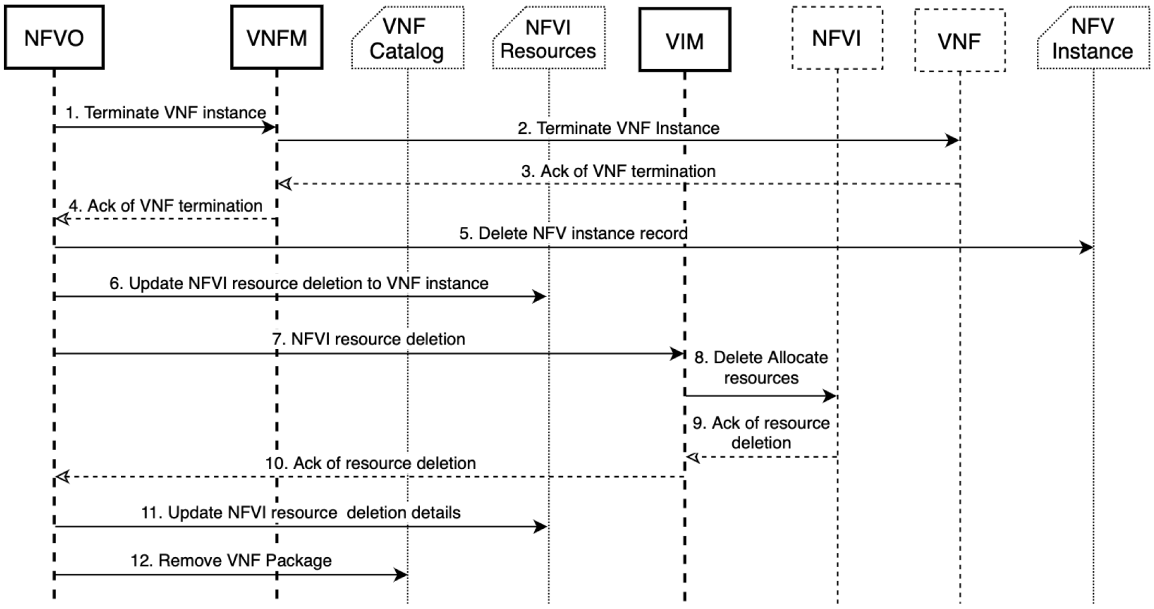

Figure 4.3: Sequence diagram of VNF termination and deletion in NFV-MANO

Figure 4.3 illustrates the sequential interactions among NFVO, VNFM, VNF Catalog, and VIM for VNF termination and deletion. Initiated by the NFVO (1), a request is forwarded to the VNFM, accompanied by pertinent VNF details. Upon receving this request, VNFM terminate the targeted VNF (2,3). Once the VNF is terminated, VNF Manager acknowledges the completion of the VNF termination back to the NFVO (4). Subsequently, (5) the NFVO delete the VNF instance record from the NFV instance repository, concurrently update NFVI resource deletion to VNF instance in the NFVI Resources repository (6). (7) The NFVO then request VIM to release NFVI resources (compute, storage and network) used by the various VDUs of the VNF instance. Then VIM (8) delete allocate resources which includes internal connectivity network and the compute (VMs) and storage resources of the various VDUs of the VNF instance. Upon receiving the acknowledge of the completion of the NFVI release resources (9), the VIM send an acknowledge of the completion of the NFVI release resources to the NFVO (10). Followed by that, (11- 12) the NFVO subsequently update the NFVI resource deletion details in the NFVI Resources repository and remove VNF Package from VNF Catalog.

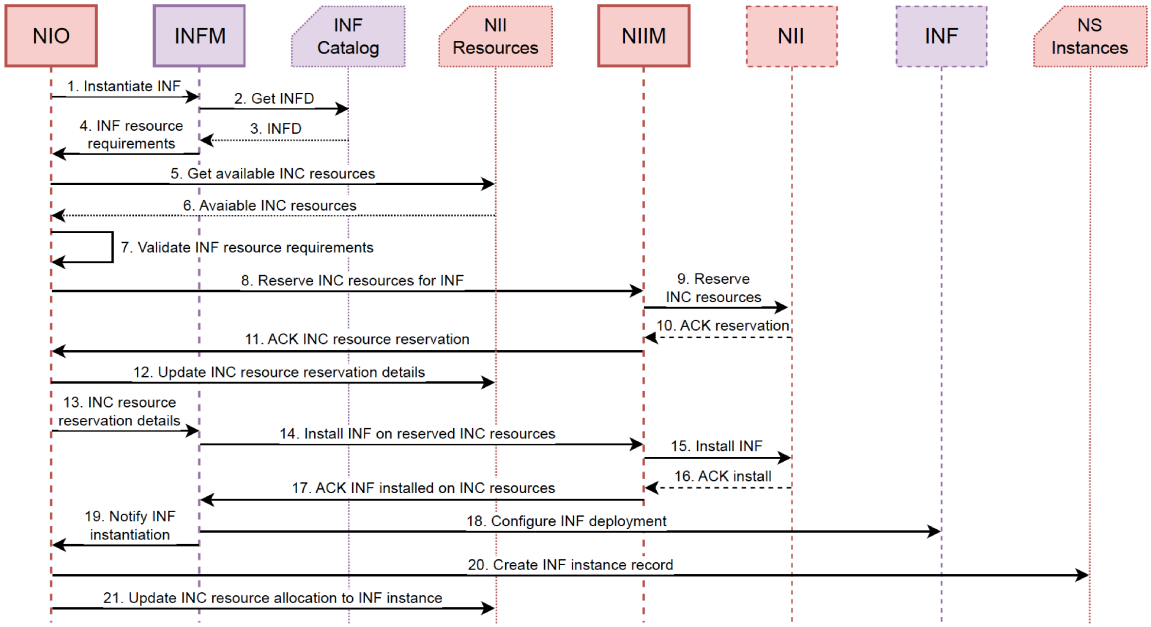

Figure 4.4: Sequence diagram of INF instantiation in NFV/INC-MANO

4.2.3 Deployment of an INC network function

Figure 4.4 depicts the deployment process of a network function, specifically a transcoder, instantiated as an In-Network Function (INF) within the proposed NFV/INC-MANO architecture. This INF may be a component of a hybrid network service (NS) that incorporates multiple interconnected Virtual Network Functions (VNFs) and INFs. For instance, a hybrid holographic NS might consist of two VNFs (encoder and renderer) and two INFs (transcoder and decoder). Upon instantiating the hybrid NS, the NFV/INC Orchestrator (NIO) creates new instances of any previously undeployed VNFs and INFs. The VNF instantiation process aligns with the depiction in Figure 4.2, with necessary adjustments to functional entities (NIO, NIIM, and NII replacing NFVO, VIM, and NFVI, respectively) and repositories (NII Resources and NS Instances replacing NFVI Resources and NFV Instances, respectively). This explanation primarily focuses on INF instantiation, assuming that the INF has already been onboarded by the NIO, ensuring the corresponding INF descriptor is available in the INF Catalog repository.

As illustrated in Figure 4.4, the INF instantiation process commences with the NFV/INC Orchestrator (NIO) requesting the INF Manager (INFM) to initiate the creation of a specific INF (1).

The INFM subsequently accesses the INF Catalog repository to acquire the deployment and operational specifications outlined in the INF descriptor (2,3). Following this, (4) the INFM submits the INF's resource requirements, encompassing tables, memory, and processing cycles, to the NIO as a prerequisite for proceeding with the INF instantiation. In response, (5,6) the NIO retrieves available INC resource information from the NII Resources repository and conducts a comparative analysis to determine the feasibility of the instantiation. Upon confirmation that the available INC resources meet the INF's requirements (7), the NIO collaborates with the NFV/INC Infrastructure Manager (NIIM) to secure a reservation of the necessary INC resources from the NFV/INC Infrastructure (NII) (8,9). Once the NIO receives acknowledgment of successful INC resource reservation from both NII and NIIM (10,11), it updates the INC resource information within the NII Resources repository (12).

Subsequently, (13) the NIO communicates the INC resource reservation details, including NIIM and switch identifiers, to the INFM as authorization to proceed with INF instantiation. In response, (14) the INFM interacts with the designated NIIM to transmit the INF deployment specifications outlined in its INF descriptor and requests the installation of the INF on the reserved INC resources, such as a programmable switch. The NIIM proceeds to install the requested INF on the allocated INC resources (15), awaiting installation confirmation from the NII before acknowledging the INFM (16). The INFM then leverages the operational specifications from the INF descriptor to directly configure the INF deployment, including the configuration of matching tables (18). Upon completing the INF configuration, (19) the INFM informs the NIO of the successful creation of the new INF instance. Finally, (20,21) the NIO establishes a new record in the NS Instances repository, documenting the operational and runtime details of the deployed INF, and updates the status of the utilized INC resources within the NII Resources repository.

4.2.4 Deletion of a INF network function

Figure 4.5 illustrates the sequential interactions among NIO, INFM, INF Catalog, and NIIM for INF termination and deletion. Initiated by the NIO (1), a request is forwarded to the INFM, accompanied by pertinent INF details. Upon receiving this request, INFM terminate the targeted

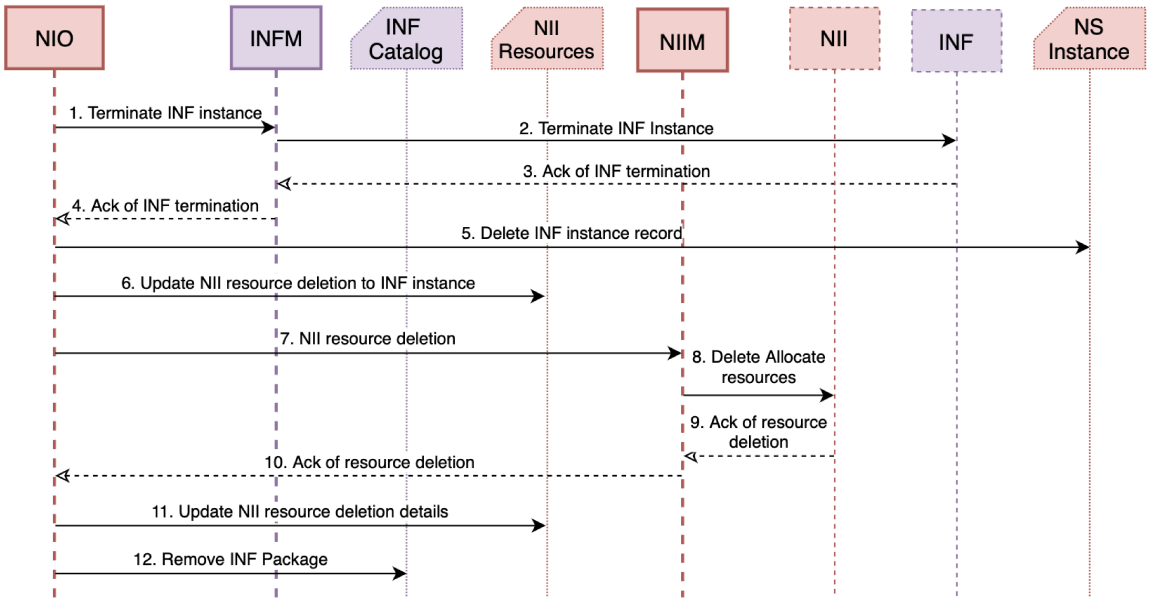

Figure 4.5: Sequence diagram of INF termination and deletion in NFV/INC-MANO

INF (2,3). Once the INF is terminated, INF Manager acknowledges the completion of the INF termination back to the NIO (4). Subsequently, (5) the NIO delete the INF instance record from the NS instance repository, concurrently update NII resource deletion to INF instance in the NII Resources repository (6). The NIO then request NIIM (7) to release NII resources used by INF instance. Then NIIM (8) delete allocate resources. Upon receiving the acknowledge of the completion of the NII release resources (9), the NIIM sends an acknowledge of the completion of the NII release resources to the NIO (10). Followed by that, (11- 12) the NIO subsequently update the NII resource deletion details in the NII Resources repository and remove INF Package from INF Catalog.

4.3 Evaluation of Proposed Architecture against the Requirements

The requirements derived from the motivating use-cases should be satisfied by the proposed architecture. In fact, the proposed architecture fully meets the requirement.

The proposed NFV/INC MANO architecture has a A hybrid management modules including VNFM and INFM for managing lifecycle of both VNFs and INC components. Hence, the first requirement regarding a hybrid NFV/INC MANO architecture is fully met.

The architecture includes the appropriate hybrid orchestration module that can distinguish the

targeted network function and decide which management module have to deploy that component on network devices as well as managing network services lifecycle including deployment, chaining, execution, monitoring, and migration. Moreover, the required repositories and catalogs introduced for maintaining information of network services. Hence, the proposed architecture supports the need for a hybrid NFV/INC orchestration.

The proposed NFV/INC MANO architecture includes a hybrid NFV/INC infrastructure manager (NIIM) for allocation, release, and monitoring of INC resources within the NII. The need for a hybrid infrastructure is also covered by adding INC Resources to the architecture. Therefore, the proposed architecture fulfills requirements regarding extention of infrastructure manager.

Finally, a hybrid network service is designed to manage the VNF and INC components mentioned in the hybrid architecture.

4.4 Conclusion

In this chapter, we presented our proposed architecture, explained the module's functionality in detail along with interfaces for communication between necessary modules. Afterward, we provided an illustrative sequence diagrams, showing how an INC components of the proposed architecture can be deployed on an INC-enabled switches or routers. Finally, we justified how the proposed architecture was able to meet the previously derived requirements from the motivating use-cases.

In the following chapter, we will present an implemented prototype of the proposed architecture followed by the results and conclusion from it.

Chapter 5

Validation of the Architecture

In this chapter, we start by an overview on prototype architecture including first used softwares, a general description of implemented prototype, a description on implemented scenario, a set of Interfaces used in prototype, followed by programming languages we used. We also describe the experimental setup in the last part of same section. Finally, we discuss the performace evaluation by mentioning the performance metric, test cases, and results of various experiments and analyze them accordingly. We finally conclude the chapter by summarizing it.

5.1 Prototype Architecture Overview

In this section we first present the implemented scenario followed by a description of how prototype operates along with the prototype architecture and the set of Interfaces used in prototype to provision the communication between necessary modules. At the end, a brief description of the hardware and software used for implementing the prototype is given as well as the experimental setup in the last subsection.

5.1.1 Used Softwares

In this subsection, we describe the softwares used for implementing the prototype NFV/INC MANO framework for holographic applications.

5.1.1.1 OSM

Open Source MANO (OSM) is an open-source platform designed for the orchestration and management of Network Functions Virtualization (NFV). It automates the lifecycle management of Virtual Network Functions (VNFs). OSM utilizes a blueprint approach. It leverages standardized Network Service Descriptors (NSDs) based on the YANG Model. OSM supports YANG-based data modeling and integrates with multiple VIM vendors, including OpenStack, VMware, and Azure. These NSDs, typically written in YAML format, define the deployment, configuration, and lifecycle management of VNFs within an OSM environment. OSM platform consists of different modules while North Bound Interface (NBI), Life Cycle Management (LCM), and Resource Orchestrator (RO) are three main components. Modules run in separate Docker containers. One of the key strengths of OSM is its multi-vendor support. It can integrate with various VNFs and VIMs, offering flexibility in network infrastructure management.

5.1.1.2 MicroK8s

MicroK8s is a simplified, open-source platform that helps for deployment, scaling, and management of container-based applications. It provides the functionality of core Kubernetes components, but in a smaller package, and can be used on a single computer or a complex network of machines.

5.1.1.3 OpenLDAP

OpenLDAP is an open-source implementation of the Lightweight Directory Access Protocol (LDAP) standard. It provides a directory service that stores, organizes, and retrieves information about users, computers, networks, and other resources.

5.1.1.4 Mininet

Mininet is a free software tool for emulating computer networks on a single machine. It allows users to build virtual networks, test network applications and protocols, and develop high-speed software. Unlike real networks, Mininet simplifies the process by using virtual network links and

servers that can be controlled programmatically. This makes it easier to experiment with and simulate real-world networking protocols in a controlled environment. Mininet is a flexible and portable tool that can be controlled using Python scripts. One of the main purposes of Mininet is to support the development of software-defined networks (SDN) and network function virtualization (NFV).

5.1.1.5 BMv2 Switch

The P4.org organization has created an open-source software switch known as Behavioral Model version 2 (BMv2). This software switch functions specifically as a target for P4 programs. In essence, P4 programs can be compiled and executed on BMv2, allowing them to determine how the switch processes network packets.

5.1.1.6 Flask

Flask is a lightweight WSGI web application framework written in python and is considered a micro-framework since it does not require any specific libraries. It offers a core set of features and lets add more functionality using third-party libraries as needed.

5.1.2 Description of Implemented Prototype

To demonstrate the implementation of our proposed NFV/INC MANO architecture, we developed a prototype system. As depicted in Figure 5.1, our prototype builds upon the Open Source MANO (OSM) (20) platform, a production-ready implementation adhering to the ETSI NFV MANO (19) standards. The OSM platform comprises three primary modules: a Northbound Interface (NBI) offering Command Line Interface (CLI) and RESTful services for user interaction, a Life Cycle Manager (LCM) overseeing VNF lifecycle management, and a Resource Orchestrator (RO) interfacing with various VIMs through dedicated plugins. These components collaborate and exchange information through a message bus and shared storage services.

Figure 5.1 illustrates the components added, modified, and reused in our prototype. New or modified elements are highlighted by background color. For the INC (in purpule), we introduce a P4 Manager and a P4 plugin which are the new modules added. The P4 Manager implements the INC infrastructure manager within the NFV/INC Infrastructure Manager (NIIM) modules, enabling

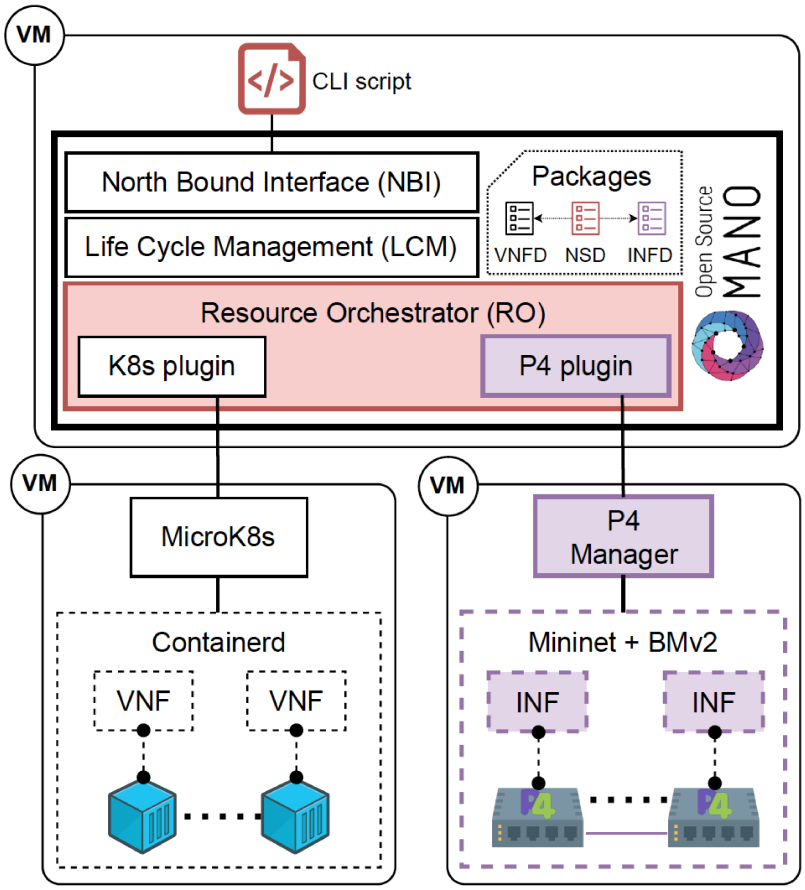

Figure 5.1: Prototype implementation

the deployment of P4 programs (INFs) on P4-programmable switches (INC resources in NFV/INC Infrastructure (NII)). The P4 plugin operates within OSM's Resource Orchestrator (RO) to interact with the P4 Manager via RESTful services. To simulate P4-programmable switches in NFV/INC Infrastructure (NII), we utilize Mininet with the BMv2 software switch (24). For the Network Functions Virtualization (NFV) domain (in black), we reused the existing OSM utilities for Kubernetes (K8s) (25). OSM provides YANG elements in the VNF Descriptor (VNFD) to describe K8s-based VNFs and a plugin to interact with K8s clusters for VNF deployment. Our prototype employs MicroK8s (26) to deploy a lightweight K8s cluster emulating the NFV infrastructure manager in NFV/INC Infrastructure Manager (NIIM), facilitating the deployment of K8s-based VNFs on containerized NFV resources in NFV/INC Infrastructure (NII). Finally, we modified the RO module in OSM (in red) to integrate the P4 plugin and its communication with the P4 Manager, enabling INF

deployment support.

5.1.3 Implemented Scenario

To validate our prototype's capacity to instantiate VNFs and INFs on container-based NFV resources and P4-programmable switches respectively, mimicking a hybrid NS deployment, we conducted a proof-of-concept. The implemented scenario is the holographic concert use-case, presented in section 3.1.1 that the performer located in a different location from the audiences. In this scenario a hybrid Holographic service is provisioned to deploy on network devices. This hybrid NS made up of encoder, transcoder, decoder and renderer components wihch composed of VNFs and INFs deployed on the edge servers or programable-switches. Encoder, decoder and renderer are assumed as VNF, while transcoder is implemented as INF.

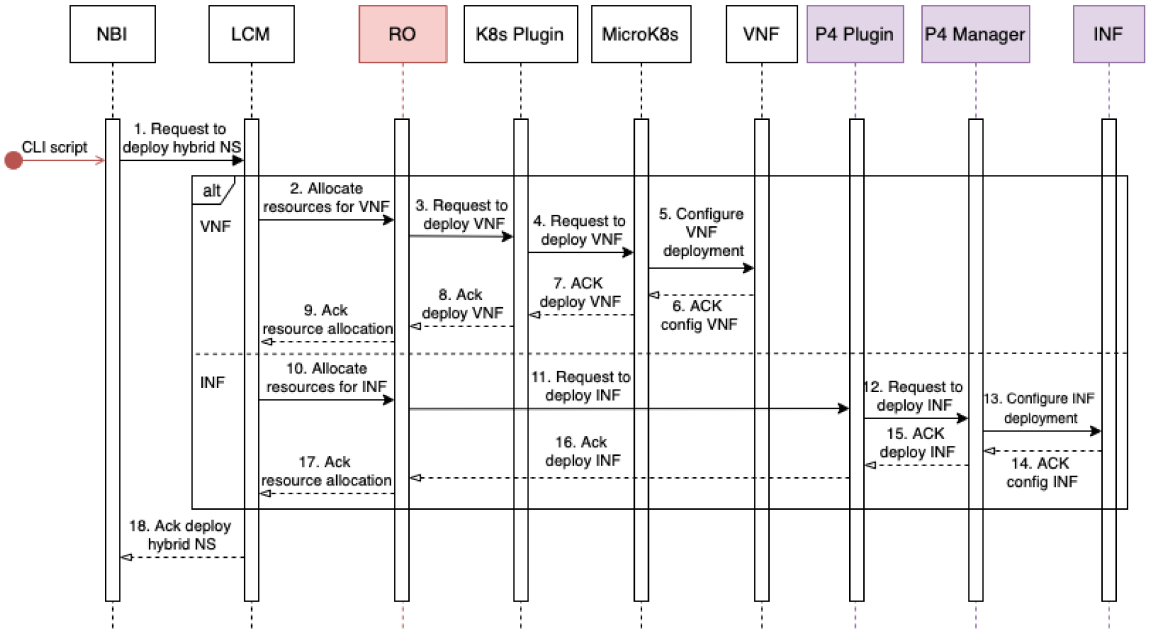

Figure 5.2: Sequence diagram of deploying a hybrid Network Service

Figure 5.2 illustrate the process of deploying a hybrid NS using the implemented prototype. In this scenario we deploy a hybrid NS which are pre-onboarded or installed NS, VNF, and INF packages within OSM through their respective descriptors (NSD, VNFD, and INFD). Typically, an external NFV/INC placement algorithm (3) would interact with the NBI to deploy the hybrid NS at specific network locations (1). We emulated this by executing a script generating NBI CLI

commands. Upon receiving the NS deployment request, NBI delegate the deployment request to the Life Cycle Manager (LCM) which identifies VNF and INF components from the NSD (2). While we onboarded NS and VNF packages using OSM's NBI CLI, the INF descriptor was embedded within the P4 plugin's code to simulate a hybrid NS. For each component, the LCM send a request to the Resource Orchestrator (RO) to allocate resources of VNF (3) and INF (11). Then RO collaborates with it's K8s plugin (4) and P4 plugin (12) to instantiate the function using the corresponding descriptor. VNF instantiation leverages the K8s plugin to deploy a K8s-based VNF on MicroK8s (5) and configuring VNF deployment (6), while INF instantiation employs the P4 plugin to install a P4 program on a designated P4 switch by requesting to P4 Manager (13,14). An acknowledgement (ACK) packet is sent back for each request until we reach to the placement algorithm (7-10,15-20).

5.1.4 Interfaces

The fundamental design principle governing the interactions between the various modules and domains is the adoption of the Representational State Transfer (REST) architectural style. The selection of REST is attributed to its lightweight, standards-based nature and adaptability to diverse data formats (e.g., plain text, XML, and JSON), enabling generic Application Programming Interface (API) descriptions. All interfaces offer Create, Read, Update, and Delete (CRUD) operations.

5.1.4.1 Interface for deploying an INC component

The INC deployment Interface allows the NFV/INC MANO modules can communicate with programmable switches for the INC components lifecycle management and orchestration. Table 5.1 shows the REST API for an INC component deployment. The REST API endpoint for deploying an INC component (a P4 program) is a POST request sent to the URI http://{host_IP}/switches/{switch_id}/programs. The switch_id parameter specifies the target switch for installation and execution of the P4 program. Upon successful deployment, the server returns a 201 Created status code with a JSON response containing the program_id.

This API streamlines the process of installing and running P4 programs on network switches. By providing the switch ID and the P4 program file, clients can offload the deployment logic to the server. The server handles the installation and execution phases, ensuring proper configuration and

operation of the P4 program on the target switch. The returned program ID serves as a reference for future management or removal of the deployed component. This API adheres to REST principles by using clear URIs, standard HTTP methods, and well-defined request and response formats.

REST Resource	Operation	HTTP Action & Resource URI	Request Body Parameters	Response code	Response content
Switch Programs	Deploy INC component (P4 program)	POST: /switches/switch_id/ programs	None	201 (created)	program_id (string)

Table 5.1: Example of POST API exposed by the NIO to the NIIM

5.1.4.2 Interface for deleting an INC component

Table 5.2 shows the REST API for deleting an INC component from a target switch. The REST API endpoint for deleting an INC component (a P4 program) from a switch is a DELETE request sent to the URI http://{host_IP}/switches/{switch_id}/programs/{program_id}. Both the switch ID and program ID are required path parameters to identify the target switch and the P4 program to be removed, respectively. Upon successful deletion, the server returns a 200 OK status code with an empty response body.

This API provides a straightforward mechanism for removing P4 programs from network switches. By specifying the unique identifiers for the switch and the P4 program, clients can request the deletion of the desired component. The server handles the removal process, ensuring that the P4 program is uninstalled and its associated resources are released. The 200 OK response indicates successful deletion without the need for additional information. This API follows REST principles by utilizing a clear URI, a standard HTTP method, and a well-defined response structure.

5.1.5 Programming Language and IDE Used

For the extended modules including P4-plugin and P4 Manager the Python programming language was used. We used Flask for developing the P4 Manager which requires the REST API to communicate with P4-plugin. We used P4 programming language for controlling packet forwarding

REST Resource	Operation	HTTP Action & Resource URI	Request Body Parameters	Response code	Response content
Switch Programs	Delete INC component (P4 program)	DELETE: /switches/switch_id/programs/program_id	None	200 (OK)	None

Table 5.2: Example of DELETE API exposed by the NIO to the NIIM

planes on bmv2 switches.

5.1.6 Experimental Setup

As illustrated in Figure 5.1, our prototype was deployed on a virtualized test environment hosted on a physical server located in the Montreal lab. The server was equipped with an Intel Xeon E5-2430 v2 processor operating at 2.50 GHz and 40 GB of RAM. A Windows Server 2012 operating system served as the host for the VirtualBox hypervisor, which managed three Ubuntu 20.04 virtual machines (VMs).

The first VM was configured with four virtual cores, 14 GB of RAM, and 85.9 GB of disk space. It served as the primary platform for the Open Source MANO (OSM) framework, modified to accommodate our specific requirements. The second VM, allocated two virtual cores and 4 GB of RAM, hosted the MicroK8s Kubernetes distribution using the containerd runtime. The third VM, with a single virtual core, 4 GB of RAM, and 34 GB of disk space, ran the P4 Manager, Mininet network emulator, and the BMv2 software switch.

For the network function virtualization (NFV) component, we leveraged the OpenLDAP K8s-based VNF (KNF) provided by OSM. The INF was implemented using a P4 program adapted from (3). The VNF and INF occupied 4.2 MB and 116.3 KB of disk space, respectively.

5.2 Performance Evaluation

In this section we commence with an exposition of the performance metrics employed, followed by a delineation of various test cases designed to evaluate these metrics. Finally, the section concludes with a presentation and analysis of the acquired results.

5.2.1 Proof-of-concept

For the prof-of-concept, we illustrate the deployment of the VNF and INF components by the screenshots. As depicted in figure 5.1, our prototype deployed on a virtualized test environment. In figure 5.3, we demonstrate the screenshots of the VNF and INF components deployment. We have the first VM (VM1) as a virtual core for running the main OSM application which is shown in 5.3, the second VM (VM2) is hosting the microk8s for deploying the VNFs which is shown in 5.4, and the third VM (VM3), which is shown in 5.5, hosting the Mininet topology and bmv2 switches provisioned for deploying INF on a switch. By running the CLI script in VM1, the VNF and INF are deployed on Microk8s in VM2 and the Mininet bmv2 switch in VM3 respectively. While the INF deployed on the bmv2 switch, the INF execution starts. In our implementation, we used a transcoder used in paper (3). This is a free software transcoder written in C as an extern function that we instantiated in the switch core through a P4 program.

For instance, in figure 5.4 we deploy a transcoder which is called *inc.p4* on switch 4 (s4). After receiving and compiling the *inc.p4* file, the *inc.p4.p4ifo.txt* is created which can be seen in the red box on the figure. According to (3), they show a network load reduction by executing a transcoder on a bmv2 switch using a network load monitoring graph. We use their network load monitor graph to show that the transcoder is executing correctly in our topology in addition showing the CLI logs of running transcoder on switch 4 (s4).

On the other hand, the figure 5.5 demonstrate the successful deployment of VNF which is an OpenLDAP by getting the list of pods running on the microk8s.

5.2.2 Performance Metric

5.2.2.1 Deployment delay

To assess our prototype's performance, we measured the time required to deploy a Network Service (NS) comprising four network functions. A holographic service consisting of encoder, decoder, transcoder, and renderer components (3). We use a transcoder as INF which is implemented as p4 and assume the VNF for the rest of the functions. In our evaluation, deployment delay is defined as the elapsed time in seconds (s) necessary to instantiate the Virtual Network Functions

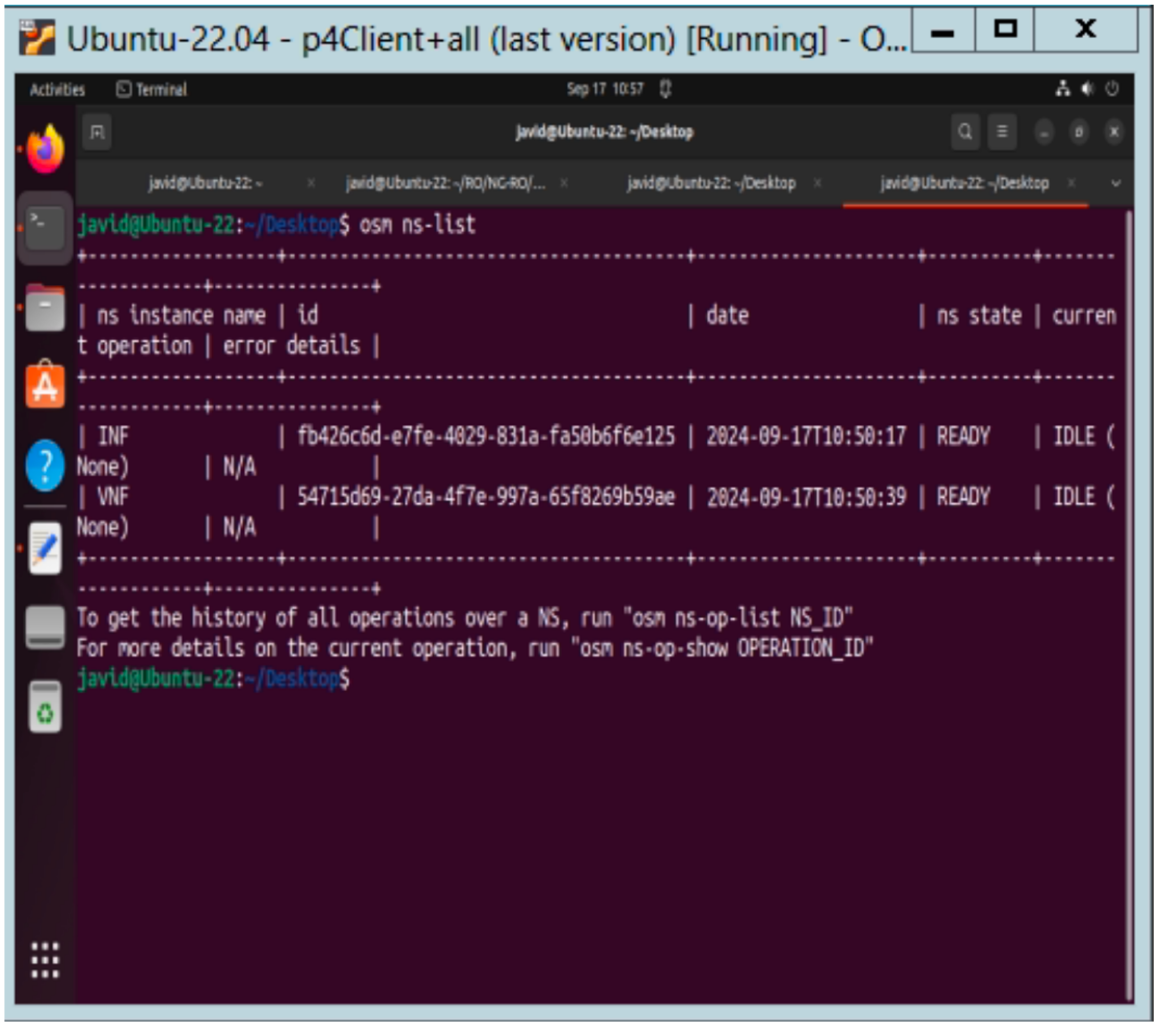

Figure 5.3: Screenshot of INF and VNF deployment

(VNFs) and In-Network Functions (INFs) constituting the hybrid NS.

5.2.3 Test Cases

We conducted five distinct test cases to evaluate our prototype. By modifying the composition of the hybrid Network Service (NS) through varying the number of Virtual Network Functions (VNFs) and In-Network Functions (INFs), we established five test scenarios: 4 VNFs, 3 VNFs and 1 INF, 2 VNFs and 2 INFs, 1 VNF and 3 INFs, and 4 INFs. Each VNF and INF was deployed on separate containers and P4 programmable switches in our prototype evaluation. For each test case, we calculated the average value from 20 measurements.

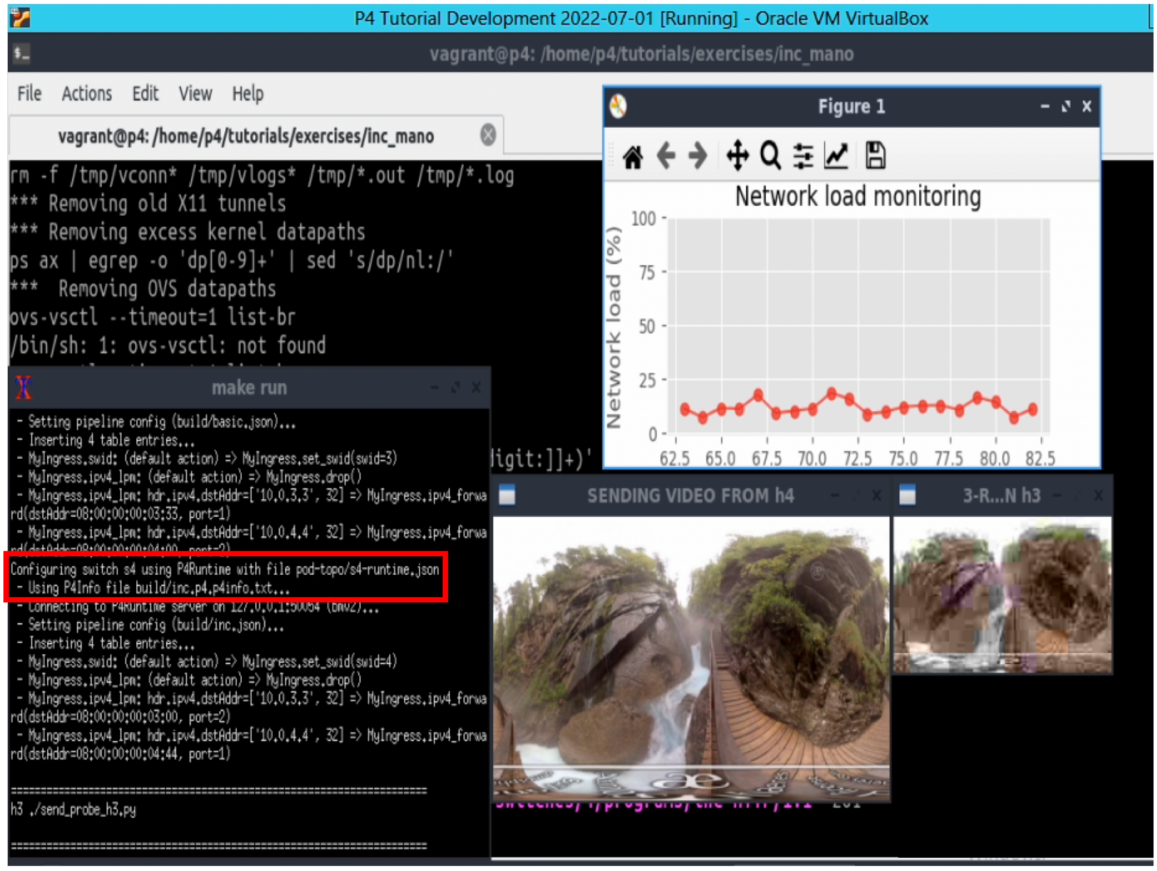

Figure 5.4: Screenshot of INF deployment on bmv2 switch in Mininet

- ***Test Case 1:*** This test case consider deployment of 4 KNF on the k8s cluster using as an edge server.
- ***Test Case 2:*** This test case consider deployment of 3 KNF on k8s cluster and deployment of 1 INF (as a transcoder) on BMv2 switch.
- ***Test Case 3:*** This test case consider deployment of 2 KNF on k8s cluster and deployment of 2 INF on BMv2 switches.
- ***Test Case 4:*** This test case consider deployment of 1 KNF on k8s cluster and deployment of 3 INF on BMv2 switches.
- ***Test Case 5:*** This test case consider deployment of 4 INF on BMv2 switches.

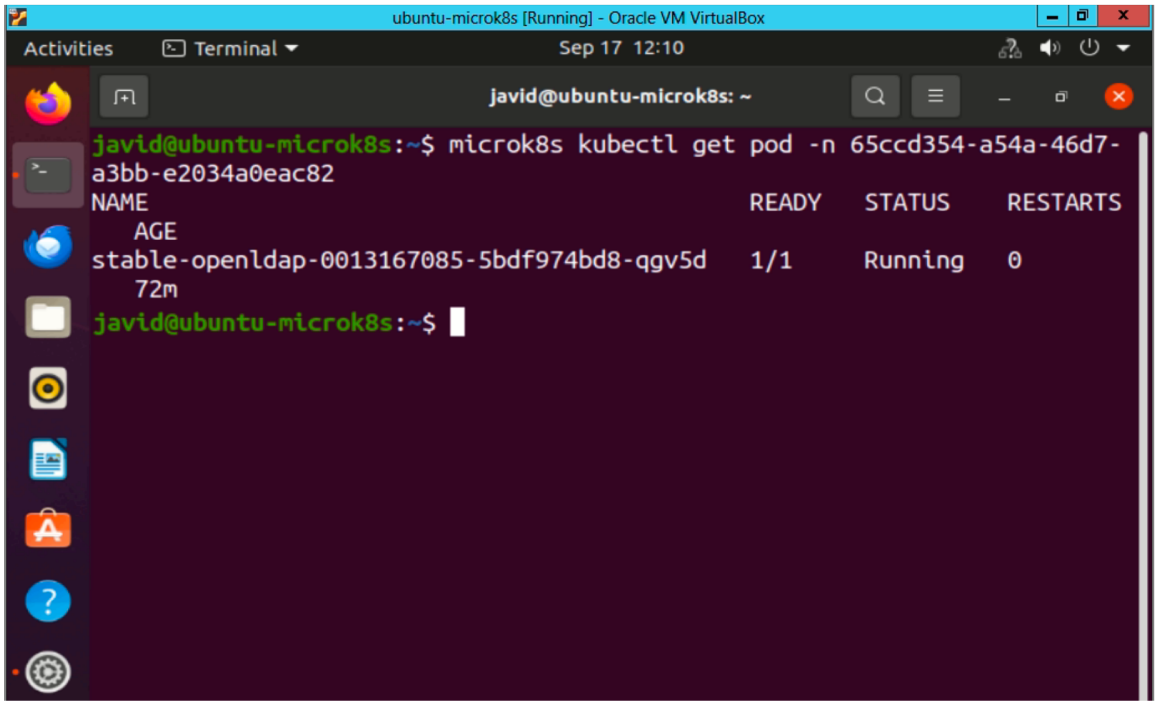

Figure 5.5: Screenshot of VNF deployment on Mikrok8s

5.2.4 Results and Analysis

As shown in Figure 5.6 and Table 5.3, the average deployment latency varies depending on the composition of the network service (i.e., the number of KNFs and INFs). Test case 1, which deploys 4 KNFs on a Kubernetes cluster, exhibits the highest average deployment latency of 66.90 seconds. This can be attributed to the overhead associated with deploying and managing containerized VNFs compared to in-network programmable functions.

In contrast, test cases with a higher proportion of INFs demonstrate significantly lower deployment latency. Test case 5, which deploys only INFs (4 in this case), achieves the lowest average deployment latency of 21.13 seconds. This represents a reduction of 68.42% compared to test case 1. This finding suggests that INFs are faster to deploy than KNFs, likely due to their simpler implementation and potentially lower resource requirements.

Test cases 2, 3, and 4 showcase the trend of decreasing deployment latency with an increasing number of INFs and a decreasing number of KNFs. Test case 2 (3 KNFs and 1 INF) achieves a 10.16% reduction in latency compared to test case 1. Test case 3 (2 KNFs and 2 INFs) reduces

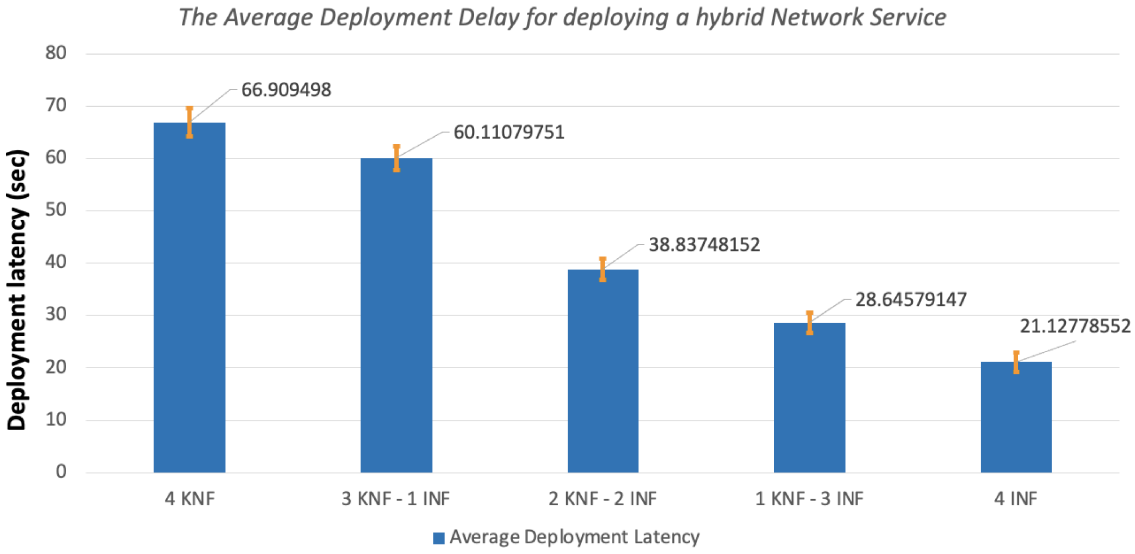

Figure 5.6: The Average Deployment Delay for deploying a hybrid Network Service

latency by 41.96% compared to test case 1. Finally, test case 4 (1 KNF and 3 INFs) demonstrates a 57.19% reduction in latency compared to test case 1.

Regarding the end to end latency, according to reference (3), as we take the same p4 transcoder as an INC component they used in their experiments, for transcoding in the network near the audience, the end to end latency is reduced compared to transcoding near the audience and on hosts.

These observations highlight the potential benefits of leveraging INFs for faster network service deployment. While KNFs offer flexibility and potential for complex service logic, their deployment overhead can introduce latency. INFs, on the other hand, provide a more lightweight and potentially faster alternative for specific network service functionalities.

The consistent and low standard deviation values across different configurations, as shown in Table 5.3, indicate stable performance in terms of latency across repeated tests. This reinforces the validity of the observed trends and suggests that the impact of the KNF/INF composition on deployment latency is statistically significant.

From an analytical point of view, the increasing deployment time when the number of VNFs is higher than INFs is due to the tools our prototype relies on to implement the NFV and INC resources in NII. We instantiate the K8s-based VNF on containers running on a lightweight production-grade K8s cluster like MicroK8s, whereas installing the P4 program (i.e., INF) on P4 switches emulated

Test Case	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5
Average Deployment Latency	66.909498	60.1107975	38.83748152	28.6457915	21.12778552
Standard Deviation	2.728128959	2.31931146	2.026029247	1.96878594	1.88851354

Table 5.3: The average deployment delay and standard deviation of deploying a network service

via Mininet and BMv2, which are not for production-grade. We would expect that the installation time of a P4 program on a real programmable switch (e.g., Tofino-based switch) takes longer than on an emulated P4 switch. Nevertheless, instantiating VNFs with standard K8s would take longer than with MicroK8s since the former provides a large-scale, multi-node production environment. The VNF instantiation time would further increase when using VMs (instead of containers) because they require the installation of a complete operating system. Further evaluation with production-grade NFV and INC resources is needed to analyze the time and performance when deploying hybrid NSs.

5.3 Conclusion

The prototyped architecture and implemented scenario was discussed in brief along with the softwares and programming languages used to develop it. Afterward, the performance metric and test cases were made. Finally, the result was shown and analyzed. We conclude our thesis in the next chapter by focusing on the summary of the thesis and the future work to be done.

Chapter 6

Conclusion

In this chapter, we will first summarize the contributions of this thesis and then focus on the possible future research direction

6.1 Contributions Summary

Holographic applications, such as holographic concerts, necessitate high-performance network infrastructure characterized by high bandwidth and low latency. Recent studies suggest that Hybrid NFV/INC networks are ideal for efficiently delivering these services. INC, a cutting-edge technology, enhances network efficiency by distributing computational tasks across network devices. However, integrating INC into existing infrastructure poses challenges related to lifecycle management and orchestration. Hybrid networks comprise VNFs and INCs. While the ETSI MANO framework supports application provisioning in NFV-enabled networks, it lacks capabilities for INC-based environments. Consequently, a new MANO architecture is required to accommodate applications that leverage both VNFs and INCs.

The hybrid management module was one of the novel requirements. The ability of deploying both VNF and INC component is essential in provisioning holographic application. The hybrid orchestration module is the next requirement. The holographic application is composed of a chain of network functions for instance encoder as VNF, transoder as INC component, and decoder and

renderer as VNF. For deploying these components as one Network Service there should be an orchestrator to distinguish these function and decide which management module should deploy them on the targeted network. We also required the hybrid infrastructure and infrastructure manager for provisioning the deployment of the hybrid network service which is another requirement. Based on the requirements we evaluated the existing NFV MANO architecture and frameworks. None of the state-of-the-art was able to fulfill all the requirements.

We then proceeded to propose a hybride NFV/INC MANO architecture that can fulfill the derived requirements. In this regard, we proposed a novel NFV MANO architecture for hybrid NFV/INC systems. In contrast to the existing MANO solutions, the proposed solution enables the management and orchestration of the deployment of both VNF and INC components in the network. We also proposed all the required interfaces between each two modules. In the end, we show the deployment procedures of VNF and INC components respectively.

Then, a prototype for validating the hybride NFV/INC MANO was implemented using the OSM. A subset of the proposed architecture was implemented. Some extensions of the OSM was made to enable the fulfillment of all the requirements. Namely, the INF Management module, the INF infrastructure manager and the INC-enabled infrastructure. Also a REST interface is used to make the communication between the INF Management and INF infrastructure manager.

Finally, a performance metric and experimental setup was defined. A set of experiments are conducted to evaluate the feasibility of the architecture and explore possible options for deployinf VNF and INC components. The results from the experiment were shown and analyzed. The results show the lower deployment delay as we increase the number of INC components in a Network Service. These experiments conducted that provisioning holographic applications over a hybrid NFV/INC setting is the best choice.

6.2 Future Research Direction

Our future research aims to expand the capabilities of our existing prototype to fully realize the proposed NFV/INC MANO architecture and facilitate the complete deployment of hybrid network services. This expansion will involve creating a new INF Descriptor (INFD) template and modifying

the existing Network Service Descriptor (NSD) template and OSM's LCM to accommodate the onboarding and instantiation of network services composed of both VNFs and INFs.

Additionally, we intend to enhance the P4 modules, such as the manager and plugin, to effectively manage P4 controllers. This will enable the configuration of match-action tables to seamlessly interconnect VNFs and INFs, thereby implementing the desired service-chaining topology of the hybrid network service. To ensure the practical viability of our hybrid NFV/INC infrastructure, we plan to establish testing environments utilizing production-grade resources. This will allow us to rigorously evaluate the performance and efficiency of deploying hybrid network services in real-world scenarios.

In this thesis, we proposed a hybrid NFV/INC MANO architecture and implement the deployment and deletion procedures of hybrid network service composed of VNFs and INFs. We did not cover sending flow through the components or executing a chain because it is not in the scope of the thesis. We simplify the implementation environment to make sure there is enough isolation that if there is latency is only due to the NS deployment and different aspects like the ip address or bridging do not interfere the latency issues and our results. So executing a chain or sending flow through the components is possible to do and can be done in the future work.

References

A. Clemm, M. T. Vega, H. K. Ravuri, T. Wauters, and F. De Turck, "Toward truly immersive holographic-type communication: Challenges and solutions," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 93–99, 2020.

ITU, "Functional requirements of e2e network platforms to enhance the delivery of cloud-vr services over integrated broadband cable networks," https://www.itu.int/rec/T-REC-J.1631/en, 2021, accessed: 2021-11.

F. Aghaaliakbari, Z. A. Hmitti, M. Rayani, M. Gherari, R. H. Glitho, H. Elbiaze, and W. Ajib, "An architecture for provisioning in-network computing-enabled slices for holographic applications in next-generation networks," *IEEE Communications Magazine*, vol. 61, no. 3, pp. 52–58, 2023.

A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computing is a dumb idea who's time has come," *Proceedings of Hot-Nets*, 2017.

D. R. Ports and J. Nelson, "When should the network be the computer?" in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2019, pp. 209–215.

E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," *IEEE Access*, vol. 9, pp. 87 094–87 155, 2021.

L. Mamushiane, A. A. Lysko, T. Mukute, J. Mwangama, and Z. Du Toit, "Overview of 9 open-source resource orchestrating etsi mano compliant implementations: A brief survey," in *2019 IEEE 2nd Wireless Africa Conference (WAC)*. IEEE, 2019, pp. 1–7.

H. Cao, "Network function virtualization," in *Software Defined Internet of Everything*. Springer, 2021, pp. 135–143.

N. ETSI, "Network function virtualization (nfv) management and orchestration: Gs nfv-man 001 v1. 2.1," 2021.

J. van der Hooft, M. T. Vega, T. Wauters, C. Timmerer, A. C. Begen, F. De Turck, and R. Schatz, "From capturing to rendering: Volumetric media delivery with six degrees of freedom," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 49–55, 2020.

J. Karafin and B. Bevensee, "25-2: On the support of light field and holographic video display technology," in *SID Symposium Digest of Technical Papers*, vol. 49, no. 1. Wiley Online Library, 2018, pp. 318–321.

P. A. Kara, A. Cserkaszky, M. G. Martini, A. Barsi, L. Bokor, and T. Balogh, "Evaluation of the concept of dynamic adaptive streaming of light field video," *IEEE Transactions on Broadcasting*, vol. 64, no. 2, pp. 407–421, 2018.

S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li *et al.*, "Emerging mpeg standards for point cloud compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2018.

ACMSigarch, "In-network computing," ACMSigarch, Tech. Rep., 2019.

A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, and T. Gayraud, "Software-defined networking: Challenges and research opportunities for future internet," *Computer Networks*, vol. 75, pp. 453–471, 2014.

M. He, A. Basta, A. Blenk, N. Deric, and W. Kellerer, "P4nfv: An nfv architecture with flexible data plane reconfiguration," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 90–98.

T. Mai, H. Yao, S. Guo, and Y. Liu, "In-network computing powered mobile edge: Toward high performance industrial iot," *IEEE network*, vol. 35, no. 1, pp. 289–295, 2020.

Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–16.

ETSI. Network functions virtualisation (nfv) release 4; management and orchestration; functional requirements specification. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/010/04.05.01_60/gs_NFV-IFA010v040501p.pdf

ETSI OSM. (2024) What is OSM? Open Source MANO. [Online]. Available: https://osm.etsi.org

L. Foundation. Onap–open network automation platform. [Online]. Available: https://www.onap.org/

F. Z. Yousaf, V. Sciancalepore, M. Liebsch, and X. Costa-Perez, "Manoaas: A multi-tenant nfv mano for 5g network slices," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 103–109, 2019.

R. Riggio, S. N. Khan, T. Subramanya, I. G. B. Yahia, and D. Lopez, "Lightmano: Converging nfv and sdn at the edges of the network," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

P4Lang, "Behavioral model (bmv2)," GitHub. [Online]. Available: https://github.com/p4lang/behavioral-model

ETSI OSM, "OSM usage," Open Source MANO, 2020. [Online]. Available: https://osm.etsi.org/docs/user-guide/latest/05-osm-usage.html

Canonical Ltd., "Microk8s documentation," MicroK8s, 2024. [Online]. Available: https://microk8s.io/docs